

# Smart Card Handbook

Fourth Edition

# **Smart Card Handbook**

**Fourth Edition**

**Wolfgang Rankl and Wolfgang Effing**  
*Giesecke & Devrient GmbH, Germany*

Translated by  
**Kenneth Cox**  
*Kenneth Cox Technical Translations, Wassenaar, The Netherlands*



A John Wiley and Sons, Ltd., Publication

First published under the title *Handbuch der Chipkarten: Fünfte Edition* by Carl Hanser Verlag  
© 2008 Carl Hanser Verlag, Munich/FRG

This edition first published 2010  
© 2010, John Wiley & Sons, Ltd

First edition published 1997  
Second edition published 2000  
Third edition published 2003

*Registered office*

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at [www.wiley.com](http://www.wiley.com).

The right of the authors to be identified as the author of this work has been asserted in accordance with the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except as permitted by the UK Copyright, Designs and Patents Act 1988, without the prior permission of the publisher.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The publisher is not associated with any product or vendor mentioned in this book. This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold on the understanding that the publisher is not engaged in rendering professional services. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

*Library of Congress Cataloging-in-Publication Data*

Rankl, W. (Wolfgang)  
[Handbuch der Chipkarten. English]  
Smart card handbook / Wolfgang Rankl. – 4th ed.  
p. cm.  
Includes bibliographical references and index.  
ISBN 978-0-470-74367-6 (cloth)  
1. Smart cards—Handbooks, manuals, etc. I. Title.  
TK7895.S62R3613 2010  
004.5'6—dc22  
2009052095

A catalogue record for this book is available from the British Library.

ISBN 978-0-470-74367-6 (Hbk)

Typeset in 10/12pt Times by Aptara Inc., New Delhi, India  
Printed in Singapore by Markono

# Contents

<b>Preface to the Fourth Edition</b>	xxiii
<b>Symbols and Notation</b>	xxv
<b>Abbreviations</b>	xxix
<b>1 Introduction</b>	1
1.1 The history of smart cards	2
1.2 Card types and applications	7
1.2.1 Memory cards	8
1.2.2 Processor cards	8
1.2.3 Contactless cards	9
1.3 Standardization	10
<b>2 Card Types</b>	15
2.1 Embossed cards	15
2.2 Magnetic-stripe cards	16
2.3 Smart cards	18
2.3.1 Memory cards	20
2.3.2 Contactless memory cards	20
2.3.3 Processor cards	21
2.3.4 Contactless processor cards	23
2.3.5 Multi-megabyte cards	24
2.3.6 Security tokens	25
2.4 Optical memory cards	25

---

<b>3 Physical Properties</b>	29
3.1 Card formats	29
3.2 Contact field	36
3.3 Card body	38
3.4 Card materials	39
3.5 Card components and security features	42
3.5.1 Guilloche patterns	42
3.5.2 Signature panel	44
3.5.3 Microtext	44
3.5.4 Ultraviolet text	44
3.5.5 Barcode	44
3.5.6 Hologram	45
3.5.7 Kinogram	45
3.5.8 Multiple Laser Image (MLI)	46
3.5.9 Embossing	46
3.5.10 Laser engraving	47
3.5.11 Scratch field	47
3.5.12 Thermochrome display	48
3.5.13 Moduliertes Merkmal (modulated feature) method	48
3.5.14 Security features	49
3.6 Chip modules	50
3.6.1 Electrical connections between the chip and the module	51
3.6.2 TAB modules	53
3.6.3 Chip-on-flex modules	54
3.6.4 Lead-frame modules	57
3.6.5 Special modules	59
<b>4 Electrical Properties</b>	61
4.1 Electrical connections	62
4.2 Supply voltage	62
4.3 Supply current	65
4.4 Clock supply	69
4.5 Data transmission with T = 0 or T = 1	69
4.6 Activation and deactivation sequences	70
<b>5 Smart Card Microcontrollers</b>	73
5.1 Semiconductor technology	76
5.2 Processor types	79
5.3 Memory types	82
5.3.1 ROM (read-only memory)	84
5.3.2 EEPROM (erasable read-only memory)	85
5.3.3 EEPROM (electrically erasable read-only memory)	85
5.3.4 Flash memory	90
5.3.5 RAM (random-access memory)	92
5.3.6 FRAM (ferroelectric random-access memory)	92
5.4 Supplementary hardware	93
5.4.1 Communication with T = 0 or T = 1	93

---

5.4.2 Communication with USB	94
5.4.3 Communication with MMC	95
5.4.4 Communication with SWP	95
5.4.5 Communication with I <sup>2</sup> C	96
5.4.6 Timer	96
5.4.7 CRC (cyclic redundancy check) calculation unit	97
5.4.8 Random number generator (RNG)	97
5.4.9 Clock generation and clock multiplication	98
5.4.10 DMA (direct memory access)	99
5.4.11 Memory management unit (MMU)	100
5.4.12 Java accelerator	101
5.4.13 Coprocessor for symmetric cryptographic algorithms	102
5.4.14 Coprocessor for asymmetric cryptographic algorithms	103
5.4.15 Error detection and correction for nonvolatile memory	103
5.4.16 Mass memory interface	104
5.4.17 Multichip module	105
5.4.18 Vertical system integration (VSI)	106
5.5 Extended temperature range	107
<b>6 Information Technology Foundations</b>	109
6.1 Data structures	109
6.2 Encoding alphanumeric data	115
6.2.1 Seven-bit code (ASCII)	115
6.2.2 Eight-bit code (PC ASCII)	115
6.2.3 Sixteen-bit code (Unicode)	116
6.2.4 Thirty-two-bit code (UCS)	116
6.3 SDL notation	117
6.4 State machines	118
6.4.1 Basic theory of state machines	118
6.4.2 Practical applications	120
6.5 Error detection and correction codes	122
6.5.1 XOR checksums	124
6.5.2 CRC checksums	125
6.5.3 Reed–Solomon codes	127
6.5.4 Error correction codes	128
6.6 Data compression	129
<b>7 Security Foundations</b>	133
7.1 Cryptology	133
7.1.1 Symmetric cryptographic algorithms	138
7.1.1.1 DES algorithm	138
7.1.1.2 AES algorithm	140
7.1.1.3 IDEA algorithm	141
7.1.1.4 COMP128 algorithms	142
7.1.1.5 Milenage algorithm	142
7.1.1.6 Operating modes of block encryption algorithms	142
7.1.1.7 Multiple encryption	144

---

7.1.2 Asymmetric cryptographic algorithms	145
7.1.2.1 RSA algorithm	146
7.1.2.2 Generating RSA keys	148
7.1.2.3 DSS algorithm	151
7.1.2.4 Elliptic curves as asymmetric cryptographic algorithms	152
7.1.3 Padding	154
7.1.4 Message authentication code and cryptographic checksum	155
7.2 Hash functions	156
7.3 Random numbers	159
7.3.1 Generating random numbers	160
7.3.2 Testing random numbers	163
7.4 Authentication	166
7.4.1 Unilateral symmetric authentication	168
7.4.2 Mutual symmetric authentication	169
7.4.3 Static asymmetric authentication	170
7.4.4 Dynamic asymmetric authentication	172
7.5 Digital signatures	174
7.6 Certificates	178
7.7 Key management	180
7.7.1 Derived keys	181
7.7.2 Key diversification	182
7.7.3 Key versions	182
7.7.4 Dynamic keys	182
7.7.4.1 Generation with a symmetric cryptographic algorithm	182
7.7.4.2 Generation with an asymmetric cryptographic algorithm	183
7.7.5 Key data	183
7.7.6 Key management example	185
7.8 Identification of persons	187
7.8.1 Knowledge-based identification	188
7.8.2 Testing a secret number	188
7.8.3 The probability of guessing a PIN	190
7.8.4 Generating PIN codes	191
7.8.5 Verifying that a terminal is genuine	192
7.8.6 Biometric methods	194
<b>8 Communication with Smart Cards</b>	201
8.1 Answer to reset (ATR)	203
8.1.1 The initial character	206
8.1.2 The format character	207
8.1.3 The interface characters	207
8.1.3.1 Global interface character $TA_1$	208
8.1.3.2 Global interface character $TA_i$	209
8.1.3.3 Global interface character $TC_1$	209
8.1.3.4 Specific interface character $TC_2$	210
8.1.3.5 Specific interface character $TA_i$ ( $i > 2$ )	210
8.1.3.6 Specific interface character $TB_i$ ( $i > 2$ )	210

---

8.1.3.7 Specific interface character $TC_i$ ( $i > 2$ )	211
8.1.3.8 Global interface character $TA_2$	211
8.1.4 The historical characters	211
8.1.5 The check character	214
8.1.6 Practical examples of ATRs	214
8.2 Protocol Parameter Selection (PPS)	217
8.3 Message structure: APDUS	221
8.3.1 Command APDU structure	221
8.3.2 Response APDU structure	224
8.4 Secure Data Transmission	225
8.4.1 Data objects for plaintext	227
8.4.2 Data objects for security mechanisms	227
8.4.3 Data objects for auxiliary functions	228
8.4.4 The authentic mode procedure	228
8.4.5 The combined mode procedure	230
8.4.6 Send sequence counter	231
8.5 Logical channels	233
8.6 Logical protocols	234
8.6.1 TCP/IP protocol	234
8.6.2 HTTP protocol	235
8.6.3 Bearer Independent Protocol (BIP)	236
8.7 Connecting terminals to higher-level systems	237
8.7.1 PC/SC	237
8.7.1.1 ICC-aware application	239
8.7.1.2 Service provider	239
8.7.1.3 ICC resource manager	240
8.7.1.4 IFD handler	240
8.7.1.5 IFD (interface device)	240
8.7.1.6 ICC (integrated chip card)	241
8.7.2 OCF	241
8.7.3 MKT	241
8.7.4 MUSCLE	242
<b>9 Data Transmission with Contact Cards</b>	243
9.1 Physical transmission layer	243
9.2 Memory card protocols	248
9.2.1 Telephone chip protocol	249
9.2.1.1 Resetting the address pointer	249
9.2.1.2 Incrementing the address pointer and reading data	250
9.2.1.3 Writing to an address	250
9.2.1.4 Erasing bytes	250
9.2.2 I <sup>2</sup> C bus	251
9.2.2.1 Reading from an address	252
9.2.2.2 Writing to an address	253
9.3 ISO transmission protocols	254
9.3.1 The T = 0 transmission protocol	255
9.3.2 The T = 1 transmission protocol	260

---

9.3.2.1	Block structure	261
9.3.2.2	Send/receive sequence counter	264
9.3.2.3	Waiting times	265
9.3.2.4	Transmission protocol mechanisms	267
9.3.2.5	Example of data transmission with the T = 1 protocol	270
9.3.3	Comparison of the T = 0 and T = 1 transmission protocols	270
9.3.4	The T = 14 transmission protocol (Germany)	271
9.4	USB transmission protocol	272
9.4.1	Electrical connection	273
9.4.2	Logical connection	274
9.4.2.1	Transfer modes	275
9.4.2.2	Data packets	275
9.4.3	Device classes	276
9.4.4	Summary and prospects	277
9.5	MMC transmission protocol	277
9.6	Single-wire protocol (SWP)	278
<b>10</b>	<b>Contactless Data Transmission</b>	283
10.1	Inductive coupling	284
10.2	Power transmission	285
10.3	Data transmission	286
10.4	Capacitive coupling	287
10.5	Collision avoidance	289
10.6	State of standardization	290
10.7	Close-coupling cards (ISO/IEC 10536)	291
10.7.1	Power transmission	292
10.7.2	Inductive data transmission	293
10.7.2.1	Transmission from the card to the terminal	293
10.7.2.2	Transmission from the terminal to the card	293
10.7.3	Capacitive data transmission	295
10.8	Remote coupling cards	296
10.9	Proximity cards (ISO/IEC 14443)	297
10.9.1	Physical properties	298
10.9.2	Power transmission and signal interface	299
10.9.3	Signal and communication interface	299
10.9.4	Type A communication interface	300
10.9.5	Type B communication interface	302
10.9.5.1	Data transmission from the terminal to the card	302
10.9.5.2	Data transmission from the card to the terminal	303
10.9.6	Initialization and anticollision (ISO/IEC 14443-3)	304
10.9.6.1	Type A initialization and anticollision	305
10.9.6.2	Type B initialization and anticollision	314
10.9.7	Transmission protocol (ISO/IEC 14443-4)	329
10.9.7.1	Protocol activation with Type A cards	330
10.9.7.2	Half-duplex block protocol (ISO/IEC 14443-4)	339
10.9.7.3	Deactivating a card	344
10.9.7.4	Error handling	344

10.10 Vicinity integrated circuit cards (ISO/IEC 15693)	344
10.11 Near field communication (NFC)	348
10.11.1 State of standardization	348
10.11.2 NFC protocol	349
10.11.3 NFC applications	350
10.11.3.1 Rapid access to information regarding services	350
10.11.3.2 Peer-to-peer information exchange	350
10.11.3.3 Mobile payment	350
10.11.3.4 Secure NFC	351
10.12 FeliCa	352
10.13 Mifare	352
<b>11 Smart Card Commands</b>	<b>353</b>
11.1 File selection commands	356
11.2 Read and write commands	358
11.3 Search commands	366
11.4 File operation commands	368
11.5 Commands for authenticating persons	370
11.6 Commands for authenticating devices	374
11.7 Commands for cryptographic algorithms	378
11.8 File management commands	384
11.9 Application management commands	389
11.10 Completion commands	391
11.11 Commands for hardware testing	395
11.12 Commands for data transmission	398
11.13 Database commands (SCQL)	399
11.14 Commands for electronic purses	402
11.15 Commands for credit and debit cards	405
11.16 Application-specific commands	406
11.17 Command processing times	407
11.17.1 Processing time estimation	407
11.17.1.1 Command processing	408
11.17.1.2 Proportionality factor for predefined functions	409
11.17.1.3 NVM operations	409
11.17.1.4 Data transfer	410
11.17.1.5 Calculated example: READ BINARY command	411
11.17.1.6 Calculated example: smart card initialization	413
11.17.2 Processing times of typical smart card commands	415
11.17.3 Typical command processing times	417
<b>12 Smart Card File Management</b>	<b>421</b>
12.1 File structure	421
12.2 The life cycle of files	422
12.3 File types	423
12.3.1 Master file (MF)	424
12.3.2 Dedicated file (DF)	424
12.3.3 Application dedicated file (ADF)	425

---

12.3.4 Elementary file (EF)	425
12.3.5 Working EF (WEF)	425
12.3.6 Internal EF (IEF)	425
12.4 Application files	425
12.5 File names	426
12.5.1 File identifier (FID)	426
12.5.2 Short file identifier (SFI)	428
12.5.3 DF name	429
12.5.4 Application identifier (AID) structure and coding	429
12.6 File selection	430
12.6.1 Selecting directories (MF and DF)	430
12.6.2 Explicit EF selection	431
12.6.3 Implicit EF selection	431
12.6.4 Selection using a path name	432
12.7 EF file structures	432
12.7.1 Transparent file structure	432
12.7.2 Linear fixed file structure	433
12.7.3 Linear variable file structure	434
12.7.4 Cyclic file structure	435
12.7.5 Data objects file structure	435
12.7.6 Database file structure	436
12.7.7 Execute structure	436
12.7.8 Sequence control file structure	436
12.8 File access conditions	436
12.9 File attributes	438
12.9.1 WORM attribute	439
12.9.2 High update activity attribute	439
12.9.3 EDC utilization attribute	439
12.9.4 Atomic write access attribute	439
12.9.5 Concurrent access attribute	440
12.9.6 Data transfer selection attribute	440
12.9.7 File encryption attribute	440
<b>13 Smart Card Operating Systems</b>	441
13.1 Evolution of smart card operating systems	442
13.2 Fundamental aspects and tasks	444
13.3 Command processing	447
13.4 Design and implementation principles	449
13.5 Operating system completion	452
13.5.1 Operating system boot loader	455
13.5.2 Hardware recognition	455
13.5.3 Soft and hard masks	456
13.5.4 Operating system APIs	457
13.6 Memory organization and memory management	457
13.6.1 RAM memory management	458
13.6.2 EEPROM memory management	459
13.6.3 Flash memory management	461

---

13.7 File management	463
13.7.1 Pointer-based file management	464
13.7.2 File management with a file allocation table (FAT)	466
13.7.3 Memory partitioning into pages	467
13.7.4 DF separation	467
13.7.5 Free memory management mechanisms	468
13.7.6 Quota mechanism	470
13.7.7 Data integrity	471
13.7.8 Cross-application access	472
13.8 Sequence control	472
13.9 ISO/IEC 7816-9 resource access	474
13.10 Atomic operations	480
13.11 Multitasking	483
13.12 Performance	484
13.13 Application management with global platform	485
13.13.1 Security domains	487
13.13.2 Issuer security domain	488
13.13.3 Global platform API	489
13.13.4 Global platform commands	489
13.14 Downloadable program code	491
13.15 Executable native code	493
13.16 Open platforms	499
13.16.1 ISO/IEC 7816 compatible platforms	499
13.16.2 Java card	499
13.16.2.1 The Java programming language	500
13.16.2.2 The properties of Java	501
13.16.2.3 Java virtual machine (JVM)	502
13.16.2.4 Java card virtual machine (JCVM)	504
13.16.2.5 Memory sizes in Java cards	505
13.16.2.6 Performance in Java cards	507
13.16.2.7 Java card runtime environment	508
13.16.2.8 Application partitioning (firewalls)	508
13.16.2.9 Command dispatching and application selection (dispatcher)	509
13.16.2.10 Transaction integrity (atomic operations)	510
13.16.2.11 Persistent and transient objects	510
13.16.2.12 Java Card application programming interface	511
13.16.2.13 Software development for Java in smart cards	514
13.16.2.14 Execution speed	517
13.16.2.15 File system	517
13.16.2.16 Cryptography and export restrictions	518
13.16.2.17 Future generations of Java cards	518
13.16.2.18 Summary and future prospects	519
13.16.3 Multos	519
13.16.4 BasicCard	520
13.16.5 Linux	521

---

13.17 The small-OS smart card operating system	521
13.17.1 Programming in pseudocode	523
13.17.2 Design aspects	524
13.17.3 File access	526
13.17.4 Access to internal secrets (PINs and keys)	526
13.17.5 Small-OS constants	529
13.17.6 Small-OS variables	529
13.17.6.1 Small-OS RAM variables	531
13.17.6.2 Small-OS EEPROM variables	532
13.17.7 Main loop and initialization	534
13.17.8 I/O manager	536
13.17.9 File manager	536
13.17.10 Return code manager	536
13.17.11 Operating system kernel	538
13.17.12 Command interpreter	539
13.17.13 Structure of program code for commands	540
13.17.14 Command set	541
13.17.14.1 SELECT command	541
13.17.14.2 READ BINARY command	545
13.17.14.3 UPDATE BINARY command	547
13.17.14.4 READ RECORD command	549
13.17.14.5 UPDATE RECORD command	552
13.17.14.6 VERIFY command	556
13.17.14.7 INTERNAL AUTHENTICATE command	560
13.17.15 A simple application example	563
<b>14 Smart Card Production</b>	567
14.1 Tasks and roles in the production process	567
14.2 The smart card life cycle	569
14.3 Chip and module production	571
14.3.1 Chip design	572
14.3.2 Smart card operating system development	573
14.3.3 Chip fabrication in semiconductor plants	575
14.3.4 Chip testing on the wafer	578
14.3.5 Wafer sawing	579
14.3.6 Packaging chips in modules	581
14.3.7 Chip bonding	582
14.3.8 Encapsulating the chips in modules	583
14.3.9 Module testing	583
14.4 Card Body production	585
14.4.1 Monolayer card	585
14.4.2 Multilayer card	586
14.4.3 Injection-molded card bodies	586
14.4.4 Direct plug-in production (plug-in only)	588
14.4.5 Card bodies with integrated antennas	589
14.4.5.1 Etched antennas	590
14.4.5.2 Wound coils	591

---

14.4.5.3	Embedded antennas	591
14.4.5.4	Printed antennas	593
14.4.5.5	Connecting the antenna to the chip	593
14.4.6	Printing the card bodies	594
14.4.6.1	Sheet printing of card bodies	594
14.4.6.2	Printing single card bodies	595
14.4.6.3	Offset printing	596
14.4.6.4	Digital printing	597
14.4.6.5	Screen printing	598
14.4.6.6	Thermal transfer and thermal dye sublimation printing	598
14.4.6.7	Inkjet printing	599
14.4.7	Stamping the foils	599
14.4.8	Applying card components to the card body	599
14.5	Combining the card body and the chip	599
14.5.1	Milling the module cavity	600
14.5.2	Implanting the modules	601
14.5.3	Module printing	603
14.5.4	Plug-in stamping	604
14.6	Electrical testing of modules	605
14.7	Loading static data	609
14.7.1	Completing the operating system	609
14.7.2	Collaboration of the card producer and the card issuer	610
14.7.3	Initializing the application	612
14.7.4	Optimized mass data transfer to smart cards	613
14.7.5	Accelerating data transfer to the smart card	616
14.8	Loading individual data	618
14.8.1	Generating card-specific secret data	618
14.8.2	Personalization (individualization)	619
14.9	Envelope stuffing and dispatching	624
14.10	Special types of production	625
14.10.1	Production on demand (PoD)	625
14.10.2	Picture cards	626
14.10.3	Direct smart card issuing (instant issuing)	628
14.11	Termination of card usage	629
14.11.1	Deactivation	629
14.11.2	Recycling	630
<b>15</b>	<b>Quality Assurance</b>	633
15.1	Card body tests	634
15.1.1	Adhesion (or blocking)	635
15.1.2	Amplitude measurement	635
15.1.3	Bending stiffness	635
15.1.4	Card dimensional stability and warpage with temperature and humidity	636
15.1.5	Card dimensions	636
15.1.6	Card warpage	636
15.1.7	Delamination	636

---

15.1.8	Dynamic bending stress	636
15.1.9	Dynamic torsional stress	637
15.1.10	Electrical resistance and impedance of contacts	637
15.1.11	Electromagnetic fields	638
15.1.12	Embossing relief height of character	638
15.1.13	Flammability	638
15.1.14	Flux transition spacing variation	638
15.1.15	Height and surface profile of the magnetic stripe	638
15.1.16	Light transmittance	638
15.1.17	Location of contacts	639
15.1.18	Resistance to chemicals	639
15.1.19	Static electricity	639
15.1.20	Surface profile of contacts	639
15.1.21	Surface roughness of the magnetic stripe	640
15.1.22	Ultraviolet light	640
15.1.23	Vibration	640
15.1.24	Wear test for magnetic stripe	640
15.1.25	X-rays test	640
15.2	Microcontroller hardware tests	641
15.3	Test methods for contactless smart cards	642
15.3.1	Test methods for proximity smart cards	644
15.3.2	Test methods for vicinity coupling smart cards	645
15.4	Test methods for software	645
15.4.1	Fundamentals of smart card software testing	647
15.4.1.1	Analysis	648
15.4.1.2	Design	648
15.4.1.3	Implementation and test	648
15.4.1.4	System integration	648
15.4.1.5	Maintenance	648
15.4.2	Testing techniques and test strategies	649
15.4.2.1	Statistical program evaluation	649
15.4.2.2	Review	649
15.4.2.3	Blackbox test	649
15.4.2.4	Whitebox test	650
15.4.2.5	Greybox test	653
15.4.3	Dynamic testing of operating systems and applications	653
15.4.4	Test strategy	654
15.5	Evaluation of hardware and software	659
15.5.1	Common criteria (CC)	661
15.5.2	ZKA criteria	663
15.5.3	Additional evaluation methods	663
15.5.4	Summary	664
<b>16</b>	<b>Smart Card Security</b>	667
16.1	Classification of attacks and attackers	668
16.1.1	Classification of attacks	669

---

16.1.2 Consequences of attacks and classification of attackers	672
16.1.3 Classification of the attractiveness of attacks	674
16.2 A history of attacks	675
16.3 Attacks and defense measures during development	675
16.3.1 Smart card microcontroller development	679
16.3.2 Smart card operating system development	680
16.4 Attacks and defense measures during production	682
16.5 Attacks and defense measures during card usage	682
16.5.1 Attacks on the hardware	684
16.5.2 Attacks on the operating system	712
16.5.3 Attacks on applications	727
16.5.4 Attacks on the system	731
<b>17 Smart Card Terminals</b>	<b>735</b>
17.1 Mechanical properties	739
17.1.1 Contact unit with wiping contacts	739
17.1.2 Mechanically driven contact unit	740
17.1.3 Electrically driven contact unit	740
17.1.4 Card ejection	741
17.1.5 Ease of card withdrawal	742
17.2 Electrical properties	742
17.3 User interface	744
17.4 Application interface	744
17.5 Security	744
<b>18 Smart Cards in Payment Systems</b>	<b>747</b>
18.1 Payment transactions with cards	747
18.1.1 Electronic payment transactions with smart cards	748
18.1.1.1 Credit cards	748
18.1.1.2 Debit cards	749
18.1.1.3 Electronic purses	749
18.1.1.4 Open and closed system architectures	750
18.1.1.5 Centralized and decentralized system architecture	751
18.1.2 Electronic money	753
18.1.2.1 Processable	753
18.1.2.2 Transferable	753
18.1.2.3 Divisible	753
18.1.2.4 Decentralized	753
18.1.2.5 Monitorable	754
18.1.2.6 Secure	754
18.1.2.7 Anonymous	754
18.1.2.8 Legal framework and retention of value	755
18.1.3 Basic system architecture options	755
18.1.3.1 Background system	755
18.1.3.2 Network	755
18.1.3.3 Terminals	756
18.1.3.4 Smart cards	756

18.2 Prepaid memory cards	757
18.3 Electronic purses	759
18.3.1 Inter-sector electronic purses compliant with EN 1546	760
18.3.1.1 Data elements	763
18.3.1.2 Files	764
18.3.1.3 Commands	764
18.3.1.4 States	765
18.3.1.5 Cryptographic algorithms	766
18.3.1.6 General processes	767
18.3.1.7 Load process	768
18.3.1.8 Payment process	771
18.3.2 Common electronic parse specifications	774
18.3.3 Proton	775
18.4 EMV Application	776
18.4.1 Files and data elements	777
18.4.2 Commands	778
18.4.3 Cryptography	778
18.4.4 System architecture and transaction processes	779
18.4.5 Future developments	781
18.5 PayPass and payWave	782
18.6 The Eurocheque System in Germany	783
18.6.1 User functions	784
18.6.2 The overall system in brief	785
18.6.3 Girocard with chip	786
18.6.4 Supplementary applications	788
18.6.5 Summary	788
<b>19 Smart Cards in Telecommunication Systems</b>	789
19.1 Public card phones in Germany	789
19.2 Telecommunication	792
19.3 Overview of mobile telecommunication systems	795
19.3.1 Multiple access methods	795
19.3.1.1 Frequency division multiple access (FDMA)	796
19.3.1.2 Time division multiple access (TDMA)	796
19.3.1.3 Code division multiple access (CDMA)	798
19.3.1.4 Space division multiple access (SDMA)	798
19.3.2 Cellular technology	799
19.3.3 Cell types	800
19.3.4 Bearer services	802
19.4 The GSM system	802
19.4.1 Specifications	804
19.4.2 System architecture and components	806
19.4.3 Important data elements	808
19.4.3.1 Coding of alphanumeric characters	808
19.4.3.2 SIM service table (SST)	810
19.4.3.3 Fixed dialing numbers (FDN)	810
19.4.3.4 ICC identification (ICCID)	810

---

19.4.3.5	International mobile equipment identity (IMEI)	810
19.4.3.6	International mobile subscriber identity (IMSI)	810
19.4.3.7	Ki (Key individual) and Kc (Key cipher)	810
19.4.3.8	Short messages service (SMS)	811
19.4.3.9	Abbreviated dialing numbers (ADN)	811
19.4.3.10	Location area information (LAI)	811
19.4.3.11	Mobile station ISDN number (MSISDN)	811
19.4.3.12	Temporary mobile subscriber identity (TMSI)	811
19.4.4	The subscriber identity module (SIM)	811
19.4.4.1	SIM commands	814
19.4.4.2	SIM files	816
19.4.4.3	Example of a typical command sequence	826
19.4.4.4	Authentication of the SIM	826
19.4.4.5	Mobile telephone switch-on and switch-off processes	830
19.4.4.6	SIM application toolkit (SAT)	833
19.4.4.7	Over-the-air (OTA) communication	838
19.4.4.8	Remote file management (RFM)	840
19.4.4.9	Remote applet management (RAM)	841
19.4.4.10	Dual IMSI	842
19.4.4.11	Implementing a home zone	844
19.4.4.12	Operating principle of SIM lock	845
19.4.4.13	Operating principle of prepaid systems	845
19.4.5	Future developments	848
19.5	The UMTS system	848
19.5.1	Future developments	852
19.6	The wireless identification module (WIM)	854
19.7	Microbrowsers	857
<b>20</b>	<b>Smart Cards in Health Care Systems</b>	861
20.1	Health insurance cards in Germany	861
20.2	Electronic health care cards in Germany	864
20.2.1	Card types	865
20.2.2	Applications in electronic health care cards	866
20.2.3	Electronic prescriptions	868
20.2.4	Summary and prospects	868
<b>21</b>	<b>Smart Cards in Transportation Systems</b>	869
21.1	Electronic tickets	869
21.1.1	System architecture	870
21.1.2	Octopus card	871
21.1.3	Trip registration	873
21.1.4	Typical transactions	874
21.1.4.1	Identification and authentication	875
21.1.4.2	Check-in transaction	876
21.1.4.3	Check-out transaction	876
21.1.5	Value-added services	876
21.1.6	Evolution of electronic tickets	877

---

21.2 Ski Passes	878
21.2.1 System architecture	878
21.2.2 Ski cards	880
21.2.3 Typical transactions	882
21.2.3.1 Identification and authentication	882
21.2.3.2 Reading data	883
21.2.3.3 Writing data	884
21.2.4 Future developments	885
21.3 Tachosmart	887
21.4 Electronic toll systems	887
<b>22 Smart Cards for Identification and Passports</b>	893
22.1 FINEID personal ID card	893
22.2 ICAO-compliant passports	894
<b>23 Smart Cards for IT Security</b>	899
23.1 Digital signatures	899
23.1.1 Applicable standards	900
23.1.2 The legal framework in Germany	900
23.1.3 System architecture	903
23.1.4 Card issuing	903
23.1.5 Signing and verifying documents	904
23.1.6 Trust center (TC)	905
23.1.7 Signature cards	906
23.1.8 Summary and prospects	909
23.2 Signature applications compliant with PKCS #15	909
23.3 Smart Card Web Server (SCWS)	912
<b>24 Application Design</b>	917
24.1 General information and characteristic data	917
24.1.1 Microcontrollers	917
24.1.1.1 Production	917
24.1.1.2 Service life	918
24.1.1.3 Data transmission	919
24.1.1.4 Algorithm execution times	920
24.1.2 Applications	920
24.1.2.1 Key management	920
24.1.2.2 Data	921
24.1.2.3 Data exchange	921
24.1.3 System aspects	922
24.1.3.1 Security	922
24.1.3.2 User interface	922
24.1.3.3 High-level design	923
24.1.4 Compliance with standards	923
24.2 Application generation tools	924
24.3 Analyzing an unknown smart card	926

<b>Contents</b>	<b>xxi</b>
<b>25 Appendix</b>	929
25.1 Glossary	929
25.2 Related reading	991
25.3 Bibliography	991
25.4 Directory of standards and specifications	999
25.5 Web addresses	1018
<b>Index</b>	<b>1021</b>

# Preface to the Fourth Edition

Preparing the fourth edition of a book with more than one thousand pages is not entirely the same as preparing the first edition of a technical book with three hundred pages. We learned this from painful experience in the course of the last two years, after we decided to write this new edition of the *Smart Card Handbook*.

Our decision was motivated by the dramatic evolution of smart card technology since the last edition of the book in 2002, which has resulted in so many fundamental changes that modifications were necessary on almost every page. With this major revision effort, we took the opportunity to migrate to a different working environment. Instead of using a certain well-known word processing program that was constantly on the verge of total collapse under the burden of this volume of material, we resolved to switch to a professional layout system. As well-known advocates of open-source software, we naturally had only one choice: LaTeX. Although we have never regretted this step, it did not exactly accelerate our project. One of the visible effects of this change for the reader is the large number of cross-references with page numbers. We also revised most of the figures and all of the tables. The result is a book that is distinctly more lucid and easier to read.

With this major revision, we have restructured the book to achieve a more logical arrangement of the various topics. This also allowed us to incorporate all the additions, changes, and special cases that have appeared in the previous editions in a structure that is once again self-contained and presents the entire subject in a clearly organized manner.

This also reflects the incipient paradigm shift in smart card technology. Until fairly recently, smart cards were largely niche products in the world of information technology, existing in a rather isolated technotope. However, in the last few years the technology of the PC and Internet worlds has made increasing inroads in the world of smart cards. As an example, we can mention cryptographically secured data transmission between smart cards and the outside world. The standard remains secure messaging, as specified by ISO/IEC 7816, but the integration of SSL and TSL protocols, long since proven in the Internet realm, is already on

the horizon. A similar situation can be seen with TCP/IP in the medium term. This will make smart cards uniquely addressable Internet devices and allow them to be integrated accordingly into the Internet infrastructure.

Another topic that made relatively large revisions necessary is the use of smart card microcontrollers with flash memory instead of mask-programmed ROM. If this evolutionary trend continues on its present course, and there is every reason to believe that it will, in only a few years there will be scarcely any ROM-based chips available for smart cards. This is accompanied by distinctly increased flexibility in operating systems and production logistics.

With regard to the overall organization of the book, we have maintained the proven approach of the previous editions. It begins with a relatively short chapter that provides a general introduction to the world of smart cards and sets the stage for the rest of the book. This is followed by several chapters devoted to the underlying aspects of the technology, which are necessary for proper understanding of this rather extensive subject. After this comes a group of chapters that deal with data transmission, commands, operating systems, smart card production, and quality assurance.

The book concludes with copious descriptions of diverse applications. We have limited the application descriptions to representative examples, since a nearly indescribable variety of new and interesting application areas have opened up for smart cards in the last few years.

At this point we would like to thank our families, friends and colleagues, whose help and encouragement made this book possible. Our particular thanks go to the following people: Bernhard Seen for his expert comments on card production; Jörn Treger for his thoroughgoing revision of the section on Java Card; Christoph Schiller for answering many questions about LaTeX; Johannes Reschreiter for his helpful information on smart cards in ski areas; Thomas Tarantino for helping with questions on card bodies; Michael Baldischweiler for his expert advice regarding USB, SWP and HCI; Peter Hildinger for reviewing the chapter on payment cards; Marcus Janke and Peter Laackmann for numerous tips and photos related to attacks on smart cards; Christopher Tarnovsky for his interesting photos of chip analysis equipment; Jürgen Hirschinger for his precise comments on the subject of testing; Harald Vater for answering many detailed questions on cryptography; Hermann Altschäfl for his practical advice on telecommunication applications; Peter van Elst and Dieter Weiβ for their always prompt and knowledgeable answers to many small questions about cards; Irene Weilhart for her outstanding suggestions and expert assistance on the typography and layout of technical books; and Margarete Metzger for her astounding patience every time we postponed the delivery date yet again, and for being an ideal partner in this book project.

Our special thanks also go to the many dedicated readers of the *Smart Card Handbook*, whose questions, comments and suggestions have often led us to new and interesting insights.

Munich, June 2008

**Wolfgang Rankl**

[Wolfgang@wrankl.de]  
[www.wrankl.de]

**Wolfgang Effing**

[Wolfgang.Effing@gi-de.com]

# Symbols and Notation

- In accordance with ISO nomenclature, the least significant bit is designated 1.
- The most significant byte of concatenated data is at the beginning and the least significant byte is at the end. In other words, concatenated data is big-endian.
- In accordance with common usage, a byte is a series of eight bits.
- Length specifications of data, objects, and all countable quantities are represented in decimal notation.
- When used in connection with data quantities or memory quantities, the prefixes ‘kilo’, ‘mega’, and ‘giga’ have the values of 1 024 ( $2^{10}$ ), 1 048 576 ( $2^{20}$ ), and 1 073 741 824 ( $2^{30}$ ).
- Binary values are used in a context-sensitive manner and are not explicitly identified as such.
- Smart card commands are set in uppercase characters (e.g. SELECT).
- As a rule, only good cases are shown in sequence diagrams.
- In diagrams, a solid arrow indicates a direction. By contrast, an open arrow is a pointer.
- Unless otherwise stated, all quantities are valid effective early 2008.
- In parameter coding tables for byte parameters consisting of two or more fields, the boundaries of the individual fields are marked by vertical rules.

## Representation of characters and numbers

0, 1	binary value (used in a context-sensitive manner and not explicitly identified as such)
8	decimal value
'00'	hexadecimal value
"ABC"	ASCII value
<i>bn</i>	bit number <i>n</i> (e.g. b8)
<i>Bn</i>	byte number <i>n</i> (e.g. B1)
<i>Dn</i>	digit number <i>n</i> (e.g. D3)

## References

See also . . . This is a reference to another location in the book.

[X Y] This is a reference to additional literature listed in the Appendix or an Internet site. In case of a literature reference, X is the surname of the first-named author and Y is the last two digits of the year of publication. A reference to a website on the Internet consists of a unique abbreviated identifier and does not include a year number.

## Cryptographic and data-related functions

$e = C(m)$	Compute the error detection code <i>e</i> of message <i>m</i> .
$t = T(d)$	Structure the data <i>d</i> using TLV coding. The result is the TLV-coded data <i>t</i> .
$p = P(d, v, l)$	Pad data <i>d</i> to an integer multiple of block length <i>l</i> using the value or method <i>v</i> . The result is the padded data <i>p</i> .
$c = E(p, k)$	Encrypt the plaintext <i>p</i> using a symmetric cryptographic algorithm and the secret key <i>k</i> . The result is the ciphertext <i>c</i> .
$p = D(c, k)$	Decrypt the ciphertext <i>c</i> using a symmetric cryptographic algorithm and the secret key <i>k</i> . The result is the plaintext <i>p</i> .
$a = M(m, k)$	Compute the message authentication code (MAC) of message <i>m</i> using the secret key <i>k</i> .
$s = S(m, sk)$	Sign the message <i>m</i> using the private key <i>k</i> .
$r = V(m, s, pk)$	Verify the signature <i>s</i> of message <i>m</i> using the public key <i>pk</i> . The result <i>r</i> is either true or false.
$h = H(m)$	Compute the hash value <i>h</i> of message <i>m</i> .
$C = (A, pk_A, S(A \parallel pk_A, sk_{CA}))$	Generate the certificate <i>C</i> of the public key <i>pk<sub>A</sub></i> of user <i>A</i> . This certificate is signed using the private key <i>sk<sub>CA</sub></i> of the certification authority <i>CA</i> .
$r = V(A \parallel pk_A, C, pk_{CA})$	Verify the certificate <i>C</i> of the public key <i>pk<sub>A</sub></i> of user <i>A</i> using the public key <i>pk<sub>CA</sub></i> of the certification authority <i>CA</i> . The result <i>r</i> is either true or false.

## Logical functions and program code

=	assignment operator (to be distinguished from the equality operator according to the context)
=, ≠, <, >, ≤, ≥	comparison operators
+, −, *, /	arithmetic operators
	concatenation operator (e.g. concatenation of two data elements or data objects)

## Program code

The syntax and semantics of the program code used in this book are based on current dialects of Basic. However, explanations in natural language may be used in a program listing for the sake of simplicity or clarity. Although this makes the code easier to understand for the reader, it prevents the code from being compiled automatically into machine code. This compromise is easily justified by the resulting significant improvement in readability.

:=	assignment operator
=, ≠, <, >, ≤, ≥	comparison operators
+, −, *, /	arithmetic operators
NOT	logical not
AND	logical and
OR	logical or
	concatenation operator (e.g. coupling two byte strings)
-	end-of-line marker in a multiline instruction
// ...	comment marker
<i>IO_Buffer</i>	variable (set in italic)
<b>Label:</b>	jump destination or call target (set in bold)
GOTO ...	jump
CALL ...	function call or subroutine call
RETURN	return from a function or subroutine
IF ... THEN ...	decision, type 1
IF ... THEN ... ELSE ...	decision, type 2
SEARCH (...)	search in a list (search string in parentheses)
STATUS (...)	query the result of a previously executed function call
STOP	terminate a process
LENGTH (...)	calculate a length
EXIST	test for presence (e.g. of an object or data element)
WITH ...	begin the declaration of a variable or an object as a reference
END WITH	end the declaration of a variable or an object as a reference

# Abbreviations

$\mu$ C	microcontroller
3DES	triple DES (data encryption standard) ( <i>see glossary</i> )
3GPP	Third Generation Partnership Project ( <i>see glossary</i> )
3GPP2	Third Generation Partnership Project 2 ( <i>see glossary</i> )
3rd FF	third form factor
A-PET	amorphous polyethylene terephthalate
A3, A5, A8	GSM algorithm 3, 5, 8 ( <i>see glossary</i> )
AAM	application abstract machine
ABA	American Bankers Association
ABS	acrylonitrile butadiene styrene
AC	access conditions ( <i>see glossary</i> )
ACD	access control descriptor
ACK	acknowledge
ACM	accumulated call meter
ADF	application dedicated file
ADK	additional decryption key
ADN	abbreviated dialing number
AES	Advanced Encryption Standard ( <i>see glossary</i> )
AFI	application family identifier
AFNOR	Association Française de Normalisation ( <i>see glossary</i> )
AGE	Autobahngebührenerfassung (motorway toll collection)
AGE	automatische Gebührenerfassung (automatic toll collection)
AID	application identifier ( <i>see glossary</i> )
AM	access mode
Amd.	amendment

AMPS	Advanced Mobile Phone Service ( <i>see glossary</i> )
ANSI	American National Standards Institute ( <i>see glossary</i> )
AoC	advice of charge
AODF	authentication object directory file
APACS	Association for Payment Clearing Services
APDU	application protocol data unit ( <i>see glossary</i> )
API	application programming interface ( <i>see glossary</i> )
AR	access rules
ARM	advanced RISC machine
ARR	access rule reference
ASC	application-specific command
ASCII	American Standard Code for Information Interchange
ASIC	application-specific integrated circuit
ASK	amplitude shift keying ( <i>see glossary</i> )
ASN.1	Abstract Syntax Notation One ( <i>see glossary</i> )
AT	attention
ATM	automated teller machine
ATQA	answer to request, type A
ATQB	answer to request, type B
ATR	answer to reset ( <i>see glossary</i> )
ATS	answer to select
AUX1, AUX2	auxiliary 1, auxiliary 2
BAC	Basic Access Control
BAFA	Bundesamt für Wirtschaft und Ausfuhrkontrolle
BASIC	Beginners All Purpose Symbolic Instruction Code
BCD	binary-coded digit
Bellcore	Bell Communications Research Laboratories
BER	Basic Encoding Rules ( <i>see glossary</i> )
BER-TLV	Basic Encoding Rules – tag, length, value
BEZ	Börsenevidenzzentrale(electronic purse clearing center for GeldKarte)
BGT	block guard time
BIBO	be-in / be-out
BIN	bank identification number
BIP	bearer independent protocol
bit	binary digit
BPF	basic processor functions
BPSK	binary phase-shift keying ( <i>see glossary</i> )
BS	base station
BSI	Bundesamt für Sicherheit in der Informationstechnik
BWT	block waiting time
C-APDU	command APDU ( <i>see glossary: command APDU</i> )
C-SET	Chip SET (secure electronic transaction)
CA	certification authority ( <i>see glossary: certification authority</i> )
CAD	chip accepting device ( <i>see glossary</i> )
CAFE	Conditional Access for Europe (EU project)
CAMEL	Customized Applications for Mobile Enhanced Logic

CAP	card application ( <i>see glossary: CAP file</i> )
CAPI	crypto API (application programming interface)
CASCADE	Chip Architecture for Smart Card and Portable Intelligent Devices
CASE	computer-aided software engineering
CAT	card application toolkit
CAT_TP	card application toolkit transport protocol
CAVE	Cellular Authentication, Voice Privacy And Encryption
CBC	cipher block chaining
CC	Common Criteria ( <i>see glossary</i> )
CCD	card coupling device
CCID	integrated circuit(s) cards interface device
CCITT	Comité Consultatif International Télégraphique et Téléphonique (now ITU) ( <i>see glossary</i> )
CCR	chip card reader
CCS	cryptographic checksum ( <i>see glossary</i> )
CD	committee draft
CDC	communications device class
CDF	certificate directory file
CDM	card dispensing machine
CDMA	code division multiple access ( <i>see glossary</i> )
CEN	Comité Européen de Normalisation ( <i>see glossary</i> )
CENELEC	Comité Européen de Normalisation Électrotechnique
CEPS	common electronic purse specifications ( <i>see glossary</i> )
CEPT	Conférence Européenne des Postes et Télécommunications ( <i>see glossary</i> )
CFB	cipher feedback
CGI	Common Gateway Interface
CHV	cardholder verification <i>or</i> cardholder verification information
CICC	contactless integrated chip card
CICO	check-in / check-out
CID	card identifier
CISC	complex instruction set computer
CLA	class
CLF	contactless front end
CLK	clock
CLn	cascade level n, type A
CMEA	Cellular Message Encryption Algorithm
CMM	capability maturity model ( <i>see glossary</i> )
CMOS	complementary metal oxide semiconductor
CMS	card management system
CoD	clear on deselect
CoR	clear on reset
COS	chip operating system ( <i>see glossary</i> )
COT	chip on tape ( <i>see glossary</i> )
CPA	Common Payment Application
CPU	central processing unit
CRC	cyclic redundancy check ( <i>see glossary</i> )

CRCF	clock rate conversion factor
CRT	Chinese remainder theorem
CRT	control reference template
Cryptoki	Cryptographic Token Interface
CSD	circuit-switched data
CT	card terminal
CT	cascade tag, type A
CT	chipcard terminal
CT	cordless telephone
CT-API	chipcard terminal API ( <i>see glossary</i> )
CTDE	cryptographic token data element
CTI	cryptographic token information
CTIO	cryptographic token information object
CVM	cardholder verification method
CWT	character waiting time
D	divisor
D-AMPS	Digital Advanced Mobile Phone Service ( <i>see glossary</i> )
DAD	destination address
DAM	DECT authentication module
DAM	draft amendment
DAP	data authentication pattern
DB	database
DBF	database file
DBMS	database management system
DC/SC	Digital Certificates on Smart Cards
DCODF	data container object directory file
DCS	digital cellular system
DEA	Data Encryption Algorithm ( <i>see glossary</i> )
DECT	Digital Enhanced Cordless Telecommunications ( <i>see glossary</i> )
DEMA	differential electromagnetic analysis
DER	Distinguished Encoding Rules ( <i>see glossary</i> )
DES	Data Encryption Standard ( <i>see glossary</i> )
DF	dedicated file <i>or</i> directory file ( <i>see glossary</i> )
DFA	differential fault analysis ( <i>see glossary</i> )
DG	data group
DIL	dual inline
DIN	Deutsche Industrienorm (German industrial standard)
DIS	draft international standard
DLL	dynamic link library
DMA	direct memory access
DO	data object
DoA	dead on arrival
DoD	Department of Defense (USA)
DOM	Document Object Model
DoS	denial of service
DOV	data over voice

---

DPA	differential power analysis ( <i>see glossary</i> )
dpi	dots per inch
DR	divisor receive (PCD to PICC)
DRAM	dynamic random access memory ( <i>see glossary</i> )
DRI	divisor receive integer (PCD to PICC)
DS	divisor send (PICC to PCD)
DSA	Digital Signature Algorithm
DSI	divisor send integer (PICC to PCD)
DSS	digital signature standard
DTD	Document Type Definition
DTMF	dual tone multiple frequency
DVD	digital versatile disc
E	end of communication, Type A
E <sup>2</sup> PROM	electrically erasable programmable read-only memory
EAC	extended access control
EAP	Extensible Authentication Protocol
EAP-SIM	extensible authentication protocol security identity module
EBCDIC	Extended Binary Coded Decimal Interchange Code
EC	elliptic curve <i>or</i> elliptic curve cryptoalgorithm
ec	Eurocheque
ECB	electronic code book
ECBS	European Committee for Banking Standards ( <i>see glossary</i> )
ECC	elliptic curve cryptosystems ( <i>see glossary</i> )
ECC	error correction code ( <i>see glossary</i> )
ECC	EU Citizen Card
ECDSA	Elliptic Curve Digital Signature Algorithm (DSA)
ECML	Electronic Commerce Modelling Language
ECTEL	European Telecom Equipment and Systems Industry
EDC	error detection code ( <i>see glossary</i> )
EDGE	Enhanced Data Rates for GSM and TDMA Evolution ( <i>see glossary</i> )
EDI	electronic data interchange
EDIFACT	Electronic Data Interchange for Administration, Commerce and Transport
EEM	Ethernet emulation model
EEPROM	electrically erasable programmable read-only memory ( <i>see glossary</i> )
EF	elementary file ( <i>see glossary</i> )
EFF	Electronic Frontier Foundation
EFI	EF internal
EFTPOS	electronic fund transfer at point of sale
EFW	EF working
eGK	elektronische Gesundheitskarte (German electronic health care card)
EGT	extra guard time, type B
EHIC	European Health Insurance Card
EMV	Europay, MasterCard, Visa ( <i>see glossary</i> )
EOF	end of frame, type B
EOP	end of packet

EP	endpoint
EPA	elektronische Patientenakte (electronic patient file)
EPROM	erasable programmable read-only memory ( <i>see glossary</i> )
ESD	electrostatic discharge
ETS	European Telecommunication Standard ( <i>see glossary</i> )
ETSI	European Telecommunications Standards Institute ( <i>see glossary</i> )
etu	elementary time unit ( <i>see glossary</i> )
ET	evaluation target ( <i>see glossary</i> )
f	following page
F2F	face to face
FAQ	frequently asked questions
FAR	false acceptance rate
FAT	file allocation table ( <i>see glossary</i> )
$f_c$	frequency of operating field (carrier frequency)
FCB	file control block
FCC	Federal Communications Commission
FCFS	first come, first served
FCI	file control information
FCOS	flip chip on substrate
FCP	file control parameters
FD/CDMA	frequency division / code division multiple access ( <i>see glossary</i> )
FDMA	frequency division multiple access ( <i>see glossary</i> )
FDN	fixed dialing number
FDT	frame delay time, type A
FEAL	Fast Data Encipherment Algorithm
FET	field effect transistor
ff	following pages
FID	file identifier ( <i>see glossary</i> )
FIFO	first in, first out
FINEID	Finnish Electronic Identification Card
FIPS	Federal Information Processing Standard ( <i>see glossary</i> )
FMD	file management data
FN	Fowler–Nordheim effect
FO	frame option
FPGA	field programmable gate array
FPLMTS	Future Public Land Mobile Telecommunication Service ( <i>see glossary</i> )
FRAM	ferroelectric random access memory ( <i>see glossary</i> )
FRR	false rejection rate
FS	file system
$f_s$	frequency of subcarrier modulation
FSC	frame size for proximity card
FSCI	frame size for proximity card integer
FSD	frame size for coupling device
FSDI	frame size for coupling device integer
FSK	frequency-shift keying
FTAM	file transfer, access, and management

---

FTL	flash translation layer ( <i>see glossary</i> )
FWI	frame waiting time integer
FWT	frame waiting time
FWTTEMP	temporary frame waiting time
GF	Galois field
GGSN	gateway GPRS support node
GMT	Greenwich Mean Time
GND	ground (electrical)
GNU	GNU's not Unix
GP	Global Platform ( <i>see glossary</i> )
GPL	GNU general public license
GPRS	General Packet Radio System ( <i>see glossary</i> )
GPS	Global Positioning System
GSM	Global System for Mobile Communications ( <i>see glossary</i> )
GSMA	GSM Association
GTS	GSM Technical Specification
GUI	graphical user interface
HAL	hardware abstraction layer ( <i>see glossary</i> )
HBA	Heilberufsausweis (health professional ID card)
HBCI	Home Banking Computer Interface ( <i>see glossary</i> )
HCI	host controller interface
HiCo	high coercivity
HLTA	halt command, type A
HTLB	halt command, type B
HMAC	keyed hash message authentication code (MAC)
HPC	health professional card
HSCSD	high-speed circuit-switched data
HSM	hardware security module
HSM	high-security module
HSP	High-speed Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HV	Vickers hardness
HW	hardware
I block	information block
I/O	input/output
I <sup>2</sup> C	inter-integrated circuit
IATA	International Air Transport Association
IBAN	international bank account number
IBE	identity-based encryption
ICAO	International Civil Aviation Organization
ICC	integrated circuit card ( <i>see glossary</i> )
ICCD	integrated circuit(s) card device
ICCSN	ICC serial number
ID	identifier

IDEA	International Data Encryption Algorithm
IEC	International Electrotechnical Commission ( <i>see glossary</i> )
IEEE	Institute of Electrical and Electronics Engineers
IEP	inter-sector electronic purse
IFD	interface device ( <i>see glossary</i> )
IFS	information field size
IFSC	information field size for the card
IFSD	information field size for the interface device
IIC	institution identification codes
IMEI	international mobile equipment identity
IMSI	international mobile subscriber identity
IMT-2000	International Mobile Telecommunication 2000 ( <i>see glossary</i> )
IN	intelligent network
INF	information field
INS	instruction
INTAMIC	International Association of Microcircuit Cards
IP	Internet protocol
IPES	Improved Proposed Encryption Standard
IPR	intellectual property rights
IrDA	Infrared Data Association
ISDN	Integrated Services Digital Network ( <i>see glossary</i> )
ISF	internal secret file
ISIM	IP security identity module
ISO	International Organization for Standardization ( <i>see glossary</i> )
IT	information technology
ITSEC	Information Technology Security Evaluation Criteria ( <i>see glossary</i> )
ITU	International Telecommunications Union ( <i>see glossary</i> )
IuKDG	Informations- und Kommunikations-Gesetz (Information and Communication Act)
IV	initialization vector
IVU	in-vehicle unit
J2ME	Java 2 Micro Edition
JC	Java Card
JCF	Java Card Forum ( <i>see glossary</i> )
JCP	Java Community Process
JCRE	Java Card runtime environment ( <i>see glossary</i> )
JCVM	Java Card virtual machine ( <i>see glossary</i> )
JDK	Java Development Kit ( <i>see glossary</i> )
JECK	Java electronic commerce framework
JFFS	journaling flash file system
JIT	just in time
JSR	Java specification request
JTC1	Joint Technical Committee One
JVM	Java virtual machine
K	key
Kc	ciphering key

KCV	check value key
KD	derived key
KFPC	key fault presentation counter
Ki	individual key
KID	key identifier
KM	master key
KS	session key
KVK	Krankenversichertenkarte (health insurance card)
LA	location area
LAN	local area network
L <sub>c</sub>	length command
LCSI	life cycle status indicator
LDS	logical data structure
L <sub>e</sub>	expected length
LEN	length
LFSR	linear feedback shift register
LIFO	last in, first out
LLC	logical link control
LND	last number dialed
LOC	lines of code
LoCo	low coercivity
LPDU	link protocol data unit
LRC	longitudinal redundancy check
LSAM	load secure application module
lsb	least significant bit
LSB	least significant byte
M	month
M2M	machine to machine ( <i>see glossary</i> )
MAC	medium access control
MAC	message authentication code ( <i>see glossary</i> )
MAO	multiapplication operating system
MBL	maximum buffer length
MBLI	maximum buffer length index
MCU	microcontroller unit
MD5	message digest algorithm 5
ME	mobile equipment
MEL	Multos Executable Language
MExE	mobile station execution environment ( <i>see glossary</i> )
MF	master file ( <i>see glossary</i> )
MFC	multifunction card
MIME	Multipurpose Internet Mail Extensions
MIPS	microprocessor without interlocked pipeline stages
MIPS	million instructions per second
MKT	Multifunktionales Kartenterminal (multifunctional card terminal) ( <i>see glossary</i> )
MLC	multilevel cell

MLI	multiple laser image
MM	moduliertes Merkmal
MMI	man–machine interface
MMS	multimedia messaging service
MMU	memory management unit
MOC	match on card
MOO	mode of operation
MOSAIC	microchip on surface and in card
MOSFET	metal oxide semiconductor field effect transistor
MoU	memorandum of understanding ( <i>see glossary</i> )
MRTD	machine-readable travel document
MRZ	machine-readable zone
MS	mobile station
msb	most significant bit
MSB	most significant byte
MSC	mass storage class
MSE	MANAGE SECURITY ENVIRONMENT
MTBF	mean time between failures
MUSCLE	Movement for the Use of Smart Cards in a Linux Environment
NAD	node address
NAK	negative acknowledgment
NBS	National Bureau of Standards (USA) ( <i>see glossary</i> )
NCSC	National Computer Security Center (USA) ( <i>see glossary</i> )
NDA	nondisclosure agreement
NFC	near field communication
NIST	National Institute of Standards and Technology (USA) ( <i>see glossary</i> )
NOK	not OK
NOP	no operation
NPU	numeric processing unit ( <i>see glossary</i> )
NRZ	non return to zero
NRZI	non return to zero inverted
NSA	National Security Agency (USA) ( <i>see glossary</i> )
NU	not used
NVB	number of valid bits
NVM	nonvolatile memory
OBU	onboard unit
OCF	Open Card Framework
OCR	optical character recognition
ODF	object directory file
OFB	output feedback
OID	object identifier
OMA	Open Mobile Alliance (formerly WAP)
OOK	on/off keying
OP	Open Platform ( <i>see glossary</i> )
OS	operating system
OSI	Open Systems Interconnect

OTA	Open Terminal Architecture
OTA	over the air ( <i>see glossary</i> )
OTASS	over the air SIM services
OTP	one-time password
OTP	one-time programmable
OTP	Open Trading Protocol
OVI	optically variable ink
P1, P2, P3	parameter 1, 2, 3
PA	power analysis
PACE	Password Authenticated Connection Establishment
PB	procedure byte
PC	personal computer
PC	polycarbonate
PC/SC	Personal Computer / Smart Card ( <i>see glossary</i> )
PCB	protocol control byte
PCD	proximity coupling device ( <i>see glossary</i> )
PCMCIA	Personal Computer Memory Card International Association
PCN	personal communication networks
PCS	personal communication system
PDA	personal digital assistant
PES	Proposed Encryption Standard
PET	polyethylene terephthalate
PETP	partially crystalline polyethylene terephthalate
PGP	Pretty Good Privacy
PICC	proximity ICC ( <i>see glossary</i> )
PIN	personal identification number
PIX	proprietary application identifier extension
PKCS	Public Key Cryptography Standards ( <i>see glossary</i> )
PKI	public key infrastructure ( <i>see glossary</i> )
PLL	phase locked loop
PLMN	public land mobile network ( <i>see glossary</i> )
PM	person month
POD	production on demand
POS	point of sale ( <i>see glossary</i> )
POZ	POS ohne Zahlungsgarantie (type of payment transaction)
PP	protection profile ( <i>see glossary</i> )
PPC	production planning and control
PPM	pulse position modulation
PPP	Point-to-point Protocol
PPS	protocol parameter selection
prEN	preliminary Europe Standard
prETS	preliminary European Telecommunication Standard
PrKDF	private key directory file
PRNG	pseudorandom number generator ( <i>see glossary</i> )
PROM	programmable read-only memory
PSAM	purchase secure application module

PSK	phase shift keying
PSO	PERFORM SECURITY OPERATION
PSTN	public switched telephone network ( <i>see glossary</i> )
PTS	protocol type selection
PTT	Post, Telegraph and Telephone
Pub	publication
PUK	personal unblocking key ( <i>see glossary</i> )
PuKDF	public key directory file
PUPI	pseudo-unique PICC identifier
PVC	polyvinyl chloride
PWM	pulse width modulation
QFN	quad flat pack, no leads
R-APDU	response APDU ( <i>see glossary</i> )
R-UIM	removable user identity module ( <i>see glossary</i> )
RACE	Research and Development in Advanced Communication Technologies in Europe
RAM	random access memory ( <i>see glossary</i> )
RATS	request to answer to select
Reg TP	Regulierungsbehörde für Telekommunikation und Post
REJ	reject
REQA	request command, type A
REQB	request command, type B
RES	resynchronisation
RF	radio frequency
RFC	Request for Comment
RFID	radio frequency identification
RFU	reserved for future use
RID	record identifier
RID	registered application provider identifier
RIPE	RACE Integrity Primitives Evaluation
RIPEMD	RACE Integrity Primitives Evaluation Message Digest
RISC	reduced instruction set computer
RMI	remote method invocation
RND	random number
RNDIS	remote network device interface specification
RNG	random number generator
ROM	read-only memory ( <i>see glossary</i> )
RS	Reed–Solomon
RSA	Rivest, Shamir and Adleman Algorithm
RST	reset
RTE	runtime environment
S	start of communication
S-HTTP	Secure Hypertext Transfer Protocol
S <sup>2</sup> C	SigIn–SigOut Connection
S@T	SIM Alliance Toolbox

---

S@TML	SIM Alliance Toolbox Markup Language
SA	security attributes
SA	service area
SAD	source address
SAGE	Security Algorithm Group of Experts
SAK	select acknowledge
SAM	secure application module ( <i>see glossary</i> )
SAS	Security Accreditation Scheme
SAT	SIM Application Toolkit ( <i>see glossary</i> )
SATSA	security and trust services API
SC	security conditions
SC	smart card
SCC	smart card controller
SCMS	smart card management system
SCOPE	smart card open platform environment ( <i>see glossary</i> )
SCP	smart card platform
SQL	Structured Card Query Language
SCSUG	Smart Card Security Users Group
SCWS	smart card web server
SDL	Specification and Description Language
SDMA	space division multiple access ( <i>see glossary</i> )
SE	security environment ( <i>see glossary</i> )
SECCOS	Secure Chip Card Operating System ( <i>see glossary</i> )
SEIS	Secured Electronic Information In Society
SEL	select code
SEMA	simple electromagnetic analysis
SEMPER	Secure Electronic Marketplace for Europe (EU project)
SEPP	Secure Electronic Payment Protocol
SET	secure electronic transaction ( <i>see glossary</i> )
SFGI	start-up frame guard time integer
SFGT	start-up frame guard time
SFI	short file identifier
SGSN	serving GPRS support node
SigG	Signaturgesetz ( <i>see glossary</i> )
SigV	Signaturverordnung ( <i>see glossary</i> )
SIM	subscriber identity module ( <i>see glossary</i> )
SIMEG	subscriber identity module expert group ( <i>see glossary</i> )
SKDF	secret key directory file
SLC	single-level cell
SM	secure messaging
SM	security mechanism
SMD	surface mounted device
SMG9	Special Mobile Group 9 ( <i>see glossary</i> )
SMIME	Secure Multipurpose Internet Mail Extensions
SMS	Short Message Service ( <i>see glossary</i> )
SMS-PP	Short Message Service Point to Point

SMSC	Short Message Service Center
SOF	start of frame
SOP	small outline package
SOP	start of packet
SPA	simple power analysis ( <i>see glossary</i> )
SPU	standard or proprietary use
SQL	Structured Query Language
SQUID	superconducting quantum interference device
SRAM	static random access memory ( <i>see glossary</i> )
SRES	signed response
SS	supplementary service
SSC	send sequence counter
SSCD	secure signature creation device
SSL	secure socket layer
SSO	single sign-on ( <i>see glossary</i> )
STARCOS	Smart Card Chip Operating System (G+D)
STC	sub-technical committee
STK	SIM Application Toolkit ( <i>see glossary</i> )
STT	secure transaction technology
SVC	Stored Value Card (Visa International)
SW	software
SW1, SW2	status word 1, 2
SWIFT	Society for Worldwide Interbank Financial Telecommunications
SWP	Single-wire Protocol
T	tag
TAB	tape automated bonding
TACS	Total Access Communication System
TAL	terminal application layer
TAN	transaction number ( <i>see glossary</i> )
TAR	toolkit application reference
tbd	to be defined
TC	technical committee
TC	thermochrome
TC	trust center ( <i>see glossary</i> )
TCOS	Telesec Card Operating System
TCP	Transport Control Protocol
TCSEC	Trusted Computer System Evaluation Criteria ( <i>see glossary</i> )
TD/CDMA	time division / code division multiple access ( <i>see glossary</i> )
TDES	triple DES ( <i>see glossary</i> )
TDMA	time division multiple access ( <i>see glossary</i> )
TETRA	Trans-European Trunked Radio ( <i>see glossary</i> )
TLS	transport layer security
TLV	tag length value ( <i>see glossary; TLV format</i> )
TMSI	temporary mobile subscriber identity
TOE	target of evaluation ( <i>see glossary</i> )
TPD	trusted personal device ( <i>see glossary</i> )

---

TPDU	transmission protocol data unit ( <i>see glossary</i> )
TRNG	true random number generator ( <i>see glossary: random number generator</i> )
TS	technical specification
TSCS	The Smart Card Simulator
TTCN	Tree And Tabular Combined Notation
TTL	terminal transport layer
TTL	transistor-transistor logic
TPP	trusted third party ( <i>see glossary</i> )
UART	universal asynchronous receiver transmitter ( <i>see glossary</i> )
UATK	UIM Application Toolkit
UCS	Universal Character Set ( <i>see glossary</i> )
UDP	User Datagram Protocol
UI	user interface
UICC	universal integrated chip card ( <i>see glossary</i> )
UID	unique identifier
UIM	user identity module ( <i>see glossary</i> )
UML	Unified Modeling Language ( <i>see glossary</i> )
UMTS	Universal Mobile Telecommunication System ( <i>see glossary</i> )
URL	uniform resource locator ( <i>see glossary</i> )
USAT	USIM Application Toolkit ( <i>see glossary</i> )
USB	Universal Serial Bus ( <i>see glossary</i> )
USIM	Universal Subscriber Identity Module ( <i>see glossary</i> )
USSD	unstructured supplementary services data
UTF	UCS transformation format
UTRAN	UMTS radio access network
VAS	value-added services ( <i>see glossary</i> )
Vcc	supply voltage
VCD	vicinity coupling device
VEE	Visa Easy Entry ( <i>see glossary</i> )
VICC	vicinity integrated chip card
VLSI	very large scale integration
VM	virtual machine ( <i>see glossary</i> )
VOP	Visa Open Platform ( <i>see glossary</i> )
Vpp	programming voltage
VSI	vertical system integration
W3C	World Wide Web Consortium
WAE	wireless application environment
WAN	wide area network
WAP	Wireless Application Protocol ( <i>see glossary</i> )
WCDMA	wideband code division multiple access ( <i>see glossary</i> )
WDP	Wireless Datagram Protocol
WfSC	Windows for Smart Cards
WG	working group
WIG	wireless Internet gateway
WIM	wireless identification module ( <i>see glossary</i> )

WML	Wireless Markup Language ( <i>see glossary</i> )
WORM	write once, read multiple
WSP	wafer-scale package
WSP	Wireless Session Protocol
WTAI	Wireless Telephony Application Interface
WTLS	Wireless Transport Layer Security
WTP	Wireless Transport Protocol
WTX	waiting time extension
WTXM	waiting time extension multiplier
WUPA	wake-up command, type A
WUPB	wake-up command, type B
WWW	World Wide Web ( <i>see glossary</i> )
XML	Extensible Markup Language ( <i>see glossary</i> )
XOR	logical exclusive OR operation
Y	year
ZKA	Zentraler Kreditausschuss ( <i>see glossary</i> )

# 1

## Introduction

This book is intended for students, engineers, and technically minded persons who want to learn more about smart card technology. It attempts to cover this broad topic as completely as possible, in order to provide the reader with a general understanding of the fundamentals and the current state of the technology.

We have put great emphasis on a practical approach. The wealth of illustrations, tables and references to real applications is intended to help the reader become familiar with the subject much faster than would be possible with a strictly technical approach. Consequently, this book is intended to be practically useful instead of academically complete. This is also the reason for making the descriptions as illustrative as possible. In places where we were faced with a choice between academic accuracy and ease of understanding, we have tried to strike a happy medium. Where this was not possible, we have given the preference to ease of understanding.

The book is structured such that it can be read in the usual way, from front to back. We have tried to avoid forward references as much as possible. The structure and content of the individual chapters are formulated to allow them to be read individually without any loss of understanding. A comprehensive index and a glossary allow this book to be used as a reference work. If you wish to know more about a specific topic, the references in the text and the annotated directory of standards will help you find the relevant documents.

Unfortunately, a large number of abbreviations have become established in smart card technology, as in so many other areas of technology and everyday life. This makes it particularly difficult for newcomers to become familiar with the subject. We have tried to minimize the use of these cryptic and frequently illogical abbreviations. Nevertheless, we have often had to choose a middle way between internationally accepted smart card terminology used by specialists and common terms more easily understood by laypersons. If we have not always succeeded, the extensive list of abbreviations should at least help overcome any barriers to understanding, which we hope will be short-lived. An extensive glossary at the end of the book explains the most important technical concepts and supplements the list of abbreviations.

An important feature of smart cards is that their properties are strongly based on international standards. This is also essential for interoperability, which is a fundamental requirement in most applications. Unfortunately, these standards are often difficult to understand, and in some problematic places they require outright interpretation. Sometimes only the members of the relevant standardization group can explain the intended meaning of certain sections. In such

cases, *The Smart Card Handbook* attempts to present the meaning generally accepted in the smart card industry. Nevertheless, the relevant standards remain the ultimate authority, and in such cases they should always be consulted.

## 1.1 THE HISTORY OF SMART CARDS

The proliferation of plastic cards began in the USA in early 1950s. The low price of the synthetic material PVC made it possible to produce robust, durable plastic cards that were much more suitable for everyday use than the paper and cardboard cards previously used, which could not adequately withstand mechanical stresses and climatic effects.

The first all-plastic payment card for general use was issued by the Diners Club in 1950. It was intended for an exclusive class of individuals, and thus also served as a status symbol, allowing the holder to pay with his or her ‘good name’ instead of cash. Initially, only the more select restaurants and hotels accepted these cards, so this type of card came to be known as a ‘travel and entertainment’ card.

The entry of Visa and MasterCard into the field led to a very rapid proliferation of ‘plastic money’ in the form of credit cards. This occurred first in the USA, with Europe and the rest of the world following a few years later.

Today, credit cards allow travelers to shop without cash everywhere in the world. A card-holder is never at a loss for means of payment, yet he or she avoids exposure to the risk of loss due to theft or other unpredictable hazards, particularly while traveling. Using a credit card also eliminates the tedious task of exchanging currency when traveling abroad. These unique advantages helped credit cards become rapidly established throughout the world. Billions of cards are produced and issued annually.

At first, the functions of these cards were quite simple. They served as data storage media that were secure against forgery and tampering. General data, such as the card issuer’s name, was printed on the surface, while personal data, such as the cardholder’s name and the card number, was embossed. Many cards also had a signature panel where the cardholder could sign his or her name for reference. In these first-generation cards, protection against forgery was provided by visual features such as security printing and the signature panel. Consequently, the system’s security depended largely on the experience and conscientiousness of the employees of the card-accepting organization. However, this did not represent an overwhelming problem, due to the card’s initial exclusivity. With the increasing proliferation of card use, these rather rudimentary functions and security technology were no longer adequate, particularly since threats from organized criminals were growing apace.

Increasing handling costs for merchants and banks made a machine-readable card necessary, while at the same time, losses suffered by card issuers as the result of customer insolvency and fraud grew from year to year. It became apparent that the security features for protection against fraud and manipulation, as well as the basic functions of the card, had to be expanded and improved.

The first improvement consisted of a magnetic stripe on the back of the card, which allowed digital data to be stored on the card in machine-readable form as a supplement to the visual information. This made it possible to minimize the use of paper receipts, which were previously essential, although the customer’s signature on a paper receipt was still required in traditional credit card applications as a form of personal identification. However, new approaches that rendered paper receipts entirely unnecessary could also be devised. This made it possible to

finally achieve the long-standing objective of replacing paper-based transactions by electronic data processing. This required a different method to be used for user identification, which previously employed the user's signature. The method that has come into widespread general use involves a secret personal identification number (PIN) that is compared with a reference number in a terminal or a background system. Most people are familiar with this method from using bank cards in automated teller machines. Embossed cards with a magnetic stripe and a PIN code are still the most commonly used type of payment card.

However, magnetic-stripe technology has a crucial weakness, which is that the data stored on the stripe can be read, deleted and rewritten at will by anyone with access to a suitable magnetic card reader/writer. It is thus unsuitable for storing confidential data. Additional techniques must be used to ensure confidentiality of the data and prevent manipulation of the data. For example, the reference value for the PIN can be stored in the terminal or host system in a secure environment, instead of on the magnetic stripe in unencrypted form. Most systems that employ magnetic-stripe cards thus use online connections to the system's host computer for reasons of security, even though this generates significant costs for the necessary data transmission. In order to minimize costs, it is necessary to find solutions that allow card transactions to be executed offline without endangering the security of the system.

The development of the smart card, combined with the expansion of electronic data processing systems, has created completely new possibilities for devising such solutions.

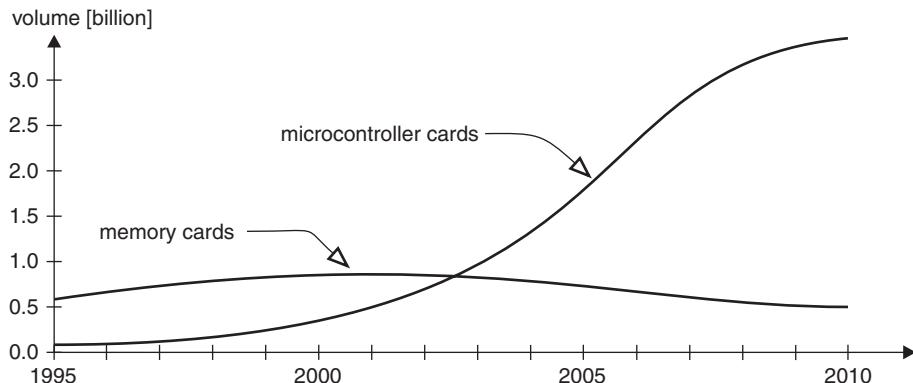
In the 1970s, rapid progress in microelectronics made it possible to integrate nonvolatile data memory and processing logic on a single silicon chip measuring a few square millimeters. The idea of incorporating such an integrated circuit into an identification card was contained in a patent application filed by the German inventors Jürgen Dethloff and Helmut Grötrupp as early as 1968. This was followed in 1970 by a similar patent application by Kunitaka Arimura in Japan. However, real progress in the development of smart cards began when Roland Moreno registered his smart card patents in France in 1974. It was only then that the semiconductor industry was able to supply the necessary integrated circuits at acceptable prices. Nevertheless, many technical problems still had to be solved before the first prototypes, some of which contained several integrated circuit chips, could be transformed into reliable products that could be manufactured in large numbers with adequate quality at a reasonable cost.

The basic inventions in smart card technology originated in Germany and France, so it is not surprising that these countries played the leading roles in the development and marketing of smart cards.

The great breakthrough was achieved in 1984, when the French PTT (postal and telecommunication services authority) successfully carried out a field trial with telephone cards. In this field trial, smart cards immediately proved to meet all expectations with regard to high reliability and protection against manipulation. Significantly, this breakthrough for smart cards did not come in an area where traditional cards were already used, but in a new application. Introducing a new technology in a new application has the great advantage that compatibility with existing systems does not have to be taken into account, so the capabilities of the new technology can be fully exploited.

A pilot project was conducted in Germany in 1984–85, using telephone cards based on several technologies. Magnetic-stripe cards, optical-storage (holographic) cards and smart cards were used in comparative tests.

Smart cards proved to be the winners in this pilot study. In addition to a high degree of reliability and security against manipulation, smart card technology promised the greatest



**Figure 1.1** Worldwide production figures for memory cards and processor cards. The numbers are estimated values, since the various sources differ considerably. Average values have been used here

degree of flexibility for future applications. Although the older but less expensive EPROM technology was used in the French telephone card chips, newer EEPROM chips were used from the start in German telephone cards. The latter type of chip does not need an external programming voltage. An unfortunate consequence is that the French and German telephone cards are mutually incompatible. Further developments followed the successful trials of telephone cards, first in France and then in Germany, with breathtaking speed. By 1986, several million 'smart' telephone cards were in circulation in France alone. The total rose to nearly 60 million in 1990, and to several hundred million worldwide in 1997.

Germany experienced similar progress, with a time lag of about three years. These systems were marketed throughout the world after the successful introduction of the smart card public telephone in France and Germany. Telephone cards incorporating chips are currently used in more than 50 countries. However, the use of telephone cards in their original home countries (France and Germany), as well as in highly industrialized countries in general, has declined dramatically in the last decade due to the widespread availability of inexpensive mobile telecommunication networks and the general use of mobile telephones.

The integrated circuits used in telephone cards are relatively small, simple and inexpensive memory chips with specific security logic that allows the card balance to be reduced while protecting it against manipulation. Microprocessor chips, which are significantly larger and more complex, were first used in large numbers in telecommunication applications, specifically for mobile telecommunication. The production trends of smart cards with memory chips (memory cards) and smart cards with microprocessor chips (microcontroller cards) in recent years are shown in Figure 1.1.

In 1988, the German Post Office acted as a pioneer in this area by introducing a modern processor card using EEPROM technology as an authorization card for the analog mobile telephone network (C-Netz). The reason for introducing such cards was an increasing incidence of fraud with the magnetic-stripe cards used up to that time. For technical reasons, the analog mobile telephone network was limited to a relatively small number of subscribers (around one million), so it was not a true mass market for processor cards. However, the positive experience gained from using smart cards in the analog mobile telephone system was decisive for the introduction of smart cards in the digital GSM network. This network was put into service in

1991 in various European countries and has presently expanded over the entire world, with more than three billion subscribers in nearly every country of the world.

Progress was significantly slower in the bank card area, in part due to the more stringent security requirements and higher complexity of bank cards compared with telephone cards. These differences are described in detail in the following chapters. Here we would just like to remark that the development of modern cryptography has been just as crucial for the proliferation of bank cards as developments in semiconductor technology.

With the widespread use of electronic data processing in the 1960s, the discipline of cryptography experienced a sort of quantum leap. Modern, high-performance hardware and software made it possible to implement complex, sophisticated mathematical algorithms in single-chip processors, which allowed previously unparalleled levels of security to be achieved. Moreover, this new technology was available to everyone, in contrast to the previous situation in which cryptography was a covert science in the private reserve of the military and secret services.

With these modern cryptographic algorithms, the strength of the security mechanisms in electronic data processing systems could be mathematically calculated. It was no longer necessary to rely on a highly subjective assessment of conventional techniques, whose security essentially rests on the secrecy of the methods used.

The smart card proved to be an ideal medium. It made a high level of security (based on cryptography) available to everyone, since it could safely store secret keys and execute cryptographic algorithms. In addition, smart cards are so small and easy to handle that they can be carried and used everywhere by everybody in everyday life. It was a natural idea to attempt to use these new security features for bank cards, in order to come to grips with the security risks arising from the increasing use of magnetic-stripe cards.

The French banks were the first to introduce this fascinating technology in 1984, after completion of a pilot project with 6000 cards in 1982–83. It took another 10 years before all French bank cards incorporated chips. In Germany, the first field trials took place in 1984–85, using a multifunctional payment card incorporating a chip. However, the Zentrale Kreditausschuss (ZKA), which is the coordinating committee of the leading German banks, did not manage to issue a specification for multifunctional Eurocheque cards incorporating chips until 1996. In 1997, all German savings associations and many banks issued the new smart cards. In the previous year, multifunctional smart cards with POS capability, an electronic purse, and optional value-added services were issued in all of Austria. This made Austria the first country in the world to have a nationwide electronic purse system.

An important milestone for the future worldwide use of smart cards for making payments was the adoption of the EMV specification, a product of the joint efforts of Europay, MasterCard and Visa. The first version of this specification was published in 1994. It provides a detailed description of the operation of credit cards incorporating processor chips, and it ensures the worldwide compatibility of the smart cards of the three largest credit card organizations. Hundreds of millions of EMV cards are presently in use worldwide.

With a delay of around ten years relative to normal contact smart cards, the technology of contactless smart cards has developed to the point of market maturity. With contactless cards, an electromagnetic field is used to supply power to the cards and exchange data with the terminal, without any electrical contact. The majority of currently issued EMV cards use this technology to enable fast, convenient payment for small purchases.

In the 1990s, it was anticipated that electronic purses, which store money in a card and can be used for offline payment, would prove to be another driver for the international proliferation

of smart cards for payment transactions. The first such system, called Danmønt, was put into service in Denmark in 1992. There are presently more than twenty national systems in use in Europe alone, many of which are based on the European EN 1546 standard. The use of such systems is also increasing outside of Europe. Payment via the Internet offers a new and promising application area for electronic purses. However, a satisfactory solution to the difficulties involved in using the public Internet medium to make payments securely but anonymously throughout the world, including small payments, has not yet been found. Smart cards could play a decisive role in such a solution.

The anticipated pioneering success of electronic purses has failed to materialize up to now. Most installed systems remain far below the original highly optimistic expectations, which among other things can be attributed to the fact that fees for online transactions have decreased dramatically, with the result that one of the key advantages of electronic purse systems – cost savings resulting from offline capability – has largely vanished. Today the electronic purse function is often included as a supplementary application in multifunction smart cards for payment transactions.

Another potentially important application for smart cards is as personal security devices for electronic signatures, which are slowly becoming established in several European countries after the legal basis for their use was created in 1999 when the European Parliament adopted an EU directive on digital signatures.

Another application has resulted the issuing of smart cards to nearly all the citizens of several countries. These smart cards serve as health insurance cards, which are issued to the insured persons and which contribute to cost savings in the billing of services to health insurance organizations. In most cases, the first cards to be issued were simple memory cards containing only the personal data of the insured person necessary for identification, but the patient cards now in common use contain complex security microcontrollers that also make it possible to store prescriptions and patient files, and to use electronic signatures to enable secure access to centrally stored data via the Internet.

The high functional flexibility of smart cards, which even allows programs for new applications to be added to a card already in use, has opened up completely new application areas, extending beyond the boundaries of traditional card uses.

As already mentioned, the technology of contactless smart cards has reached a level of maturity that enables economical mass production. For this reason, contactless smart cards are used as electronic tickets for local public transport in many cities throughout the world. In addition, this technology has established a firm position in electronic passports. Although electronic passports do not have the same size or shape as a credit card, which is standardized as an ID-1 card, under the cover they have the same circuitry as a contactless smart card, consisting of a security microcontroller connected to an antenna coil for contactless data exchange.

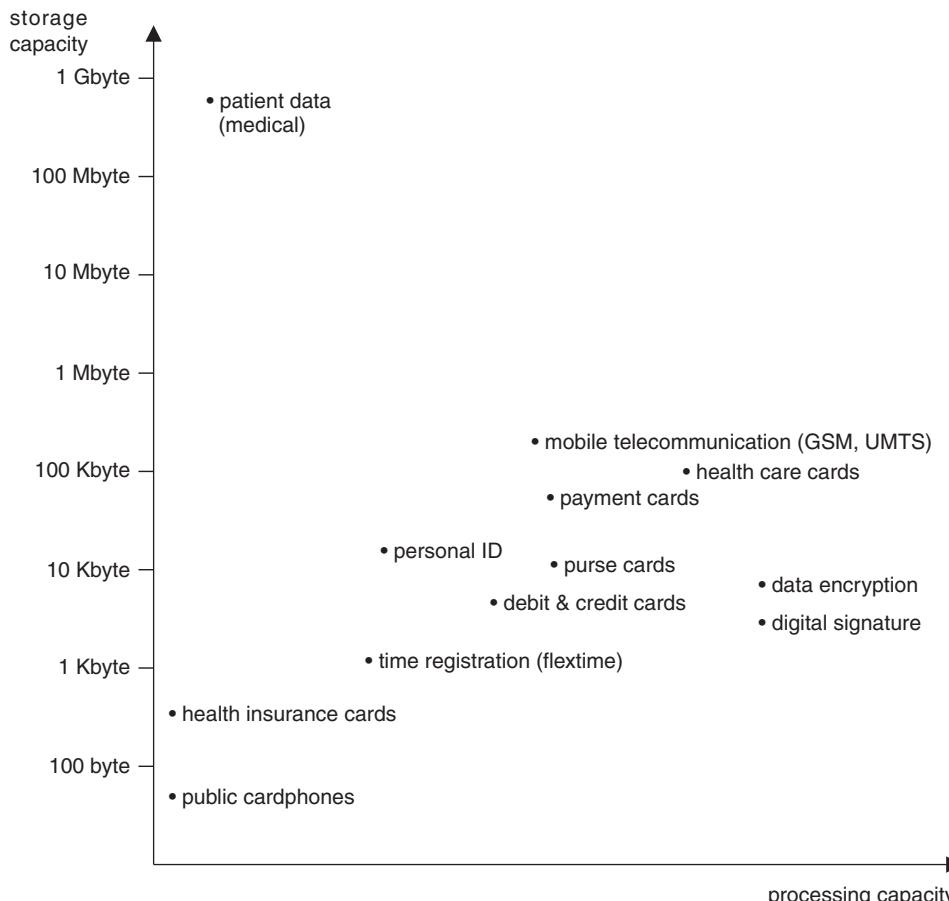
Intensive efforts are presently underway at the European level to achieve standardization of a contactless electronic card to be issued to all citizens, which will have an ID1 form factor (the same as a credit card) and is intended to be used as a personal identification card, among other things.

Although the history of smart cards and their applications goes back more than 25 years, a steady stream of promising new applications is still being developed. The increasing, almost omnipresent networking of our world creates major problems with regard to the security, confidentiality, and anonymity of personal data. Smart cards as personal security devices, with their ability to store and encode data securely, can make a major contribution to solving these problems.

## 1.2 CARD TYPES AND APPLICATIONS

As can be seen from the historical summary, the potential applications of smart cards are extremely diverse. With the steadily increasing storage and processing capacities of available integrated circuits, the range of potential applications is constantly expanding. Since it is impossible to describe all of these applications in detail within the confines of this book, a few typical examples must serve to illustrate the basic properties of smart cards. This introductory chapter is only meant to provide an initial overview of the functional versatility of these cards. Some typical application areas with their memory and processing capacities are shown in Figure 1.2, and several typical applications are described in detail in later chapters.

To make this overview easier to follow, it is helpful to divide smart cards into two categories: memory cards and processor cards.



**Figure 1.2** Typical smart card application areas, and the required memory capacity and arithmetic processing capacity

### 1.2.1 Memory cards

The first smart cards used in large quantities were memory cards for telephone applications. These cards are prepaid, with the value stored electronically in the chip being decreased by the amount of the calling charge each time the card is used. Naturally, it is necessary to prevent the user from subsequently increasing the stored value, which could easily be done with a magnetic-stripe card. With such a card, all the user would have to do is record the data stored at the time of purchase and rewrite it to the magnetic stripe after using the card. The card would then have its original value and could be reused. This type of manipulation, known as buffering, is prevented in smart phone cards by security logic in the chip that makes it impossible to erase a memory cell once it has been written. Decreasing the card balance by the number of charge units used is thus irreversible.

This type of smart card can naturally be used not only for telephone calls, but also whenever goods or services are to be sold against prior payment without the use of cash. Examples of possible uses include local public transport, vending machines of all types, cafeterias, swimming pools, car parks and so on. The advantage of this type of card lies in its simple technology (the surface area of the chip is typically only a few square millimeters), and hence its low cost. The disadvantage is that the card cannot be reused once it is empty, but must be discarded as waste – unless it ends up in a card collection.

Another typical application of memory cards is the German health insurance card, which has been issued since 1994 to all persons enrolled in the national health insurance plan. The information previously written on the patient's card is now stored in the chip and printed or laser-engraved on the card. Using a chip for data storage makes the cards machine-readable using simple equipment. However, the next generation of German health insurance cards will have a security microcontroller and significantly expanded functionality.

In summary, we can say that memory cards have limited functionality. Their integrated security logic makes it possible to protect stored data against manipulation. They are suitable for use as prepaid cards or identification cards in systems where low cost is a primary consideration.

### 1.2.2 Processor cards

As already mentioned, processor cards were first used as bank cards in France. Their ability to store secret keys securely and to execute modern cryptographic algorithms made it possible to implement highly secure offline payment systems.

As the processor embedded in the card is freely programmable, the functionality of processor cards is restricted only by the available memory and the computing power of the processor. The only limits to the designer's imagination when implementing smart card systems are thus technological, and they are extended enormously with each new generation of integrated circuits.

As the prices of processor cards steadily decline due to mass production and ongoing technological progress, more and more new applications are developed. The use of smart cards with mobile telephones has been especially important for their international proliferation.

After being successfully tested in the German national C-Netz (analog mobile telephone network) for use in mobile telephones, smart cards were specified as the access medium for the European digital mobile telephone system (GSM). In part, this was because smart cards allowed a high degree of security to be achieved for accessing the mobile telephone network.

At the same time, they provided new possibilities and thus major advantages in marketing mobile telephones, since they made it possible for network operators and service providers to sell telephones and services separately. Without smart cards, mobile telephones would certainly not have spread so quickly across Europe or developed into a worldwide standard.

Other potential applications for processor cards include identification cards, access control systems for restricted areas and computers, secure data storage, electronic signatures, electronic purses, and multifunctional cards incorporating several applications in a single card. Modern smart card operating systems also allow new applications to be loaded into a card after it has been issued to the user, without endangering the security of the various applications. This new flexibility opens up completely new application areas.

For example, personal security modules are indispensable if Internet commerce and payments are to be made trustworthy. Such security modules can securely store personal keys and execute high-performance cryptographic algorithms. This task can be handled elegantly by a processor card with a cryptographic coprocessor.

In summary, we can say that the essential advantages of processor cards are large storage capacity, secure storage of confidential data, and the ability to execute cryptographic algorithms. These advantages make a wide range of new applications possible, in addition to the traditional bank card application. The potential of smart cards is by no means yet exhausted, and furthermore, it is constantly being expanded by progress in semiconductor technology.

### 1.2.3 Contactless cards

The rapid progress of integrated circuit technology has led to a dramatic decrease in the power consumption of smart card microcontrollers. As a result, contactless cards, in which energy and data are transferred without any electrical contact between the card and the terminal, have become mature, inexpensive mass-produced products in the form of memory cards as well as processor cards. Although contactless processor cards are limited to operation at a distance of up to ten centimeters from the terminal due to their relatively high power consumption, contactless memory cards can be used up to a meter away from the terminal. This means that contactless memory cards do not necessarily have to be held in the user's hand in use, but can remain in the user's purse or wallet. Contactless cards are thus particularly suitable for applications in which people or items should be identified quickly. Sample applications include access control, local public transport, ski passes, airline tickets, and luggage identification.

However, there are also applications where operation over a long distance could cause problems and should be prevented. A typical example is an electronic purse. A declaration of intent on the part of the cardholder is normally required to complete a financial transaction. This confirms the amount of the payment and the cardholder's agreement to pay. With a contact card, this declaration takes the form of inserting the card in the terminal and confirming the indicated amount using the keypad. If contactless payments over relatively long distances were possible, a swindler could remove money from the electronic purse without the knowledge of the cardholder. Dual-interface cards offer a possible solution to this problem. These cards combine contact and contactless interfaces in a single card. Such a card can communicate with the terminal via either its contact interface or its contactless interface, according to what is desired.

There is considerable interest in using contactless cards for local public transport. If the functionality of smart cards used in payment systems, which are generally contact cards, is

expanded to enable them to act as electronic tickets with a contactless interface, transport system operators can utilize the infrastructure and cards of the credit card industry.

### 1.3 STANDARDIZATION

The prerequisite for the worldwide use of smart cards in everyday life, such as their present worldwide use in the form of SIM cards, health insurance cards, bank cards and passports, was the generation of national and international standards. Due to the special significance of such standards, in this book we repeatedly refer to currently applicable standards and those that are in preparation.

A smart card is normally part of a complex system. This means that the interfaces between the card and the rest of the system must be precisely specified and coordinated. Of course, this could be done for each system on a case-by-case basis, without regard to other systems. However, this would mean that a different type of smart card would be needed for each system. Users would thus have to carry a separate card for each application. In order to avoid this, an attempt has been made to generate application-independent standards that allow multifunctional cards to be developed. Since the smart card is usually the only component of the system that the user holds in his or her hand, it is enormously important for user awareness and acceptance of the entire system. However, from a technical and organizational perspective the smart card is usually only the tip of the iceberg, since complex systems (which are usually networked) are often hidden behind the card terminal, and it is these systems that make the customer benefits possible in the first place.

Let us take telephone cards as an example. In technical terms, they are fairly simple objects. By themselves, they are almost worthless, except perhaps as collector's items. Their true benefit, which is to allow public telephones to be used without coins, can be realized only after umpteen thousand card phones have been installed throughout a region and connected to a network. The large investments required for this can only be justified if the long-term viability of the system is ensured by appropriate standards and specifications. Standards are also an indispensable prerequisite for multifunctional smart cards that can be used for several different applications, such as phoning, an electronic purse, an electronic ticket, and so on.

#### What are standards?

This question is not as trivial as it may appear at first glance, especially because the terms 'standard' and 'specification' are often used interchangeably. A standard requires the consensus of all interested parties, while a specification has looser requirements with regard to consensus and open consultation. To make things clear, let us consider the ISO/IEC definition of a standard:

A document that is produced by consensus and adopted by a recognized organization, and which, for general and recurring applications, defines rules, guidelines or features for activities or their results, with the objective of achieving an optimum degree of regulation in a given context.

Here it should be noted that standards are based on the established results of science, technology and experience, and their objective is to promote the optimization of benefits for society. International standards should thus help make life easier and increase the reliability and usefulness of products and services.

In order to avoid confusion, ISO/IEC have also defined the term ‘consensus’ as general agreement, characterized by the absence of continuing objections to essential elements on the part of any significant portion of the interested parties, and achieved by a procedure that attempts to consider the views of all relevant parties and to address all counter-arguments. Here it should be noted that consensus does not necessarily mean unanimity.

Although unanimity is not required for consensus, the democratic process naturally takes a lot of time in many cases, especially because it is necessary to consider not only the views of the technical specialists, but also the views of all involved and affected parties, since the objective of a standard is the promotion of optimum benefits for the whole of society. Hence, the preparation of an ISO or CEN standard usually takes several years. A frequent consequence of the slowness of this process is that a limited group of interested parties, such as commercial firms, generates its own specification (‘industry standard’) in order to accelerate the launch of a new system. This is particularly true in the field of information technology, which is characterized by especially fast development and correspondingly short innovation cycles. Although industry standards and specifications have the advantage that they can be developed significantly faster than ‘true’ standards, they carry the risk of ignoring the interests of the parties that are not involved in their development. For this reason, ISO uses the ‘fast track’ procedure to allow important, publicly accessible specifications to be quickly published as ISO standards after the fact.

### What does ISO/IEC mean?

The relevant ISO/IEC standards are especially significant for smart cards because they are based on a broad international consensus and define the fundamental properties of smart cards. What lies behind the abbreviations ‘ISO’ and ‘IEC’? ‘ISO’ stands for the International Organization for Standardization, while ‘IEC’ stands for the International Electrotechnical Commission.

The International Organization for Standardization (ISO) is a worldwide association of around 100 national standards organizations, with one per country. ISO was founded in 1947 and is a nonnational organization. Its task is to promote the development of standards throughout the world, with the objective of simplifying the international exchange of goods and services and developing cooperation in the fields of science, technology and economy. The results of the activities of ISO are agreements that are published as ISO standards.

Incidentally, ‘ISO’ is not an abbreviation (the abbreviation of the official name would of course be ‘IOS’). Instead, the name ‘ISO’ is derived from the Greek word *isos*, which means ‘equal’ or ‘the same’. The prefix ‘iso-’ is commonly used in the three official languages of ISO (English, French and Russian), as well as in many other languages.

As already noted, the members of ISO are the national standards bodies of the individual countries, and only one such body per country is allowed to be a member. Germany is represented in ISO by the DIN organization. The member organizations have four basic tasks, as follows:

- Informing potentially interested parties in their own countries about relevant activities and opportunities for international standardization,
- Fashioning agreed national opinions and representing these opinions in international negotiations,
- Providing secretarial services for the ISO committees in which the country has a particular interest,

- Paying the country's financial contribution to support the activities of the central ISO organization.

The IEC (International Electrotechnical Commission) is an international standardization organization whose scope of responsibility is electrical technology and electronics. The first card standards, which did not include parts on the subject of electronics, were issued by ISO. After the introduction of smart cards, a difference of focus arose between the ISO and the IEC. In order to avoid duplication of effort, standards are developed in a joint technical committee (JTC 1, Joint Technical Committee for Information Technology) and published as ISO/IEC standards.

### **How is an ISO standard generated?**

The need for a standard is reported to a national standards organization by a special interest group, such as an association or a industrial sector committee. The national organization then proposes this to ISO as a new working topic. If the proposal is accepted by the responsible working group, which consists of technical experts from countries that are interested in the topic, the first thing that is done is to define the application area of the future standard.

After agreement has been reached on the technologies and applications to be defined in the standard, the details of the standard are discussed and negotiated between the various countries. This is the second phase in the development of a standard. The objective of this phase is to arrive at a consensus of all participating countries, if possible. The outcome of this phase is a 'draft international standard' (DIS).

The final phase consists of a formal vote on the draft standard. Acceptance of a standard requires the approval of two thirds of the ISO members that actively participated in drafting the standard, as well as three quarters of all members participating in the vote. Once the standard has been accepted, the agreed document is published as an ISO standard.

To prevent standards from becoming outdated as the result of ongoing development, ISO rules state that standards should be reviewed, and if necessary revised, after an interval of at most five years.

### **Cooperation with the IEC and CEN**

As already mentioned, ISO is not the only international standardization organization. In order to avoid duplication of effort, ISO cooperates closely with the IEC in certain areas. The areas of responsibility are defined as follows: the IEC is responsible for electrical technology and electronics, while ISO is responsible for all other areas. Combined working groups are formed to deal with topics of common interest, and these groups produce combined ISO/IEC standards. Most standards for smart cards belong to this category.

ISO and the Comité Européen de Normalisation (CEN) (European Standardization Committee) have also agreed on rules for the development of standards that are recognized as both European and international standards. This leads to time and cost savings.

The major industrial countries are represented in all relevant committees, and they generally also maintain 'mirror' committees in the form of national working groups and voting committees. The ISO website [ISO] provides an overview of the structure of ISO and its standardization projects. Smart card standards are developed by JTC 1/SC17 ('Cards and Personal Identification'). This working group also provides an overview of recently published standards and standards in progress.

At CEN, the topic of smart cards is handled by the TC 224 committee ('Personal identification, electronic signature and cards, and their related systems and operations').

The activities of CEN complement those of ISO. As much as possible, ISO standards are taken as the basis for CEN standards. If necessary, they are augmented with specifically European sections. In many cases, the number of options is reduced to simplify their implementation for purely European applications. The CEN working groups also produce standards for specific European applications that would not be able to achieve a consensus with ISO in a given form or at a given time.

An additional European standardization body, the European Telecommunications Standards Institute (ETSI), has made a significant contribution to the widespread international use of smart cards with its standard for SIM cards. ETSI generates standards for information and telecommunication technologies, which include mobile telecommunication and Internet technology.

ETSI is recognized by the European Commission as a European standardization organization. The members of ETSI are not the national standardization committees, but instead nearly 700 member organizations worldwide, which essentially represent the industrial sector, telecommunication companies, user groups, and research organizations. The smart card standards are prepared by the Technical Committee for Smart Card Platform (TC SCP). The TS 51.011 family of standards (formerly GS 51.011) specifies the interface between the smart card (called the subscriber identity module, SIM, in the GSM system) and the mobile telephone. This family of standards is based on the ISO/IEC standards. With the international expansion of GSM systems outside Europe, the ETSI standards have achieved global significance for the smart card industry.

After more than thirty years of standardization effort, the most important basic ISO standards for smart cards are now complete. They form the basis for further, application-oriented standards, which are currently being prepared by ISO and CEN.

These standards are based on prior ISO standards in the 7810, 7811, 7812 and 7813 families, which define the properties of identification cards in the ID-1 format. These standards include embossed cards and cards with magnetic stripes, which we all know in the form of credit cards.

Compatibility with these existing standards was a criterion from the very beginning in the development of standards for smart cards (which are called ‘integrated circuit(s) cards’, ICC, in the ISO standards), in order to provide a smooth transition from embossed cards and magnetic-stripe cards to smart cards. Such a transition is possible because all functional components, such as embossing, magnetic stripes, contacts and interface components for contactless interfaces, can be integrated into a single card. Of course, a consequence of this is that the integrated circuits, which are sensitive electronic components, are exposed to high stresses during the embossing process and recurrent impact stresses when the embossed characters are printed onto paper. This makes heavy demands on the packaging of the integrated circuits and the manner in which they are embedded in the card.

A summary of the currently available standards, with brief descriptions of their contents, can be found in the Appendix.<sup>1</sup>

In the last few years, an increasing number of specifications have been prepared and published by industrial organizations and other nonpublic groups, with no attempt being made to incorporate them in the standardization activities of ISO. The reason most often given for this approach is that the way ISO operates is too slow to keep pace with the short innovation cycles of the information technology and telecommunication industries. Some examples of consortiums that generate specifications relevant to smart cards are Java Card Forum, Open

<sup>1</sup> See also Section 25.4, ‘Directory of Standards and Specifications’, on page 999

Mobile Alliance (OMA), Global Platform, and NFC Forum. In many cases, only a few interest groups are involved in drafting these industry standards, so there is a risk that the interests of smaller companies, and especially the interests of the general public, may be ignored in the process. Fortunately, the most important consortiums work closely together with standardization organizations and try to include the most important specifications in the standardization process later on. It is a major challenge to the future of ISO and IEC to devise processes that make it possible to safeguard general interests without hampering the pace of innovation.

# 2

## Card Types

Smart cards are the youngest member of the family of identification cards using the ID-1 format defined in ISO/IEC standard 7810, ‘Identification Cards – Physical Characteristics’. This standard specifies the physical properties of identification cards, including their material properties such as flexibility and temperature resistance, as well as the dimensions of three different card formats: ID-1, ID-2, and ID-3. The ISO 7816-1 family of smart card standards is based the ID-1 card format, which is commonly used for the payment cards used by millions of people.

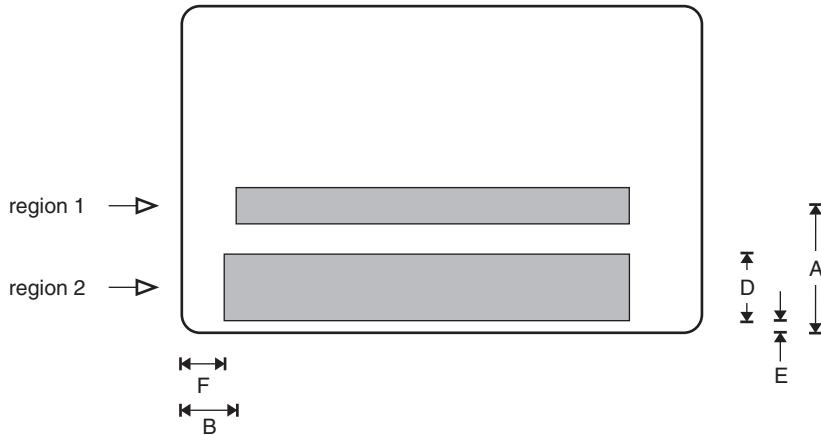
This chapter provides an overview of a variety of cards in ID-1 format because many applications have a special interest in combining several functions, especially when the cards used in an existing system (such as magnetic-stripe cards) are intended to be replaced by smart cards. In such cases, it is usually not possible to replace the existing infrastructure (such as magnetic-stripe card terminals) by a new technology overnight.

The solution to this problem usually consists of issuing cards with magnetic stripes as well as chips for use during a transition period. Such cards can be used with both types of terminals (old and new). Naturally, new functions that are only possible with a chip cannot be used with a terminal that only supports magnetic-stripe cards.

Similar considerations apply to the transition from contact smart cards to contactless smart cards. In many cases, the infrastructure must be able to support both types of cards during an extended transition period. Form factors other than ID-1 have now become established in some applications where compatibility with magnetic-stripe cards is not necessary. SIM cards in ID-000 format are one example. Naturally, the information regarding electrical properties and functions also applies to these other form factors.

### 2.1 EMBORESSED CARDS

Embossing is the oldest technique for adding machine-readable features to identification cards. The embossed characters on the card can be transferred to paper using simple, inexpensive devices, and they can easily be read visually (by humans). The nature and location of the embossing are specified in the ISO/IEC 7811 standard (‘Identification Cards - Recording Techniques’). This standard, which is divided into five parts, deals with magnetic stripes as well as embossing.



**Figure 2.1** Embossing regions according to ISO 7811. Region 1 is intended for the identification number (19 characters), while region 2 is intended for the name and address (4 lines of 27 characters each). The following dimensions are specified in the standard: A:  $21.42 \pm 0.12$  mm; B:  $10.18 \pm 0.25$  mm; D: 14.53 mm; E: 2.41–3.30 mm; F:  $7.65 \text{ mm} \pm 0.25$  mm

Two different embossing regions are defined, as shown in Figure 2.1. Region 1 is reserved for the card identification number, which identifies the card issuer as well as cardholder. Region 2 is reserved for additional cardholder data, such as the cardholder's name and address.

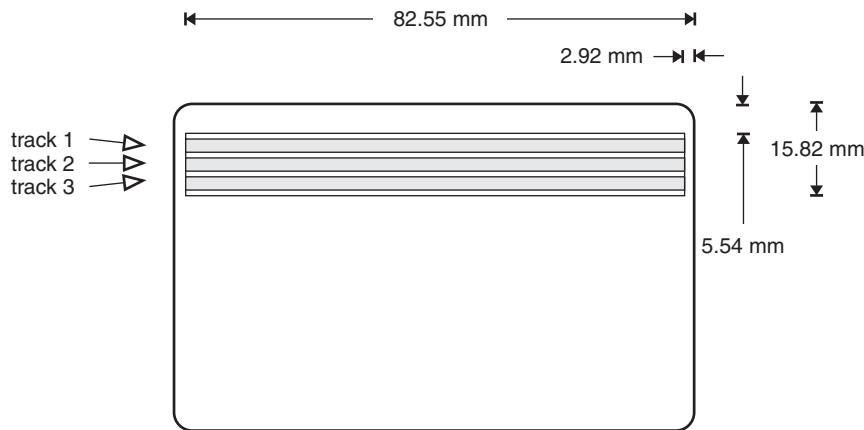
At first glance, transferring information by printing from embossed characters may appear quite primitive. However, the simplicity of this technique has made worldwide use of credit cards possible, even in developing countries. Utilization of this technology requires neither electrical power nor a connection to a telephone network.

## 2.2 MAGNETIC-STRIPE CARDS

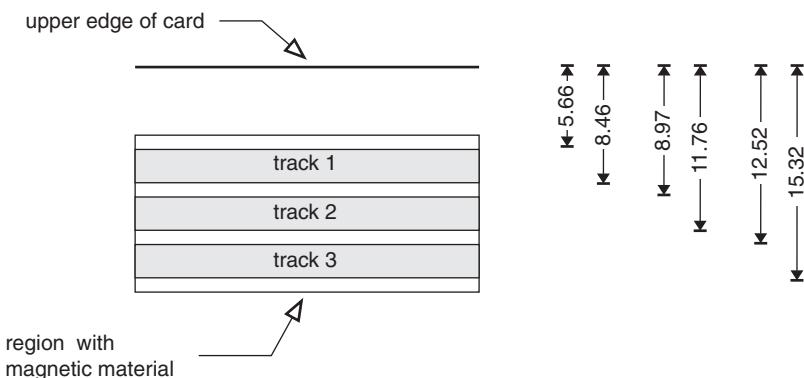
The essential disadvantage of embossed cards is that their use creates a flood of paper receipts, which are expensive to handle and process. One remedy to this problem is to digitally encode the card data on a magnetic stripe located on the back of the card.

The magnetic stripe is read by pulling it across a read head, either manually or automatically. After this, the read data can be used in the system without any human intervention. One of the properties of magnetic stripes is the strength of the magnetic field (measured in oersteds) necessary to modify the data on the magnetic stripe. With low-coercivity (*loco*) magnetic stripes, the required magnetic field strength is 300 to 650 oersted, while with high-coercivity (*hico*) magnetic stripes the required magnetic field strength is 1250 to 4000 oersted. High-coercivity magnetic stripes have the advantage that they cannot be erased accidentally, such as by a nearby magnet, but they require special write heads.

Parts 2, 6, 7 and 8 of ISO/IEC standard 7811 specify the properties of the magnetic stripe, the coding method, and the locations of the magnetic tracks. The magnetic stripe may have up to three tracks, as illustrated in Figure 2.2 on the next page. Tracks 1 and 2 are specified to be read-only tracks, while track 3 may also be written to. The specified contents of the data tracks are listed in Table 2.1 on page 18, while typical contents and their positions are listed in Table 2.2 on page 18.



**Figure 2.2** Magnetic stripe location on an ID-1 card. The data region of the magnetic stripe is intentionally not extended to the edges of the card, since the use of hand-operated card readers causes rapid wear at the ends of the stripe



**Figure 2.3** Locations of the data tracks on an ID-1 card (all dimensions in mm)

Although the storage capacity of the magnetic stripe is fairly small (around 1000 bits), it is more than sufficient for storing the information contained in the embossing. Additional data can be read and written on track 3, such as the most recent transaction data in the case of a credit card. The track locations are shown in Figure 2.3.

The main drawback of magnetic-stripe technology is that the stored data can be altered very easily. Manipulating embossed characters requires at least a certain amount of manual dexterity, and such manipulations can readily be detected by a trained eye. By contrast, the data recorded on the magnetic stripe can easily be altered using a standard read/write device, and it is difficult to prove such changes afterward. In addition, magnetic-stripe cards are often used in automated equipment such as cash dispensers, in which visual inspection is not possible. A potential criminal, having obtained valid card data, can easily use duplicated cards in such unattended machines without having to forge the visual security features of the cards.

**Table 2.1** The data tracks of a magnetic-stripe card as specified in ISO/IEC 7811

Property	Track 1	Track 2	Track 3
Data volume	79 characters max.	40 characters max.	107 characters max.
Data coding	6-bit alphanumeric	4-bit BCD	4-bit BCD
Data density	210 bit/inch (8.3 bit/mm)	75 bit/inch (3 bit/mm)	210 bit/inch (8.3 bit/mm)
Writing	not allowed	not allowed	allowed

**Table 2.2** Example magnetic stripe data content and coding of a typical credit card

Track	Position	Data
1	2–17	credit card number
1	19–44	surname of the cardholder
1	46–47	expiry year
1	48–49	expiry month
2	1–16	credit card number
2	18–19	expiry year
2	20–21	expiry month

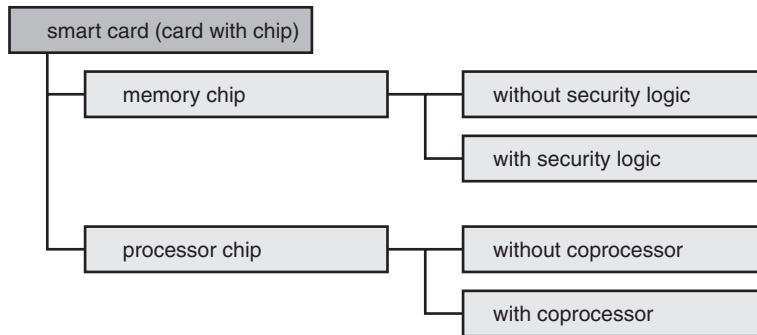
Manufacturers of magnetic-stripe cards have developed various techniques to protect the data recorded on the magnetic stripe against forgery and duplication. For example, German Eurocheque cards contain an invisible, unalterable code in the body of the card, which effectively makes it impossible to alter or duplicate the data on the magnetic stripe.<sup>1</sup> However, such techniques require a special sensor in the card terminal, which considerably increases the cost of the terminal. For this reason, none of these techniques has so far become established internationally.

## 2.3 SMART CARDS

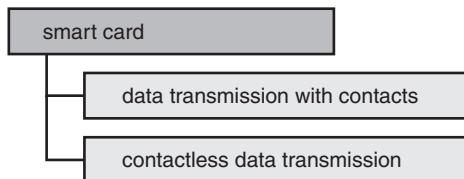
Smart cards are the latest innovation in the family of identification cards in ID-1 format. Their characteristic feature is an integrated circuit embedded in the card body, which has components for transmitting, storing and processing data. The data can be transmitted by contacts on the surface of the card or by an electromagnetic field without using contacts.

Smart cards offer several advantages compared with magnetic-stripe cards. For example, the maximum storage capacity of a smart card is many times greater than that of a magnetic-stripe card. Chips with memory capacities in the megabyte range are now available, and the maximum capacity increases with each new chip generation. Only optical memory cards, which are described in the next section, and smart cards with supplementary NAND flash memory have greater capacity.

<sup>1</sup> See also Section 3.5.13, ‘Moduliertes Merkmal’, on page 48



**Figure 2.4** Classification of smart cards by chip type



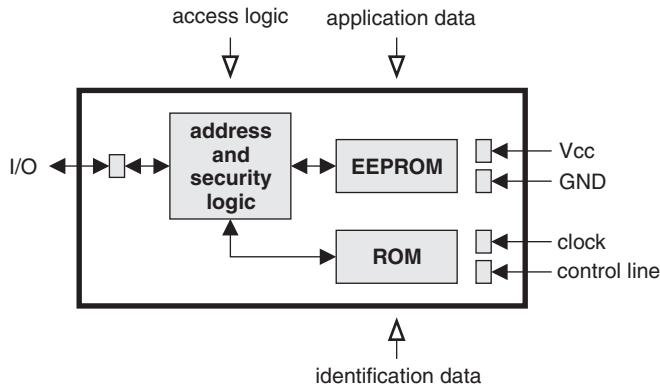
**Figure 2.5** Classification of smart cards by data transmission method

However, one of the most important advantages of smart cards is that their stored data can be protected against unauthorized access and manipulation. The data can only be accessed via a serial interface that is controlled by the operating system or security logic, which means that confidential data can be stored in the card in a manner that prevents it from ever being read from outside the card. Such confidential data can be processed only internally by the chip's processing unit. In principle, hardware and software mechanisms can both be used to restrict use of the memory functions (reading, writing, and erasing data) and subject them to specific conditions. This makes it possible to construct a variety of security mechanisms, which can also be tailored to the specific requirements of a particular application.

In combination with the ability to compute cryptographic algorithms, this allows smart cards to be used to implement convenient security modules that can be carried by users at all times, for example in a purse or wallet. Some additional advantages of smart cards are their high reliability and long life compared with magnetic-stripe cards, whose useful life is generally limited to two to three years.

The fundamental properties and functions of smart cards are specified in the ISO/IEC 7816 family of standards, which are described in detail in subsequent chapters. Smart cards can be divided into two groups that differ in both functionality and price: memory cards and processor cards (see Figure 2.4).

Smart cards can also be classified on the basis of their data transmission method (see Figure 2.5). Data can be transmitted using mechanical contacts or wirelessly using electromagnetic coupling. Memory cards and processor cards are both available in contact and contactless forms.



**Figure 2.6** Typical architecture of a contact memory card with security logic. The figure shows only basic energy and data flows and is not a detailed schematic diagram

### 2.3.1 Memory cards

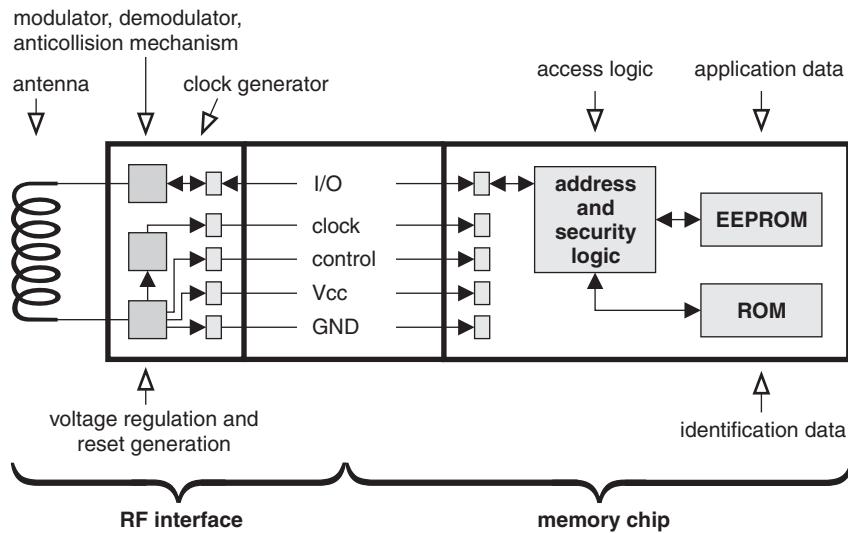
Figure 2.6 shows the architecture of a memory card in block diagram form. The data needed by the application is stored in nonvolatile memory, which is usually EEPROM. Access to the memory is controlled by the security logic, which in the simplest case consists only of write protection or erase protection for the memory or certain memory regions. However, there are also memory chips with more complex security logic that can also perform simple encryption. Data is transferred to and from the card via a serial interface. Part 3 of the ISO/IEC 7816 standard defines a special synchronous transmission protocol that enables an especially simple and inexpensive implementation in the chip. However, some smart cards employ the I<sup>2</sup>C bus, which is widely used for serial-access memories.

The functionality of memory cards is usually optimized for a particular application. Although this severely restricts the flexibility of these cards, it makes them quite inexpensive. Typical applications for memory cards are prepaid telephone cards and simple health insurance cards.

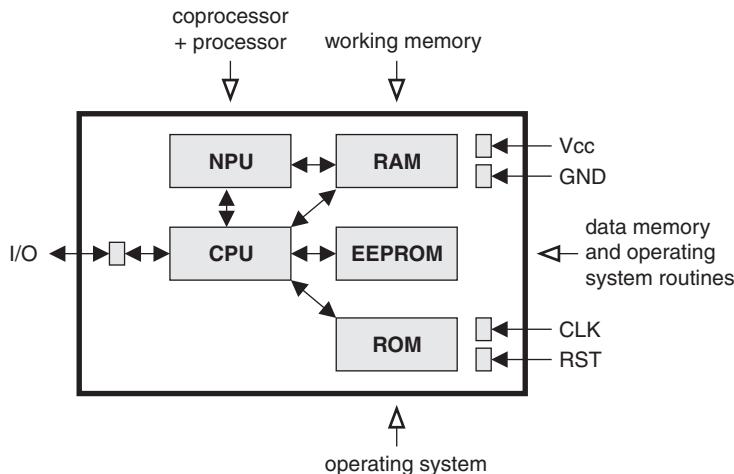
### 2.3.2 Contactless memory cards

Contactless memory cards have come into widespread use in recent years. Standard cards compliant with ISO/IEC 14443 have an operating range of up to 10 cm. The usable memory capacity ranges from several hundred bytes to a few kilobytes. The memory can be partitioned into several sectors that are independently protected against unauthorized reading, writing, and erasing. This enables a single card to support several different applications. Each card has a unique serial number stored in ROM, as well as authentication logic using a challenge-response method. A typical architecture for contactless memory cards is shown in Figure 2.7 on the next page.

Typical applications for contactless memory cards are contactless ticketing in local transport systems and identification cards for companies, official bodies, or universities. Electronic purse functionality, such as for a company cafeteria or university cafeteria, can also be implemented on such cards. Contactless technology compliant with ISO/IEC 14443 is also used extensively now in radio-frequency identification (RFID) tags.



**Figure 2.7** Typical architecture of a memory card with security logic and a contactless interface. The figure shows only basic energy and data flows and is not a detailed schematic diagram

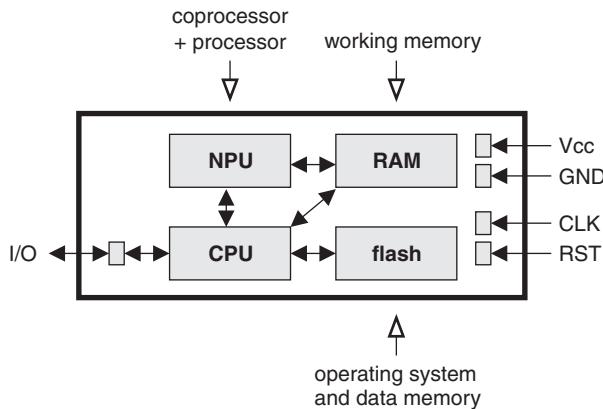


**Figure 2.8** Typical architecture of a contact processor card with a coprocessor and mask-programmed ROM. The figure shows only basic energy and data flows and is not a detailed schematic diagram

### 2.3.3 Processor cards

The principal component of a processor card, which is formally known as a microprocessor card, is the processor (CPU), which is usually surrounded by four other functional blocks: mask ROM, EEPROM, RAM, and an I/O port. Figure 2.8 shows the typical architecture of this type of device.

The mask ROM contains the chip's operating system, which is permanently stored in memory when the chip is manufactured. The content of the ROM is thus identical for all chips



**Figure 2.9** Typical architecture of a contact processor card with a coprocessor and flash memory. The figure shows only basic energy and data flows and is not a detailed schematic diagram

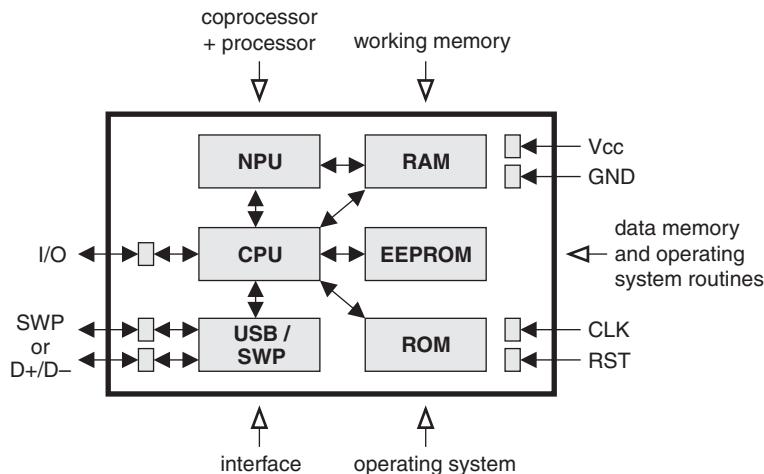
of a production batch, and it cannot be changed during the chip's lifetime. The EEPROM is the chip's nonvolatile memory. Data and program code can be written to and read from the EEPROM under control of the operating system. The RAM is the processor's working memory. This memory is volatile, so all data stored in it is lost when the chip is de-energized. In its simplest form, the serial I/O interface consists only of a single register used to transfer data bit by bit.

Processor cards are very versatile in use. In the simplest case, they contain a program optimized for a single application, which means that they can only be used for this particular application.

However, modern smart card operating systems allow several different applications to be integrated in a single card. In this case, the ROM contains only the basic components of the operating system, with the application-specific components being loaded into the EEPROM only after the card has been manufactured. Recent developments also allow application programs to be loaded into a card after it has already been personalized and issued to the cardholder. Special hardware and software measures are used to ensure that the different security conditions of the individual applications are not violated. Semiconductor manufacturers can supply microprocessor chips with high processing power, large memory capacity and sophisticated security logic that are specially optimized for this purpose.

In a trend that parallels that of technological progress in other areas, such as digital cameras and MP3 players, mask-programmed ROM and EEPROM are increasingly being supplanted by flash (short for flash EEPROM) in newer types of processor cards. Flash memory has the advantage of distinctly greater flexibility in production and personalization compared with ROM, which cannot be modified after manufacturing. Flash memory enables capabilities such as adapting the operating system to the wishes of the customer and loading it into the cards after manufacturing. Figure 2.9 shows the architecture of a contact processor card with flash memory.

The dramatic increase in the memory capacity of processor cards in recent years, which shows no signs of ending, has increasingly highlighted the limitations imposed on the data transmission rate by the serial interface. Consequently, new interfaces and transmission protocols have been developed to accommodate the new requirements. For instance, ETSI standardized the USB interface for SIM cards in 2007. Another new interface is the Single



**Figure 2.10** Typical architecture of a contact processor card with a coprocessor, T = 0/T = 1 interface, USB interface, and SWP interface. The figure shows only basic energy and data flows and is not a detailed schematic diagram

Wire Protocol (SWP), which links the SIM card to an NFC controller in a mobile telephone. In the future, more and more processor cards will support several I/O interfaces in parallel, as illustrated in Figure 2.10.

### 2.3.4 Contactless processor cards

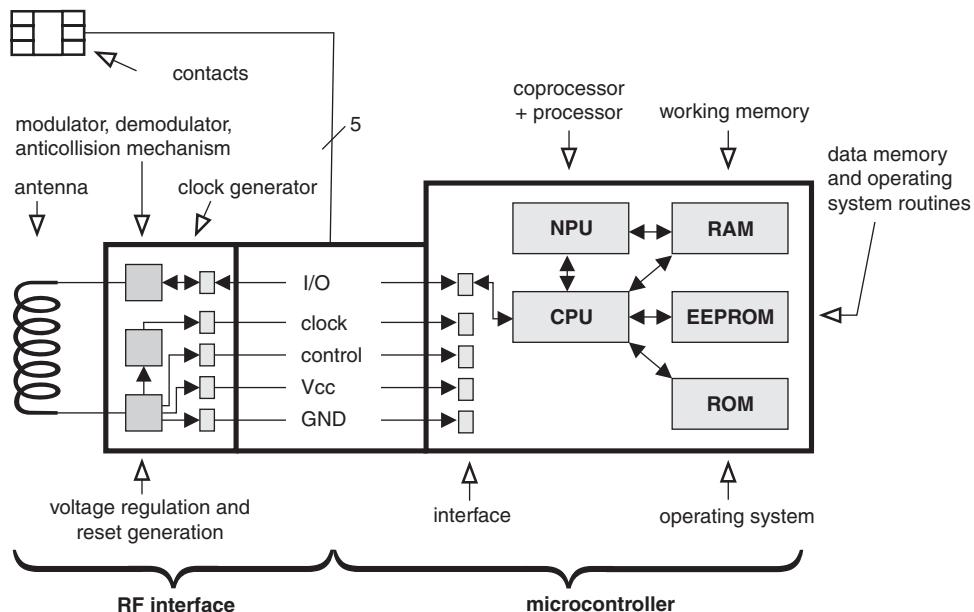
Contactless smart card technology enables a variety of interesting new applications for card issuers and card users. For instance, contactless cards do not have to be inserted into a card reader, but instead only have to be held close to a reader, since contactless processor cards have a working range of up to 10 cm. This is a great advantage in applications such as access control systems that control whether a door opens or a turnstile turns. A major application area for this is local public transport, which requires reliable recognition of a large number of people in the shortest possible time.

A further interesting variation on using contactless cards involves utilizing the surface of the reader. With this option, the card is not inserted into a slot, but instead simply held against a marked area on the surface of the card reader. In addition to ease of use, this solution is attractive because it significantly reduces the risk of vandalism, such as pressing chewing gum or superglue into the card slot.

For card marketing, contactless technology offers the advantage that no technical components are visible on the surface of the card, so the graphic design of the card is not constrained by magnetic stripes or contacts. However, this advantage comes at the price of more complex terminals with correspondingly higher prices.

Contactless card technology has now matured to the point that high-quality products are available at prices that are not significantly higher than those of comparable contact cards. Up to now, contactless cards have been used primarily in local public transport systems, in which they serve as electronic tickets that enable modern electronic fare management.

Although most currently operating systems still use single-function cards, which typically contain inexpensive memory chips with hard-wired security logic, there is an increasing



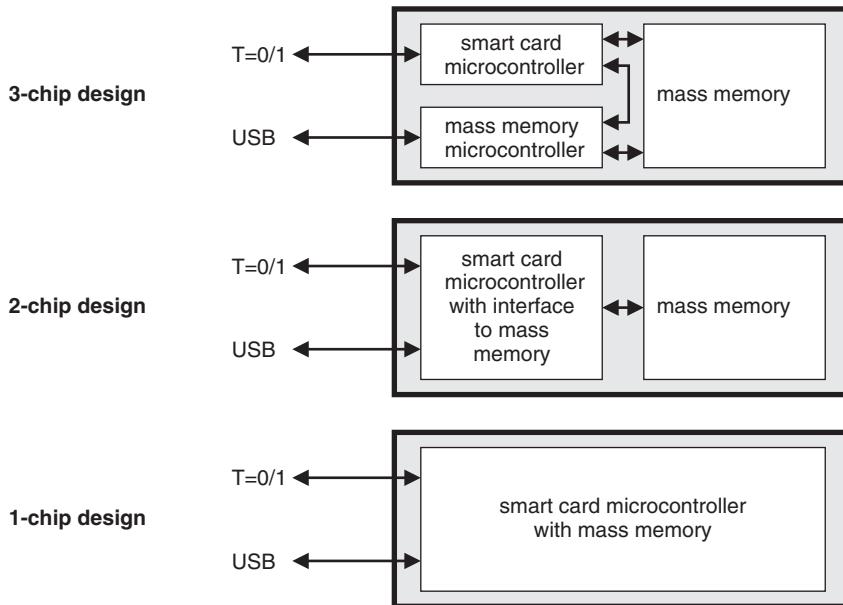
**Figure 2.11** Typical architecture of a processor card with a coprocessor, a contactless interface, and a contact interface. The latter interface is not present in a purely contactless processor card. The figure shows only basic energy and data flows and is not a detailed schematic diagram

demand for adding value-added services to electronic tickets or incorporating electronic ticket functionality in payment cards. Consequently, multifunction cards with integrated processors are being used more and more often, with the payment function usually implemented with conventional contact technology in order to utilize the existing payment card infrastructure. These cards have contacts as well as contactless coupling elements and are called dual-interface cards, as illustrated in Figure 2.11. The technology and operating principles of contactless smart cards are described in detail in Chapter 10, ‘Contactless Data Transmission’, on page 283.

### 2.3.5 Multi-megabyte cards

The growing popularity of flash memory as a replacement for hard disk drives in the PC realm is reflected in the smart card realm. It has become technically possible to produce processor cards with a memory capacity ranging from a few megabytes to somewhere in the gigabyte range. The standard T = 0 and T = 1 protocols specified in ISO/IEC 7816 are far too limited to cope with such large memory sizes. Consequently, such cards also have a USB or MMC interface.

In their simplest form, these cards are implemented as three-chip solutions. This does not require the development of new integrated circuits; instead, three standard ICs are wired together on a printed circuit board. The disadvantage of this approach is that three individual chips are more expensive in mass production than a design integrated in a single chip. In addition, the interconnections between the chips provide an additional target for attacks. For these reasons, two-chip and single-chip solutions can be expected increasingly often in the future, as shown in Figure 2.12 on the facing page.



**Figure 2.12** Three options for connecting NAND flash mass memory to a smart card microcontroller with a  $T = 0 / T = 1$  interface and a USB interface

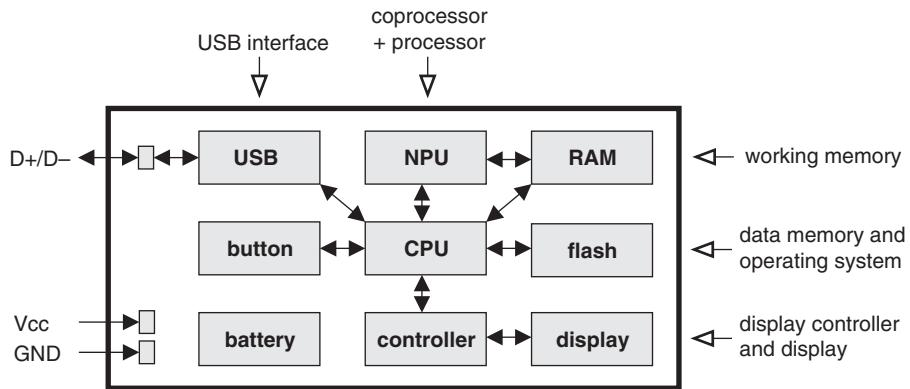
### 2.3.6 Security tokens

A security token is a hardware component containing a security chip, such as a smart card microcontroller, and it is usually connected to a USB port. A typical security token architecture is shown in Figure 2.13 on the next page. A USB security token thus combines the advantages of a smart card with the connection convenience of the USB interface, without any need for a card reader. Currently available USB security tokens usually contain several integrated circuits, such as a security chip and a memory chip. As the USB interface is specified as a new I/O interface for high data transmission rates in the latest smart card standards, there are already some smart card security microcontrollers available with an integrated USB interface. This enables economical, high-security single-chip solutions for security tokens. They differ from smart cards with the same functionality only in their physical form, as can be seen in Figure 3.47 on page 59.

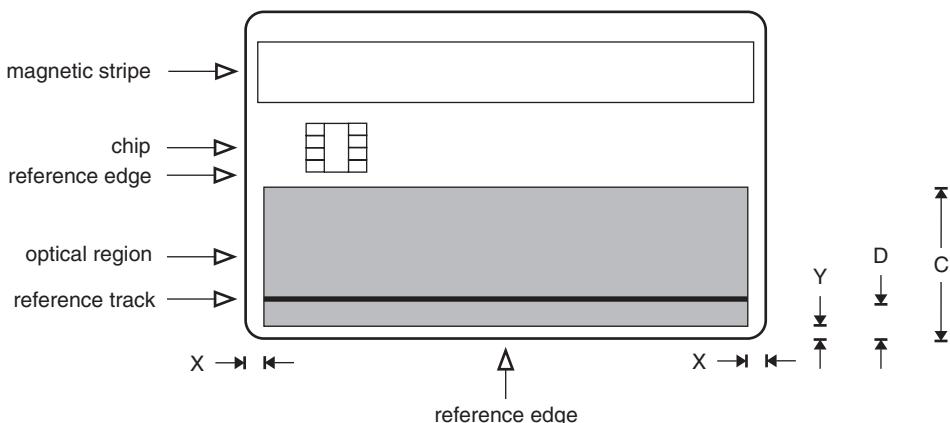
There are also security tokens with no direct connection to a PC. In addition to the security microcontroller, these security tokens usually have a keypad for PIN entry and a display to show a pseudorandom number used for authentication between the token and a server (one-time password). These one-time password tokens are also available in the form of small hardware components or smart cards with integrated displays.

## 2.4 OPTICAL MEMORY CARDS

Optical memory cards utilize optical memory technology, such as is used on CDs. Data on optical memory cards is protected against read errors by conventional methods, such as



**Figure 2.13** Typical architecture of a token with a coprocessor, display and USB interface, used for purposes such as generating one-time passwords. The figure shows only basic energy and data flows and is not a detailed schematic diagram



**Figure 2.14** Location of the optical storage area on an ID-1 card according to ISO/IEC 11694-2. The following dimensions are specified in the standard: C: 9.5–49.2 mm; D:  $5.8 \pm 0.7$  mm; X: 3 mm max. with MMI or 1 mm max. with PPM; Y:  $Y < D$  and  $\geq 1$  mm with PWM or 4.5 mm max. with PWI (PWM = pulse width modulation; PPM = pulse position modulation)

checksums, which of course reduces their useful memory capacity. The ISO/IEC 11693 and ISO/IEC 11694 standards define the physical characteristics of optical memory cards and the linear data recording technology.

Combining the large storage capacity of optical memory cards with the intelligence of smart cards opens up interesting new possibilities. For example, data can be written to the optical memory in encrypted form, with the key stored securely in the private memory of the chip. This protects the optically stored data against unauthorized access. Figure 2.14 shows the layout of a typical optical smart card with contacts, a magnetic stripe, and an optical data storage area. As you can see, the area available for optical storage is reduced by the chip



**Figure 2.15** A typical optical memory card with a net storage capacity (including error correction) of approximately 4 MB. The raw storage capacity (without error correction) is approximately 6 MB

contacts, which naturally restricts the total storage capacity. The magnetic stripe is located on the back the card. An actual optical memory card is shown in Figure 2.15.

The prices of optical memory cards are comparable to those of smart cards. However, reading and writing devices for optical memory cards are much more expensive than comparable devices for smart cards, which has severely restricted their use up to now. Optical memory cards are used in areas such as health care for storing patient data, where their large memory capacity allows even X-ray images to be stored on the card. The previously described multi-megabyte cards now provide memory capacity that is comparable to or even greater than that of optical memory cards. Whether optical memory cards will nevertheless manage to establish a position in the market remains to be seen.

# 3

## Physical Properties

The card body of a smart card inherits its fundamental properties from its predecessor, the familiar embossed card, which still dominates the market in the credit card sector. Technically speaking, such cards are simple plastic structures that are personalized by being embossed with a variety of user features, such as the name and customer number of the cardholder.

Later versions of these cards were provided with a magnetic stripe to enable simple machine processing. When the idea of adding chips to cards first arose, this existing type of card was used as the basis and a module with a memory chip or processor chip was embedded in the body of the card. Many standards relating to the card's physical properties are thus not specific to smart cards, but apply equally well to magnetic-stripe and embossed cards.

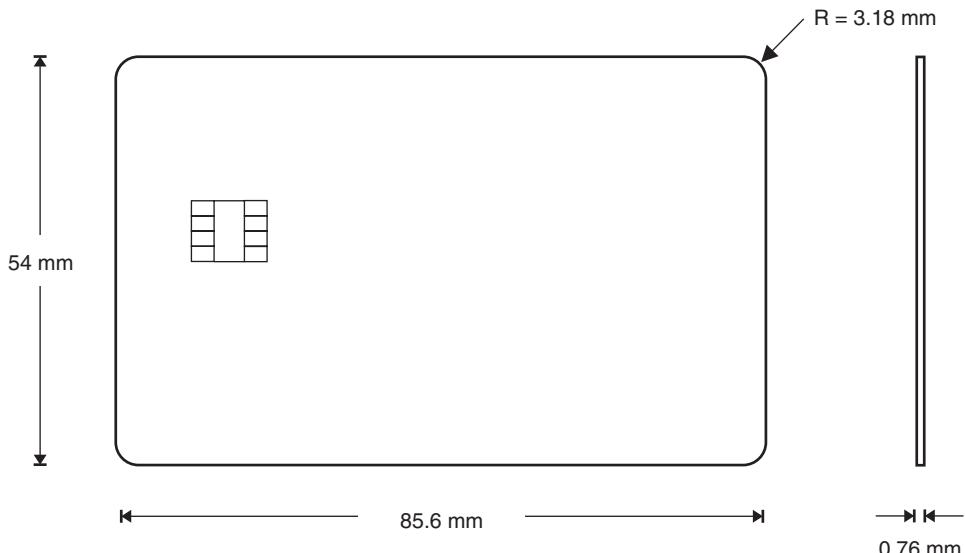
If you hold a smart card in your hand, the first thing you notice is its format. After this, you might see that it has set of contacts, although a contactless smart card may not have any visible electrical interface. The next feature to catch your eye might be a magnetic stripe, embossing or a hologram. All of these features and functional components are part of the physical properties of a smart card.

Most of the physical properties are actually purely mechanical in nature, such as the format of the card and its resistance to bending or twisting. These properties are familiar to every user from personal experience. In practice, however, physical properties such as sensitivity to temperature or light and resistance to moisture are also important.

The interaction between the body of the card and the implanted chip must always be considered, since only the combination of the two components makes a functional card. For instance, a card body designed for use at high ambient temperatures is of little benefit if its embedded microcontroller does not share this property. These two components must individually and collectively meet all of the relevant requirements, since otherwise high failure rates can be expected in use.

### 3.1 CARD FORMATS

Small cards with the typical smart card dimensions of 85.6 by 54 mm have been in use for a very long time. Almost all smart cards are still produced in this format, which is also the most



**Figure 3.1** The ID-1 format as specified in ISO 7810. Thickness:  $0.76 \pm 0.08$  mm; corner radius:  $3.18 \pm 0.30$  mm. The indicated dimensions show the size of the card excluding tolerances

familiar.<sup>1</sup> It is designated ID-1, and its size is specified in the ISO 7810 standard (see Figure 3.1). An example of an ID-1 card is shown in Figure 3.8 on page 34.

The ID-s standard originated in 1985 and thus has nothing to do with smart cards as we know them today, as can easily be seen from the abbreviation ‘ID’, which stands for ‘identification’. This standard simply describes an embossed plastic card with a magnetic stripe that is intended to be used for the identification of a person. When it was written, no one had thought of putting a chip in the card. The presence of a chip and location of its contacts on the card were only defined several years later in other standards.

With the diversity of cards available today, which are used for all possible purposes and have a wide range of dimensions, it is often difficult to determine whether a particular card is actually an ID-1 smart card. In addition to the embedded chip, one of the best identifying features is the thickness of the card. If this measures 0.76 mm and the card has an embedded microcontroller, it can be regarded as a smart card in the sense of the ISO standard.

The conventional ID-1 format has the advantage of being very easy to handle. The card’s format is specified such that it is not too large to be carried in a wallet, but not so small that it is easily lost. In addition, the card’s flexibility makes it more convenient than a rigid object.

Nevertheless, this format does not always meet the demands of modern miniaturization. Mobile telephones typically weigh less than 200 g and are only half the size of a packet of tissues. It thus became necessary to define a smaller format in addition to the ID-1 format, in order to address the needs of small terminal devices.

This led to the development of additional card formats, all of which can be produced by stamping them from ID-1 cards or breaking them free from an ID-1 card. The cards used in

<sup>1</sup> Telecommunication cards are also produced in ID-000 format now. See also Section 14.4.4, ‘Direct plug-in production’, on page 588



**Figure 3.2** A smart card in plug-in format, from which the user can break out a card in mini-UICC format

mobile telephones can be very small because they are usually plugged into the device only once and remain there forever. The ID-000 format was defined to suit this purpose, and it bears the descriptive name ‘plug-in’. This format is only used in mobile telephones and as a security module in terminals.

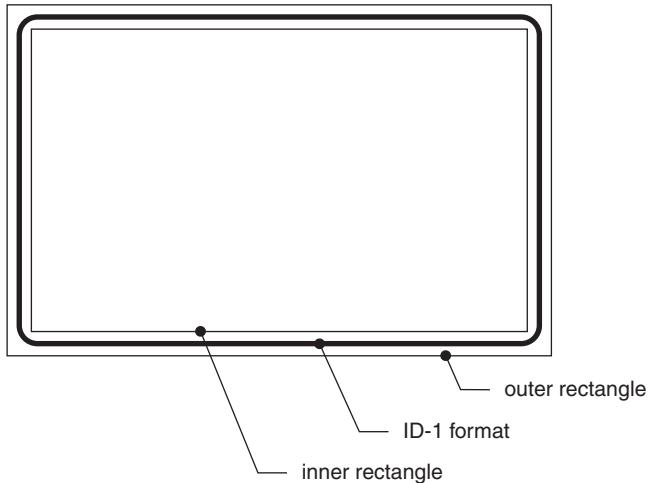
However, cards in ID-000 format are inconvenient to handle in production and by end users, which led to the development of an additional format. This format is designated ID-00 or ‘mini-card’. Its dimensions are approximately halfway between those of ID-1 and ID-000 cards. This type of card is more convenient to handle and less expensive to produce because it can be stamped from the ID-1 format. The ID-00 format was defined in the mid-1990s, but it has not yet become established either nationally or internationally.

Starting around 2003, ongoing miniaturization of mobile telephones led to the demand for a card format even smaller than the plug-in format. This was discussed for a long time under the working name ‘Third Form Factor’ (3FF), a term that is still often used for this new card size. The official ETSI name is ‘mini-UICC’, which designates a smart card format that is only slightly larger than a module with eight contacts. Mini-UICC cards can also be obtained by breaking them free from a larger ID-1 or plug-in card, as illustrated in Figure 3.2. They are already being produced in quantity for various network operators.

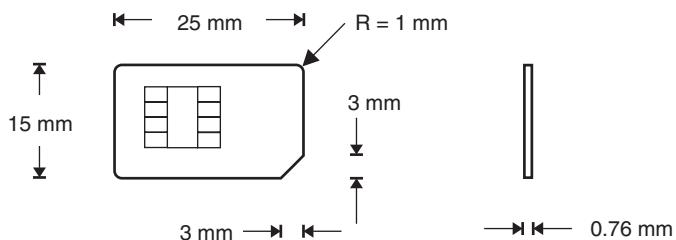
Developments in the payment card sector have paralleled the constant miniaturization of mobile telecommunication cards. Visa specified the Visa Mini card format in 2005. Cards in this format can also be produced by breaking them free from cards in standard ID-1 format, as illustrated in Figure 3.9 on page 35. This makes it possible to manufacture cards in the new format without modifying existing production lines for ID-1 cards. MasterCard has also defined a format with a similar size, which is called ‘mc2’. Figure 3.12 on page 36 shows the two card formats in comparison.

The formats are defined in the relevant standards in a way that simplifies measuring the card dimensions, as illustrated in Figures 3.4 and 3.5 on the following page and Figure 3.9 on page 35. For example, the height and width of an ID-1 card must be such that it fits between two concentric rectangles (ignoring the rounded corners). The outer rectangle has a width of 85.72 mm (3.375 inch) and a height of 54.03 mm (2.127 inch). The inner rectangle has a width of 85.46 mm (3.365 inch) and a height of 53.92 mm (2.123 inch). The thickness must be 0.76 mm (0.03 inch), with a tolerance of  $\pm 0.08$  mm ( $\pm 0.003$  inch). The corner radii and card body thickness are dimensioned conventionally. Based on these definitions, the dimensions of an ID-1 card can be represented as shown in Figure 3.3 on the following page.

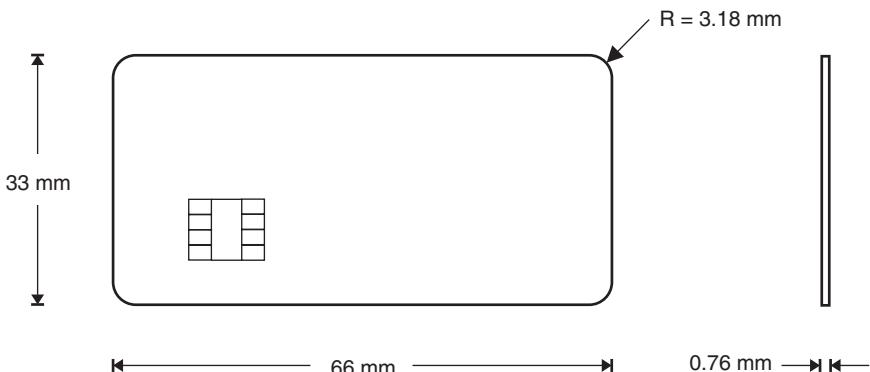
The ID-000 format as specified in TS 51.011 and TS 102 221 is also defined using two concentric rectangles. As this format originated in Europe (based on the GSM mobile telephone system), the basic dimensions are metric. The dimensions of the outer rectangle are 25.10 mm (width) by 15.10 mm (height). The inner rectangle has a width of 24.90 mm and a height



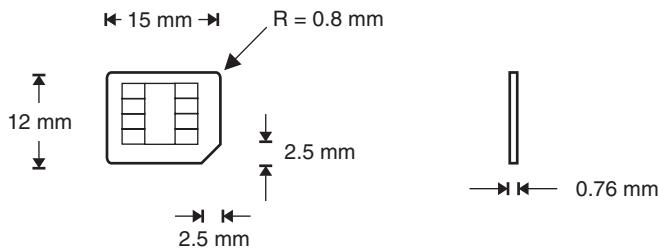
**Figure 3.3** Dimensions of the ID-1 card format as specified by ISO/IEC 7810



**Figure 3.4** The ID-000 format. Thickness:  $0.76 \pm 0.08$  mm; corner radius:  $0.8 \pm 0.10$  mm; diagonal corner  $2.5 \pm 0.1$  mm. The indicated dimensions show the size of the card excluding tolerances



**Figure 3.5** The ID-00 format. Thickness:  $0.76 \pm 0.08$  mm; corner radius:  $3.18 \pm 0.30$  mm. The indicated dimensions show the size of the card excluding tolerances



**Figure 3.6** The mini-UICC format. Thickness:  $0.76 \pm 0.08$  mm; corner radius:  $0.8 \pm 0.10$  mm; diagonal corner  $2.5 \pm 0.8$  mm. The indicated dimensions show the size of the card excluding tolerances

of 14.90 mm. The bottom right-hand corner of a plug-in card is beveled at angle of  $45^\circ$  to facilitate correct insertion of the card into the card holder.

The ID-00 format is also based on metric measurements, and its maximum and minimum dimensions are again defined by two concentric rectangles. The outer rectangle has a width of 66.10 mm and a height of 33.10 mm. The inner rectangle has a width of 65.90 mm and a height of 32.90 mm.

The mini-UICC format is the smallest possible smart card format that still allows reliable module embedding. Its dimensions are also based on the metric system and specified such that a mini-UICC card can be produced by breaking it free from an ID-000 card.

In the payment sector, the need to define a new form factor in order to create a differentiating feature relative to competitive products arose around 2003. The new card should also be suitable for carrying on a key ring, for which purpose it is required to have a hole. The result is the Visa Mini format, which is issued by the Visa credit card company. It has a width of 65.5 mm and a height of 40 mm, with a thickness of 0.76 mm, as illustrated in Figure 3.9.

To further differentiate the various card types, cards with a wide variety of dimensions are issued in the payment sector in particular, as illustrated in Figures 3.10 and 3.11 on page 35. Here the most important consideration is that these cards must also be usable in the existing infrastructure. This primarily involves magnetic stripes and embossing. The advantage of contactless smart cards becomes quite evident here, since they can take almost any desired form as long as a microcontroller and antenna can be fitted in the card body.

The ID-1, ID-00, ID-000, mini-UICC and Visa Mini formats can be produced from the larger card formats by stamping. This is especially important for card manufacturers, since it allows the production process to be designed for a single, standard format (ID-1).

For instance, card manufacturers commonly produce card blanks in only one format (preferably ID-1), embed modules in them and fully personalize them. Depending on the specific application area of the produced cards, they can then be converted to the desired format in a subsequent production operation.

Alternatively, the format may be modified later by the customer. This has become common practice with cards for mobile telephones. The customer receives an ID-1 card that is pre-punched so it can be converted onto an ID-000 card by breaking it free from the larger card. In another technique, the ID-000 card is stamped entirely free from the ID-1 body and attached to the surrounding portion of the ID-1 card by adhesive tape on the side without contacts. The customer can thus convert it into a card with the appropriate format for the intended use, while the manufacturer has only to produce and ship one card format. The same approach is used if the customer needs to be able to convert the card into the smallest currently available format: mini-UICC.



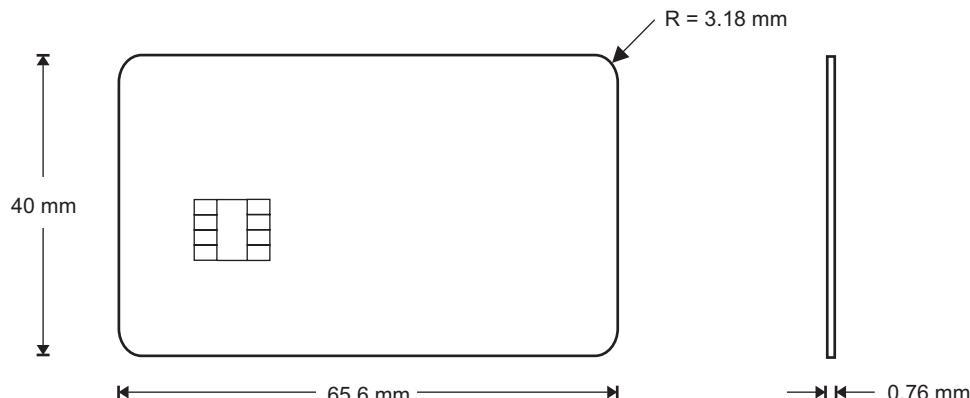
**Figure 3.7** Example of a mobile telephone card in ID-1 format, which the customer can convert into an ID-000 or mini-UICC card if necessary by breaking free the smaller-format version (Reproduced with permission from Giesecke & Devrient)



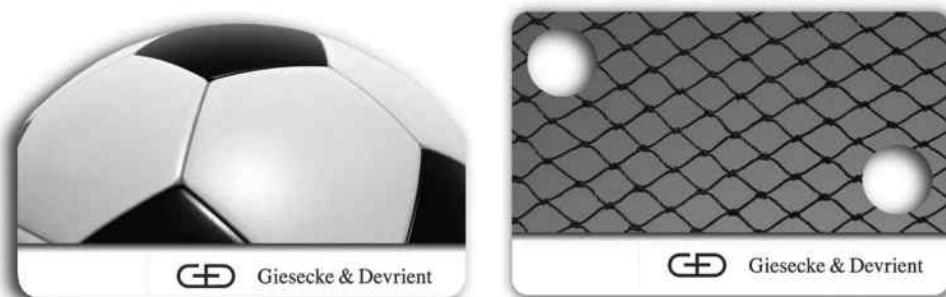
**Figure 3.8** Example of a payment card in ID-1 format, which the customer can convert into a Visa Mini card if necessary by breaking free the smaller-format card (Reproduced with permission from Giesecke & Devrient)

However, the usual card format has some disadvantages for certain applications. In such cases, other form factors can be used, such as a USB plug with an integrated smart card microcontroller, as illustrated in Figure 3.13 on page 36. Generally speaking, the logical behavior of such smart card variants is fully equivalent to that of the usual forms.

There are also methods available for integrating smart card functionality with other components on a printed circuit board if necessary. This is typically done by fitting the smart card microcontrollers in conventional IC packages and using automated equipment to fit them on circuit boards. Some examples of SMD modules for smart card chips are shown in Figure 3.14 on page 37. The SOP8 package is often used for this purpose due to its very compact size, which is usually an important requirement in such applications. The SOP8 package has a length of 5.0 mm and a width of 4.4 mm. Its width including solder leads is 6.2 mm, with a component height of 1.5 mm. Other packages are also used, such as QFN (quad flat pack, no leads).



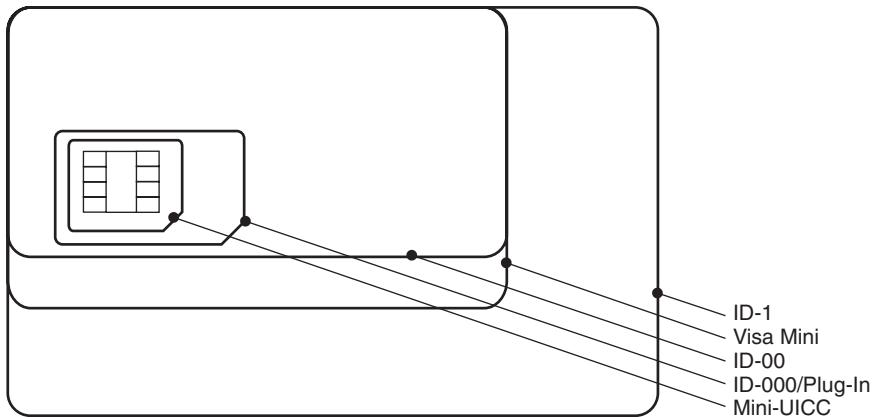
**Figure 3.9** The Visa Mini format. This card is based on the ID-1 format specified in ISO 7810 and has a thickness of  $0.76 \pm 0.08 \text{ mm}$ ; the corner radius is  $3.18 \pm 0.30 \text{ mm}$ . The indicated dimensions show the size of the card excluding tolerances



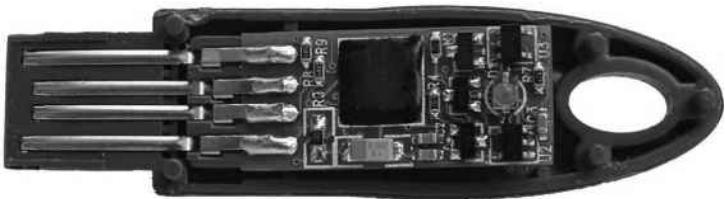
**Figure 3.10** Two examples of smart cards with special shapes. They are produced as ID-1 cards and then converted to the final shape by stamping (Reproduced with permission from Giesecke & Devrient)



**Figure 3.11** Examples of smart cards with special shapes. The card on the left has rounded corners, while the card on the right can be made smaller by breaking off part of the card (Reproduced with permission from Giesecke & Devrient)



**Figure 3.12** Relative sizes of the ID-1, ID-00 and ID-000 formats



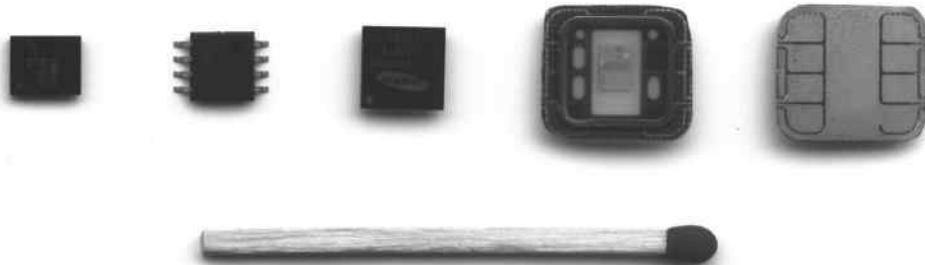
**Figure 3.13** Example of an alternative smart card form factor. This photo shows a USB plug with a soldered-in smart card microcontroller and the necessary interface components, which has been opened up to reveal its internal components

## 3.2 CONTACT FIELD

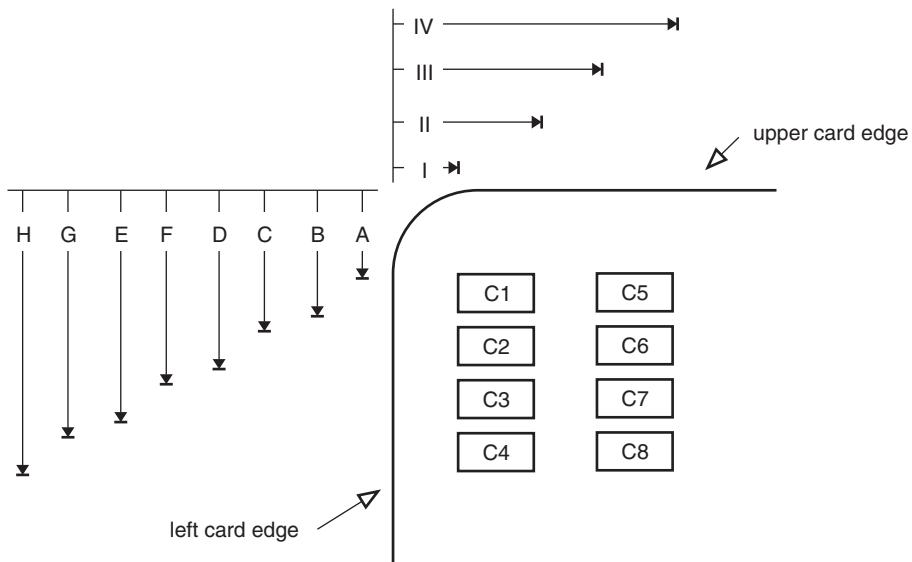
The main difference between a smart card and other types of cards is the embedded microcontroller. If electrical power and data are transferred by direct electrical connection to the microcontroller, the card must also have electrical contacts. They consist of six or eight gold-plated contacts, which can be seen on every standard smart card. The location of these contacts on the card body and the contact dimensions are specified by the ISO 7816-2 standard, the first version of which dates from 1988.

In France, a national standard generated by AFNOR was already in use long before ISO 7816-2 was issued. It specifies a slightly higher location for the contacts than the ISO standard. This location is also included in the ISO standard as a ‘transitional contacts location’, although the standard recommends that this location not be used in the future. However, there are still cards in France with this contact location, so it is not likely that it will disappear all that quickly.

The absolute location of the contact field is in the upper left corner of the card body, as is illustrated in Figure 3.15 on the next page.



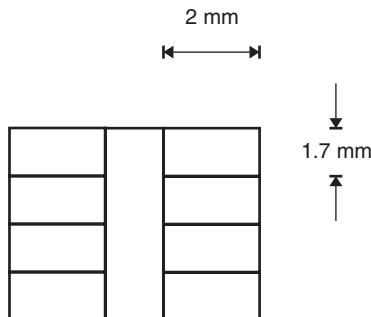
**Figure 3.14** Examples of various types of SMD modules used to package smart card microcontrollers. The first and third components from the left are QFN packages, while the second component is an SOP8 package. For comparison, the front and back sides of a conventional module are shown at the right



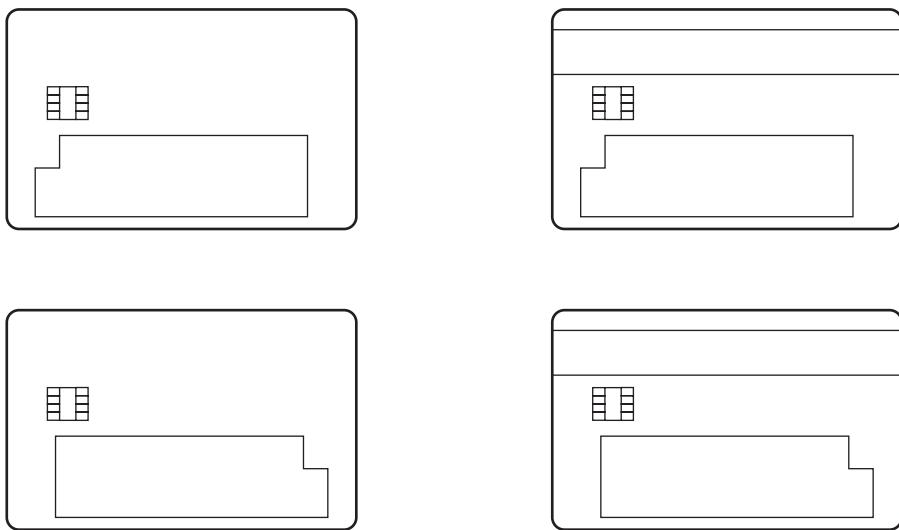
**Figure 3.15** Location of the contacts on the card body (drawing not to scale). The following minimum and maximum distances are standardized: I max: 10.25 mm; II min: 12.25 mm; III max: 17.87 mm; IV min: 19.87 mm; A max: 19.23 mm; B min: 20.93 mm; C max: 21.77 mm; D min: 23.47 mm; E max: 24.31 mm; F min: 26.01 mm; G max: 26.85 mm; H min: 28.55 mm

The minimum dimensions of any contact are 1.7 by 2 mm (height by width), as shown in Figure 3.16 on the next page. The maximum dimensions of any contact are not specified, but they are of course limited by the fact that the individual contacts must be electrically isolated from each other.

The location of the module in the card body is specified in the standard. The locations of the magnetic stripe area and the area reserved for embossing are also specified precisely (see ISO 7811). All three of these components may be present on a single card. However, in this case the following mutual relationships must be taken into account: (a) if only a chip and an embossing field are present, they may be located on the same side or on opposite sides of the card; (b) if a magnetic stripe is also present, it and the embossing area must be located on opposite sides of the card. Figure 3.17 on the next page shows the various options.



**Figure 3.16** Minimum contact dimensions as specified in ISO 7816-2



**Figure 3.17** The various possible arrangements of the card components (chip, embossing field and magnetic stripe) according to the ISO 7816-2 standard

### 3.3 CARD BODY

The materials, construction and production of the card body are effectively determined by the card's functional components (see Figure 3.18 on page 40), as well as by the stresses to which it is subjected during use. Typical functional components of a card include:

- magnetic stripe
- signature panel
- embossing
- imprinting of personal data via laser beam (text, photo, fingerprint)
- hologram

- security printing
- invisible authentication features (e.g. fluorescence)
- chip with contacts or antenna

Clearly, even a relatively small card with a thickness of only 0.76 mm must sometimes have a large number of functional components. This places severe demands on the quality of the materials used and the manufacturing process.

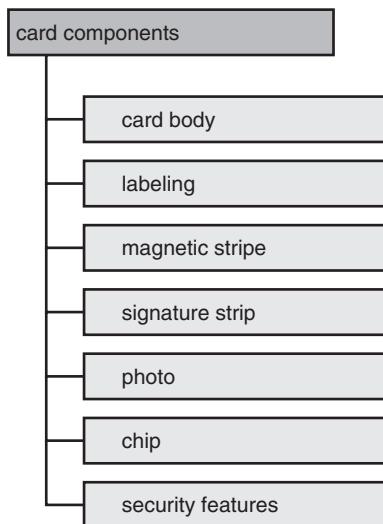
The minimum requirements relating to card robustness are specified in ISO standards 7810, 7813 and 7816 Part 1. The requirements essentially relate to the following areas:

- mechanical robustness of the card and contacts
- temperature resistance
- surface profile of the card
- electrostatic discharge
- electromagnetic susceptibility
- ultraviolet radiation
- X-ray radiation

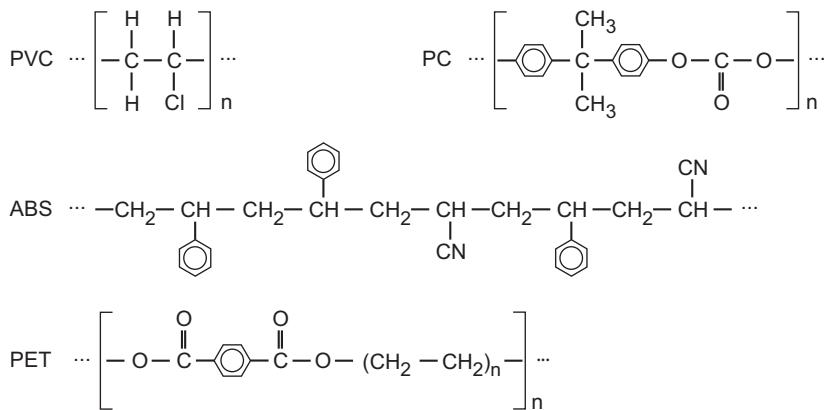
The ISO/IEC 10373 standard specifies test methods for many of these requirements, to enable users and card manufacturers to objectively test card quality. The bending and twisting tests are particularly important for smart cards, since the chip, which is as fragile and brittle as glass, is a delicate foreign object in the elastic card. Special structural features are required to protect it against the mechanical stresses produced by bending and twisting the card. Chapter 15, ‘Quality Assurance’, on page 633 contains a detailed list of tests and the methods used to perform them.

## 3.4 CARD MATERIALS

A variety of materials are used for card bodies. Their main characteristics are listed in Table 3.1 on page 41, and the structural formulas of the most important materials are shown in Figure 3.19 on the next page. The first material employed for ID cards, which is still widely used, is polyvinyl chloride (PVC), an amorphous thermoplastic material. It is the least expensive of all the available materials, easy to process, and suitable for a wide range of applications. It is used worldwide for credit cards. Its drawbacks are a limited lifetime due to physical deterioration and limited resistance to heat and cold. PVC is used in sheet form to manufacture cards, since injection molding is not possible. PVC is considered to be environmentally hazardous because the feedstock, vinyl chloride, is a known carcinogen. In addition, hydrochloric acid and (under unfavorable conditions) possibly dioxins are released when it is burned. In addition, heavy-metal compounds are often used as stabilizers. Nonetheless, PVC is still by far the most widely used material for cards. This is primarily due to its low cost and good processing characteristics. However, it is used less and less each year due to its undesirable environmental properties. Many card issuers have now decided not to use PVC for reasons of environmental policy.



**Figure 3.18** Classification of card components



**Figure 3.19** Structural formulas of the most important plastics used for card bodies

To avoid the drawbacks of PVC, acrylonitrile butadiene styrene (ABS) has been used for some time to make cards. It is also an amorphous thermoplastic that is distinguished by its high strength and resistance to temperature extremes. Consequently, it is commonly used for mobile telephone cards, which for obvious reasons may be subjected to relatively high temperatures. ABS can be processed readily in sheet form or by injection molding, and it does not have any known environmental drawbacks.

Polycarbonate (PC), which incidentally is also the substrate material of CDs and DVDs, is used for applications where high strength and durability are required. Due to its high thermal stability, relatively high temperatures are needed to apply holograms or magnetic stripes using the hot-stamp process. This can easily cause problems due to the limited thermal

**Table 3.1** Summary of the characteristics of the usual card body materials. The relative cost is based on the cost of PVC

Properties	PVC	ABS	PC	PET
Primary use	Credit cards	Mobile phone cards	ID cards	Health insurance cards
Principal feature	Inexpensive	Thermally stable	Durable	Environmentally friendly
Card production	Sheet only	Sheet and injection molding	Sheet and injection molding	Sheet and injection molding
Heat tolerance	65–90°C	75–100°C	160°C	Up to 80°C
Cold tolerance	Moderate	High	Moderate	Moderate
Mechanical stability	Good	Good	Good	Very good
Embossing	Good	Poor	Good	Good
Printing	Good	Moderate	Moderate	Good
Hot stamping (e.g. for holograms, etc.)	Good	Good	Difficult	Good
Laser engraving	Yes	Poor	Good	Good
Typical lifetime	2 years	3 years	5 years	3 years
Cost	1	2	7	2.5
Environmental aspects	Burning may release dioxins Stabilizers contain heavy metals		Burning does not release hazardous materials	Currently the most environmentally friendly card material Burning does not release hazardous materials
Special aspects	Negative public image Low thermal stability		Low scratch resistance	

stability of the materials being applied. The main drawbacks of polycarbonate are its low resistance to scratching and high cost relative to other card materials. A further drawback is that phosgene and chlorine are needed for the production of polycarbonate, and both of these materials are environmentally problematical. Polycarbonate cards can be easily recognized by the characteristic ‘tinny’ sound they produce when dropped on a hard surface.

An environmentally friendly material that is mainly used as a PVC substitute is polyethylene terephthalate (PET), which has been used for a fairly long time as a packaging material. It is commonly known as polyester. This thermoplastic material is used in smart cards in both its amorphous form (A-PET) and its crystalline form (PETP). Both types are suitable for processing in sheet form or by injection molding. However, PETP is difficult to laminate, which makes additional processing steps necessary in the manufacturing process.

Repeated attempts have been made to find new or better materials for card bodies in addition to the four usual materials (PVC, ABS, PC and PET). One example is cellulose acetate, which although having good environmental properties has up to now proven to be poorly suited to the mass production of cards. Truly different materials, such as paper, have been frequently discussed, but as yet they have never been used in any significant quantity. The requirements imposed on cards in terms of cost, durability and quality are after all very high, and they can presently only be met by plastics.

Although it does not represent a true alternative to plastic card bodies, an interesting (or at least remarkable) field trial was carried out in 1996–97 in Denmark by Danmønt.<sup>2</sup> Around 6 000 smart cards with a card body made from laminated and printed birch (eight laminations, each 0.1 mm thick) were issued. Although these cards did not pass the various tests specified in ISO 10373, such as the bending and twisting tests, and they were naturally not suitable for embossing, around 90 % of their users responded positively and said that they experienced no problems with their cards. Unfortunately, laminated birch cards are not especially innovative from an environmental perspective because the layers must be laminated with a synthetic glue and the usual printing processes are still necessary.

### 3.5 CARD COMPONENTS AND SECURITY FEATURES

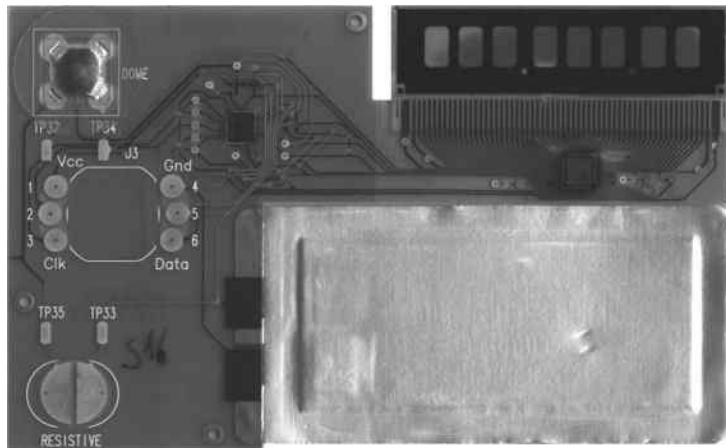
Smart cards are primarily used to provide authorization for specific actions or identify the cardholder, so security features on the card body are often needed in addition to the embedded chip. To enable the authenticity of the card to be checked by humans as well as by machines, many security features are based on visual elements. However, some security features employ a modified smart card microcontroller and thus can only be checked by a computer. In contrast to the security features used with microcontrollers, the usual features for human verification of card authenticity are not based on cryptographic methods (such as mutual authentication). Instead, they are primarily based on secret materials and production methods or technological processes whose mastery requires a large amount of effort and considerable expertise or is technically difficult.

Particularly with regard to new card components, there is considerable potential for innovations in the near future because one of the possible evolution paths of smart cards is in the direction of additional integrated components such as a keypad, display, solar cell, and battery, as illustrated in Figures 3.20 and 3.21 on the next page.

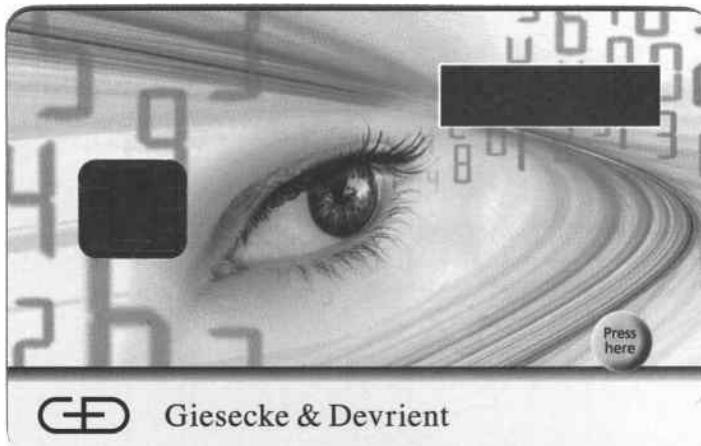
#### 3.5.1 Guilloche patterns

A somewhat more complicated technique is to insert a foil printed with color guilloche patterns under the transparent outer foil of the card. Guilloche patterns are decorative patterns consisting of very fine interwoven lines, usually circular or oval, such as are found on some bank notes and share certificates (see Figure 3.22 for an example). These patterns have such fine structures that they can presently only be produced by special printing processes and are thus difficult to copy.

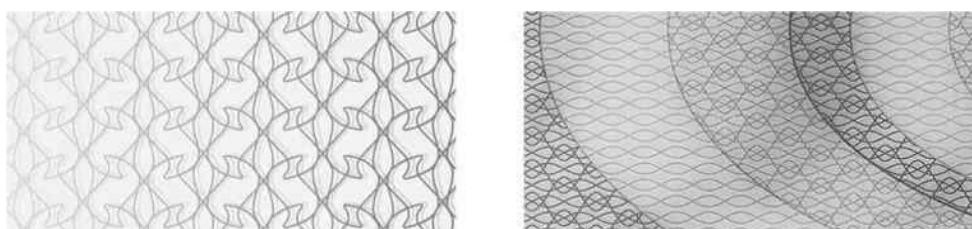
<sup>2</sup> [a la Card 97]



**Figure 3.20** Inlay foil for a super smart card. A pushbutton switch is located at the upper left, with the connections for the actual smart card microcontroller below. A display is located at the upper right, with the display controller below. The rectangular component at the bottom right is the battery that powers the card (Reproduced with permission from Giesecke & Devrient)



**Figure 3.21** Inlay foil for a super smart card for contactless and contact communication. A pushbutton switch for confirming transactions is at the bottom right, and a display for showing the purse balance and other data is at the upper right (Reproduced with permission from Giesecke & Devrient)



**Figure 3.22** Examples of originally multicolored guilloche patterns (highly enlarged). They cannot be printed using ordinary commercial printing processes (Reproduced with permission from Giesecke & Devrient)

### 3.5.2 Signature panel

A very simple way to identify the cardholder is to use a signature panel bonded to the card, as is common with credit cards. Once such a panel has been signed, it cannot be altered, so it is erasure-proof. A very fine colored pattern printed on the panel makes any attempt to glue something on top of the panel immediately apparent. The signature panel is permanently bonded to the card body by using a hot-glue process to attach a printed paper strip to the card. Alternatively, the signature panel can be part of the top layer of the card body, laminated into the card when it is assembled.

### 3.5.3 Microtext

Another technique that is based on the security provided by fine printed structures is microtext lines. They look like simple lines to the naked eye, but they can be recognized as text under a loupe (see Figure 3.23 for an example). Like guilloche patterns, microtext cannot be photocopied.



**Figure 3.23** A highly enlarged example of microtext, which cannot be printed using ordinary commercial printing processes (Reproduced with permission from Giesecke & Devrient)

### 3.5.4 Ultraviolet text

To avoid affecting the visible layout of the card, control characters or control numbers can be printed on the card using ink that is only visible under ultraviolet light.

### 3.5.5 Barcode

For storing a small amount of data, a barcode can be printed on the surface of the card using laser printing or thermal-transfer printing. The advantage of barcodes is that they can be read automatically at close range using optical equipment. The barcodes used on smart cards include not only the widely use one-dimensional type, but also two-dimensional barcodes in the form of stacked or matrix barcodes. A two-dimensional barcode, such as PDF 417, can easily encode up to 1000 bytes. If an integrated Reed–Solomon code is used for error correction, the data can be recovered even if up to 25 % of the barcode area is unreadable.

### 3.5.6 Hologram

A hologram integrated in the card is a security feature that is now familiar to all card users (see Figure 3.24 for an example). The security of holograms is primarily based on the fact that they are produced by only a few companies in the world and their availability is restricted.



**Figure 3.24** Example of a hologram attached to a substrate printed with guilloche patterns (originally multicolor) (Reproduced with permission from Giesecke & Devrient)

The holograms used for smart cards are called ‘embossed’ holograms, since they must be recognizable under diffuse reflected daylight illumination. For this reason, they are also called white-light reflection holograms. By contrast, a conventional transmission hologram requires coherent laser light for proper viewing. Supplementary security features that can only be seen with laser light are sometimes incorporated in the hologram. In order to produce an embossed hologram, a master hologram must first be generated using the conventional holographic recording method. A master embossing stamp is then prepared from the master hologram using a transfer process. The embossing stamp contains the microstructures that will produce the subsequent embossed holograms. Daughter stamps are made from the master stamp using electroplating processes, and these daughter stamps are used to emboss the hologram structure in plastic films. These films are then vapor-coated with aluminum to produce the well-known white-light reflection holograms.

The hologram is permanently bonded to the card body and cannot be removed without destroying it. The bonding can be performed using either lamination or the roll-on method. With the latter method, a hologram located on a carrier film is pressed onto the card by a heated roller. The carrier film is then pulled off, and the hologram remains permanently welded to the plastic card body. A third option is the hot-stamp method, which is similar to the roll-on method but uses a heated stamp instead of a heated roller.

### 3.5.7 Kinegram

Kinegrams have same structure as holograms, but they show different images when viewed at different angles. Kinegrams are just as hard to forge as holograms, and they have the advantage that they are more readily recognized and thus can be checked more quickly.

### 3.5.8 Multiple laser image (MLI)

A multiple laser image is a sort of kinegram that is very similar to a simple hologram. It is based on an array of lenses embossed in the surface of the card, some of which are blackened by a laser. The main difference between an MLI and a hologram is that card-specific information is shown in the small MLI image. For instance, this technique can be used to mark an individual card with the cardholder's name in the form of a kinegram.

A small trick (which incidentally also works with all personalization features of this sort) is used to ensure that the authenticity of the MLI is also checked when the card is verified manually. Some of the information necessary for the verification process (such as the expiry date), which must be read during the verification process, is stored in the MLI. As a result, the person who verifies the card more or less automatically checks the authenticity of the feature by reading this information. This would not necessarily happen if the feature were not merged into the verification process in this manner.

### 3.5.9 Embossing

Another way to add user data to a card is to emboss characters on the card, as shown for example in Figure 3.25. This is done by hammering metal letter punches against the card with considerable force. In principle, this works the same way as a mechanical typewriter. Nowadays there is only one reason for using embossing, but it is very important in practice: the characters of embossed cards can be copied relatively easily onto preprinted forms using carbon paper. On the global scale, this is still the most widely used method of paying with a credit card.

It is very easy to manipulate embossed characters, since the plastic can be flattened by moderately heating the embossed characters (using an iron, for example). Different characters can then be embossed in place of the original ones. To prevent this sort of manipulation, some



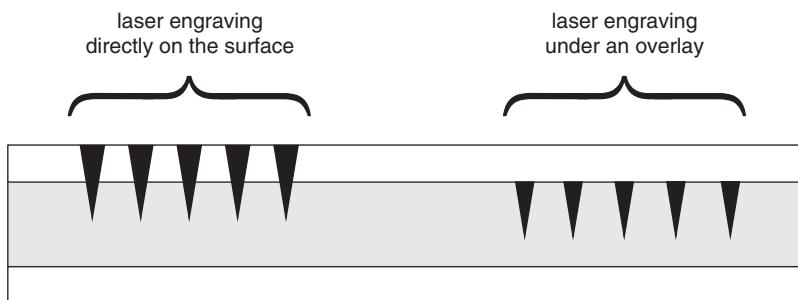
**Figure 3.25** An example of a credit card with embossing and an antenna for contactless communication. This is a functional sample without issuer-specific printing (Reproduced with permission from Giesecke & Devrient)

of the embossed characters are often placed in the hologram, which will be destroyed if it is heated.

### 3.5.10 Laser engraving

Darkening a special plastic layer by charring it with a laser beam is called laser engraving (see Figure 3.26), or simply lasering. In contrast to embossing, this is a secure way to write information on an individual card, such as the cardholder's name and the card number. It is secure because the necessary equipment and the knowledge of how to use it are not readily available.

Two different methods are used for laser engraving: vector engraving and raster engraving. In the vector method, the laser beam is directed along its path without interruption. This is very well suited to writing characters and has the advantage of being quick. With the raster method, by contrast, a large number of adjoining points are blackened to produce an image, similar to the operation of an inkjet or dot-matrix printer. This method is primarily used to place a picture on the card. Although it has the advantage of high resolution, which allows details to be reproduced well, it has the disadvantage of being very time-consuming. For instance, it takes approximately ten seconds to laser engrave a standard-quality passport photograph.

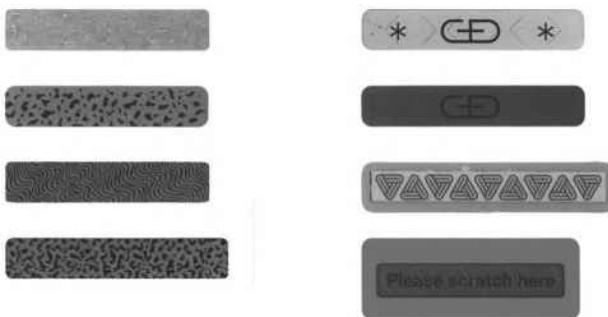


**Figure 3.26** Cross section of laser engraving on a card (not to scale). Laser engraving can be performed either on the surface of the card or in an internal layer below a cover foil that is transparent to the laser light

### 3.5.11 Scratch field

It is sometimes necessary to mark a smart card with a confidential, sealed multidigit number or character string. This card-specific data can be used for purposes such as enabling a particular function or crediting a balance. A scratch field can be laminated onto the card for this purpose. The scratch field consists of a printable substrate covered by a seal layer that can be scratched off as illustrated by several examples in Figure 3.27.

The confidential data is printed on the substrate, which is then covered by the seal layer. The seal layer must have several specific properties. It must be possible for everyone to readily recognize that the layer has been removed, even if this is done with the aid of heat or cold. In addition, the printed data must not be visible from the rear if the entire scratch field is removed from the card body. It must be impossible to read the printed data underneath the



**Figure 3.27** Several types of scratch fields (multicolored in the original) (Reproduced with permission from Giesecke & Devrient)

seal layer by using visible, ultraviolet or infrared light. This is essential in order to ensure that the confidential data under the seal layer cannot be spied out before the user scratches off the protective layer.

### 3.5.12 Thermochrome display

There are applications in which it is desirable to be able to modify the text and imagery on a card from time to time. A good example is a student identification card in the form of a smart card that must be renewed every six months. Ideally it should be possible to read the expiry date without technical aids, which means that it should be possible to print it on the card in addition to storing it in the chip.

Smart cards with microcontroller-driven displays are currently technically possible, but they are still too expensive for large-scale use. Thermochrome (TC) displays are a simple alternative that have some drawbacks compared with real displays, but which are inexpensive and already available. A TC display is a supplementary card component on which characters and imagery can be printed reversibly (printed and subsequently reprinted) using a special card reader.

The operating principle is relatively simple: a print head with a resolution of 200 or 300 dpi, such as is used in thermal-transfer and dye-sublimation printers, is used to heat individual pixels on a thermochrome strip laminated to the card, which has a thickness of 10 to 15 µm. The strip turns dark at the points where it is heated to 120 °C. The darkened areas can be restored to a nearly transparent state by heating the entire TC strip, which amounts to erasing the strip.

The thermochrome method is presently the only economical manner to provide card users with temporary information on the surface of the card that can be read without special equipment. Its major disadvantages are that it is subject to fraud and that it requires special card terminals with built-in thermochrome printer mechanisms.

### 3.5.13 Moduliertes Merkmal (modulated feature) method

In 1979, the German banking industry decided to incorporate a machine-readable security feature in all German Eurocheque (ec) cards. After several different methods were tested, the Moduliertes Merkmal (MM) method (developed by the firm GAO, now known as Giesecke &

Devrient) was selected as the security method for German ec cards. This security feature is still used in all German Eurocheque cards, even though they are now equipped with microcontroller chips. The objective of this security feature is to prevent unauthorized copying or manipulation of the magnetic-stripe data.

The MM method is a typical example of a secret and very effective security feature. It has been used for more than two decades in millions of cards. Its basic structure is described in summary form in an article by Siegfried Otto [Otto 82].

The name ‘MM method’ comes from the German term *moduliertes Merkmal* (modulated feature), which can be understood as referring to a machine-readable substance that is incorporated inside the card body [Meyer 96]. A card is verified by reading its MM code using a special sensor and passing the code to a security module called the ‘MM box’. The MM box also receives all the data from the magnetic stripe, in particular the MM check value, which is also stored on the magnetic stripe. Inside the MM box, a one-way function based on the DES algorithm is used to calculate a value from the magnetic-stripe data and the MM code. If the result of this calculation is the same as the MM check value, it can be concluded that the magnetic-stripe data matches the card.

If a valid set of magnetic-stripe data is written to a blank card, this will be detected by the fact that the blank card does not have the MM feature. Copying the magnetic-stripe data from one ec card to another ec card can also be detected because the MM check value will not match. The MM feature is invisible, and the details of how it works and where it is located in the card are secret. In addition, it is produced using materials and technology that are not commercially available.

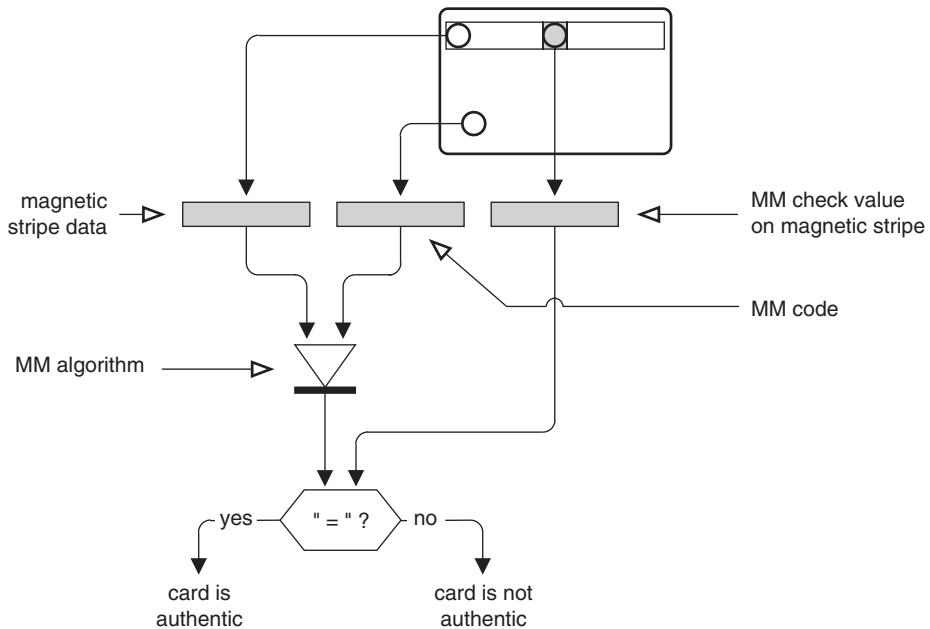
A MM box is built into every German bank machine (ATM), as well as some POS terminals. These devices can thus check whether the magnetic-stripe data matches the card, as illustrated in Figure 3.28 on the following page. The method itself is not specified by any standard, and it is used only in Germany. Thanks to the MM method, the magnetic stripes of German Eurocheque cards are protected against copying, which nowadays does not otherwise present any technical difficulties.

As the security of the MM method is primarily based on materials and technology instead of secret keys, confidentiality is essential for protecting it against attacks. Generally speaking, confidentiality is a widely used and well-proven way to achieve additional protection with physical security features of this sort.

### 3.5.14 Security features

An especially large number of visual security features were developed in the period between the large-scale use of cards without chips and the introduction of smart cards. During this period, such features were the only way to verify the authenticity of the cards. The embedded microcontrollers in the new cards and the cryptographic methods that they make possible have diminished the importance of these features. They are nevertheless still very important whenever the authenticity of a card must be verified by a person instead of a machine, since a person cannot access the chip without special equipment.

Here we can only provide a highly condensed description the essential and best-known security features used with cards. There are many other features available, such as invisible markings that can only be seen with IR or UV illumination, magnetic codes, and special printing processes using rainbow-colored inks.



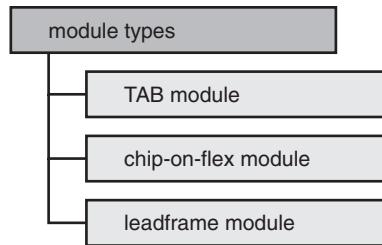
**Figure 3.28** Operating principle for verifying the authenticity of a German Eurocheque card using the MM method. The security module (MM box) protects the MM algorithm and the subsequent comparison; only a yes/no result is reported to the higher-level system

In principle, it would be possible at some time to have security features in the chips as well as on the card body. It is conceivable that security chips could be used in the same way as banknote paper is now used. Genuine bank notes cannot be printed without real banknote paper, which has specific features to show that it is genuine. In order to incorporate similar security features into chips, special chips with specifically modified hardware are necessary. A terminal can then detect the modification, which constitutes the feature of the chip, and assess the genuineness of the chips from the result.

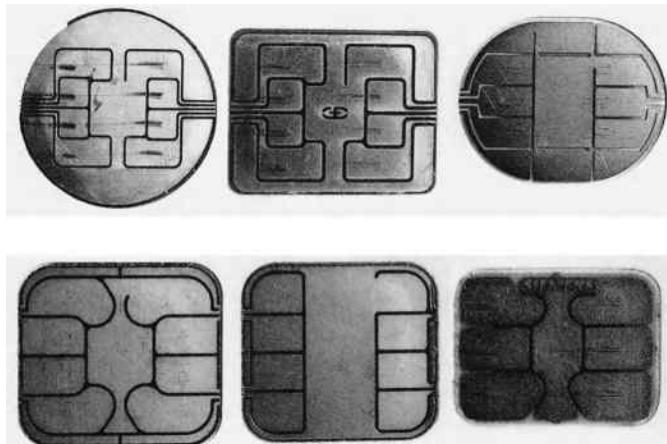
As an example of a hardware feature, suppose that computation of a fast cryptographic algorithm is implemented in supplementary hardware in a certain chip. The time required to compute a particular value could be made so short, thanks to the hardware implementation of the algorithm, that it would not be possible to perform the same computation in an equally short time using a software emulation with a different chip. A terminal could thus distinguish this chip from other chips by making a simple timing measurement. There are chips available with hardware features similar or identical to what we just described. Naturally, they are not freely available, just as banknote paper is not freely available. Of course, such hardware features are only suitable for very large-scale applications due to the high cost of developing chip-specific hardware. The fact that such chips are almost inevitably available from only one manufacturer, with no possibility of an alternate source, is also difficult for many card issuers to accept.

## 3.6 CHIP MODULES

The most important component of a smart card is naturally the chip. This very fragile component cannot simply be laminated to the surface of the card like a magnetic stripe. Instead, it



**Figure 3.29** Classification of the various types of chip modules



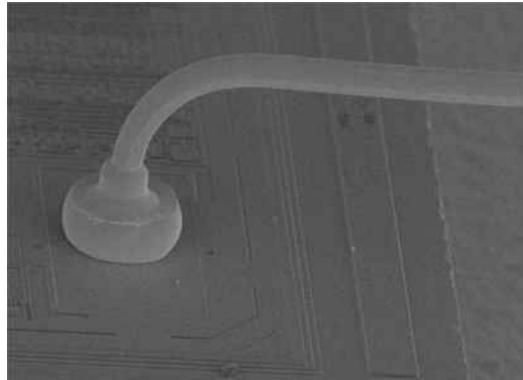
**Figure 3.30** The evolution of chip-on-flex technology illustrated by several examples, starting with one of the first eight-contact chip-on-flex modules at the upper left and proceeding to modern modules with six or eight contacts

needs a sort of enclosure to protect it from the rough everyday world of the card. This enclosure is called a chip module, or sometimes a micromodule. In addition to protection from ambient conditions, chips for contact smart cards need six or eight contacts that provide power to the chip and enable data communication with the terminal. A portion of the module's surface is used to provide these electrical contacts to the outside world. Naturally, the chip module should be as inexpensive as possible.

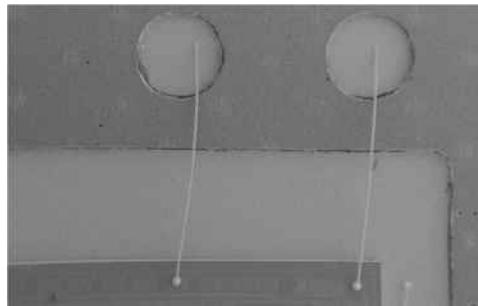
A wide variety of module designs have been devised in the course of smart card development in order to meet these two requirements – protection of the fragile semiconductor chip and provision of contact surfaces. The most important types are listed in Figure 3.29 and illustrated in Figure 3.30.

### 3.6.1 Electrical connections between the chip and the module

Electrical connections are needed between the chip inside the module and the contacts on the outside of the module. Presently, two methods are generally used for this. With the



**Figure 3.31** Photograph of the joint between a bonding wire and a bonding pad of a smart card microcontroller, magnified 1000 times (Reproduced with permission from Giesecke & Devrient)

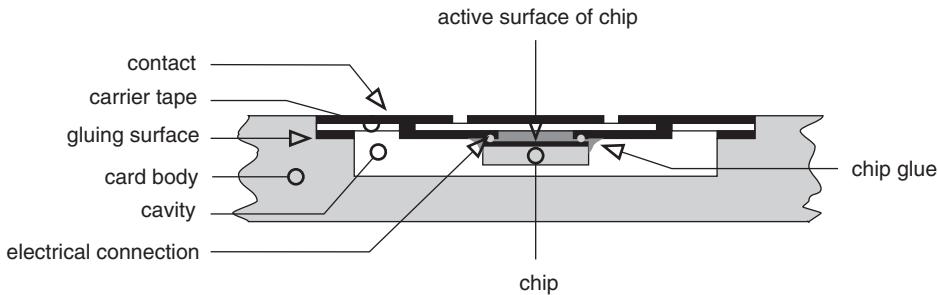


**Figure 3.32** View of the electrical connections between a smart card microcontroller (bottom) and the chip module (top), magnified 400 times (Reproduced with permission from Giesecke & Devrient)

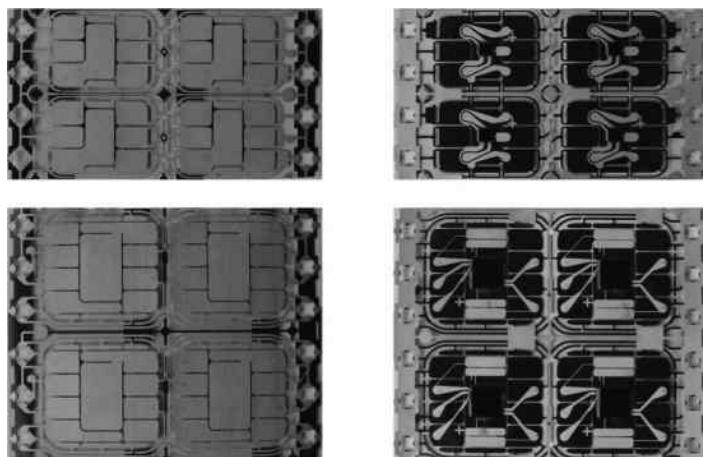
wire-bonding method (see Figures 3.31 and 3.32), an automatic bonding machine connects thin gold wires (only a few micrometers in diameter) between the chip and the backs of the contacts. The wires are electrically bonded to the chip and the module using ultrasonic welding. With this method, the contact arrangement on the top surface of the chip is always opposite that of module. This has been a standard method in the semiconductor industry for a long time, and it can be readily used for mass-producing chip modules. Each chip must be electrically connected to the module by five wires.

The die-bonding method was developed to further reduce the cost of fitting chips into modules. With this method, the electrical connections between the chip and module are not made with wires. Instead, the chip is mechanically attached to the back of the module such that each of its contacts is electrically connected to the module.

Another method is flip-chip technology (see Figures 3.33 and 3.34 on page 53), in which the chip is placed with its face against the back of the module with the electrical connections to the module provided by small solder bumps, after which the assembled module is filled with a casting resin. This type of low-cost module is usually designated FCOS (flip-chip on substrate). To ensure backward compatibility with the widely used wire-bonding technology, FCOS chips



**Figure 3.33** Cross section of a chip module made using flip-chip technology



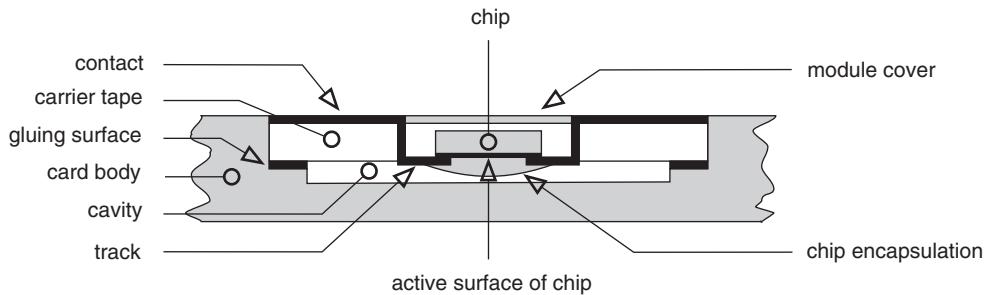
**Figure 3.34** Front and back sides of modules for flip-chip technology

often have solder bumps for flip-chip connection as well as the previously standard pads for wire bonding. Such flip-chips can be used equally well in lead-frame and chip-on-flex processes.

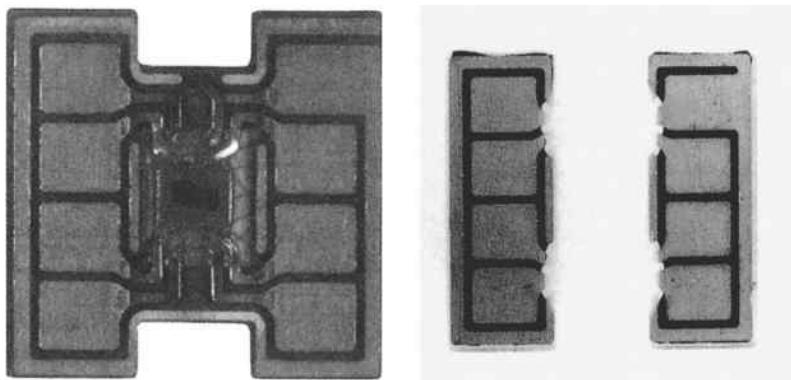
### 3.6.2 TAB modules

Although tape-automated bonding (TAB) was the standard technology for large-volume chip packaging in the early 1990s, it is rarely used now. It has become technically obsolescent and too expensive. It is described here only for the sake of completeness.

Figures 3.35 and 3.36 on the following page illustrate the use of TAB technology for chip modules in smart cards. A specific feature of this technology is that metallic bumps are first electrically attached to the pads of the chip, and the leads of the carrier film are then soldered to these bumps. The solder connections are so sturdy that no additional support is required for the chip, which hangs from its leads. The active surface of the chip is protected against ambient conditions by an encapsulation resin. The advantages of TAB technology are the mechanical strength of the connections to the chip and the low profile of the module. However, these advantages come at the price of higher cost relative to other module technologies.



**Figure 3.35** Cross section of a chip module in TAB technology



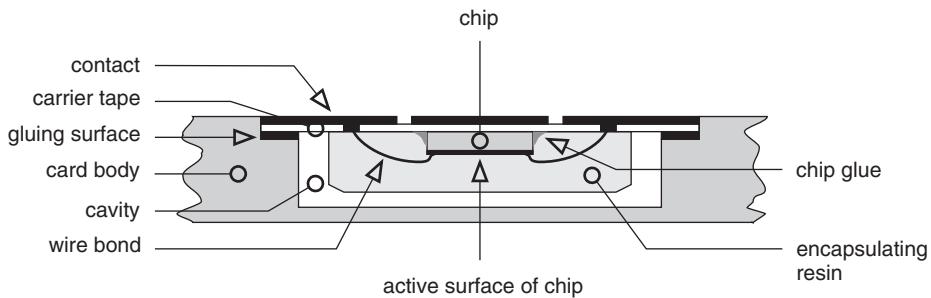
**Figure 3.36** A TAB module ready for embedding in a smart card (left), and a TAB module fitted in a smart card (right)

Fitting a TAB module into a smart card is not easy, since the module properties must be taken into account when preparing the lamination foils for the card. Suitable openings are punched in the layers before they are laminated, and then the chip module is inserted. The chip module is welded to the card body during the lamination process. This method produces a very reliable bond between the chip module and the card body. It is nearly impossible to remove the chip from the card without destroying the card.

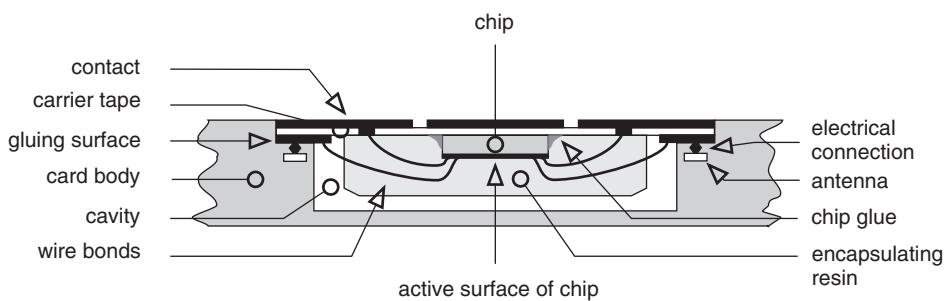
### 3.6.3 Chip-on-flex modules

Currently, chip-on-flex modules with wire-bonded chips are the most commonly used type. The structure of such a module is shown in cross section in Figure 3.37 on the next page. With this method, an opening into which the chip module can be glued is milled in the finished card body. The structure of chip-on-flex modules, the production process for these modules, and module embedment in smart cards are illustrated in Figures 3.37 to 3.42 on the following pages.

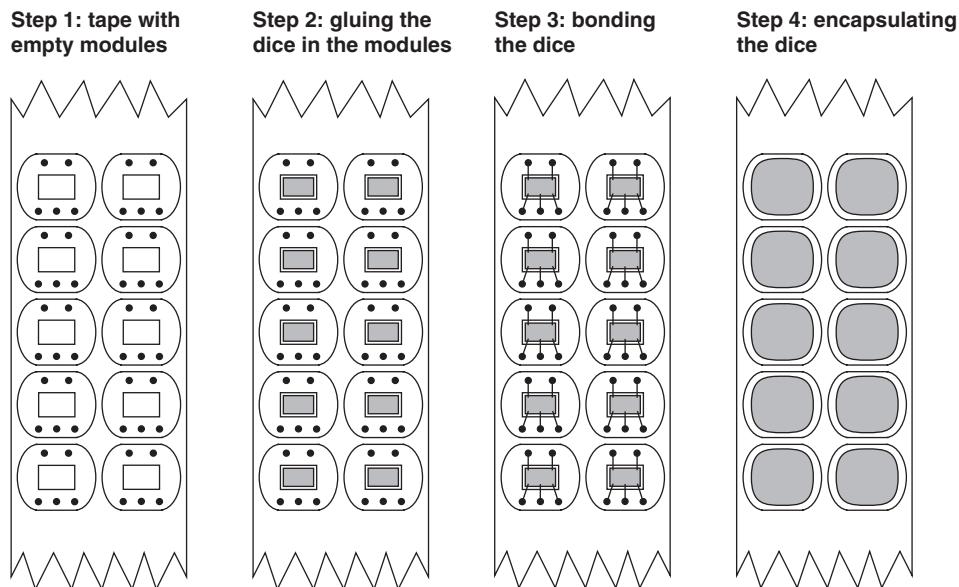
The substrate is a thin circuit board made from fiberglass-reinforced epoxy resin with a thickness of 120 µm. The contacts are formed from a 35- or 75-µm copper layer laminated onto



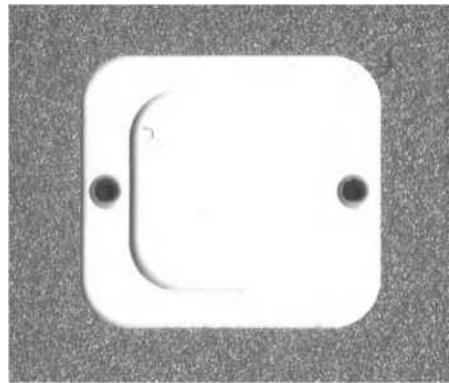
**Figure 3.37** Cross-section of a chip-on-flex module



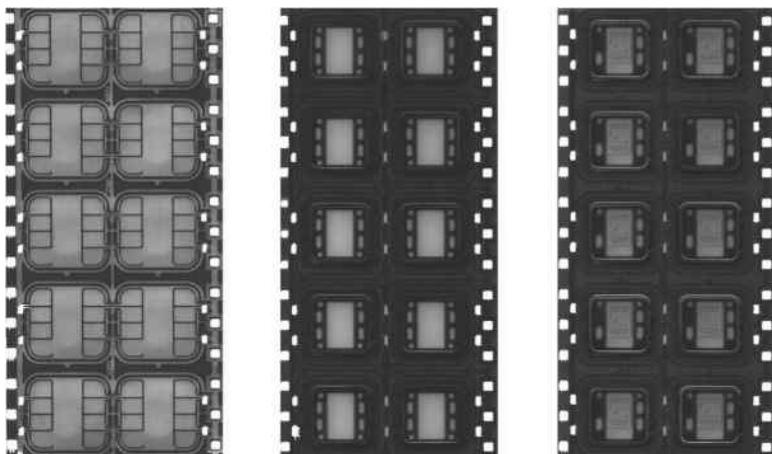
**Figure 3.38** Cross-section of a chip module in chip-on-flex technology with supplementary antenna connections for contactless communication



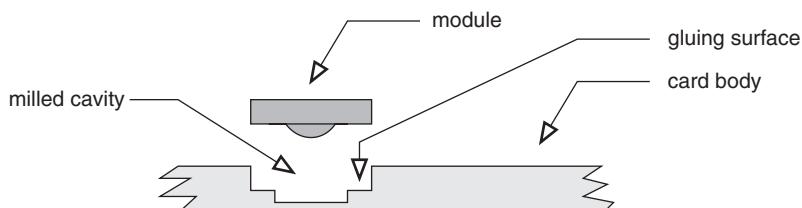
**Figure 3.39** The most important process steps in the production of chip-on-flex modules



**Figure 3.40** Rear view of a chip-on-flex module for a dual-interface card. The two contacts for the electrical connection between the chip module and the antenna embedded in the card body can be seen to the left and right of the first level of the cavity. The cross section of the corresponding module is shown in Figure 3.38 on the previous page



**Figure 3.41** Front and rear views of chip-on-flex modules on 35-mm tape. The second figure shows the module without the chip, and the figure on the right shows the module with the bonded chip. The five openings in the substrate board for the bonding wires are clearly visible at the rear



**Figure 3.42** Inserting a chip module in a milled card body

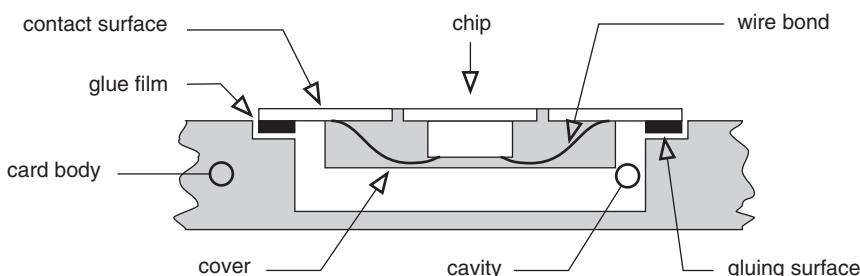
the substrate, which is subsequently gold plated using an electrolytic process. This protects the contact surfaces from processes that could degrade their electrical conductivity, such as oxidation. Holes are punched in the substrate to receive the chips and the bonding wires. The chips, which are around  $200\text{ }\mu\text{m}$  thick, are taken from the sawn wafer by a pick-and-place robot and attached to the rear of the modules by inserting them in the prepared openings in the circuit board material. Next, the chip contacts are connected to the backs of the contacts using bonding wires a few micrometers in diameter. Finally, the chip and the bonding wires are encapsulated in a blob of opaque resin to protect them against ambient conditions. The total thickness of the finished module is typically around  $600\text{ }\mu\text{m}$ .

The advantage of this method is that it is largely based on a standard process used in the semiconductor industry for packaging chips in standard packages, which makes it relatively inexpensive. This method also lends itself well to producing very complex card bodies with many functional elements, since card bodies with manufacturing defects can be rejected before the expensive chip modules are inserted. The disadvantage is that the thickness and area of the chip module are larger than with a TAB module, since not only the chip but also the bonding wires must be protected by the encapsulation. This is especially disadvantageous because the standard smart card thickness of  $0.76\text{ mm}$  does not allow much room for overly thick modules.

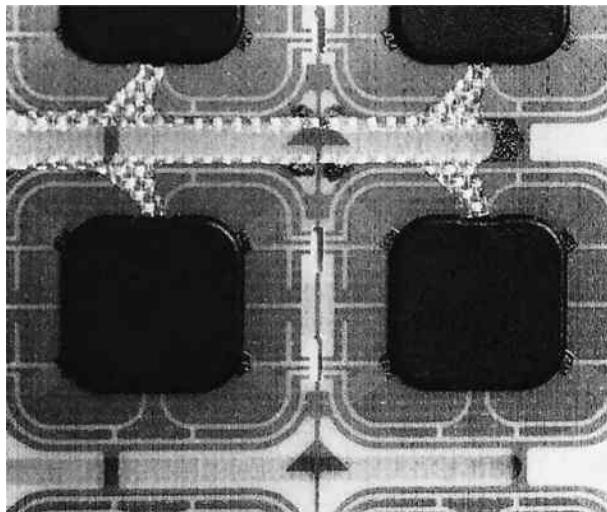
### 3.6.4 Lead-frame modules

From a technical perspective, TAB and chip-on-flex technologies are suboptimal because they provide little scope for reducing production cost. With TAB technology, producing the card body is very costly due to the module, and with chip-on-flex technology, the complexity of the module and the wire bonding process create an unfavorable production cost situation. These considerations led to the development of a type of module that has the same mechanical robustness as TAB and chip-on-flex technology but lower production costs: the lead-frame module.

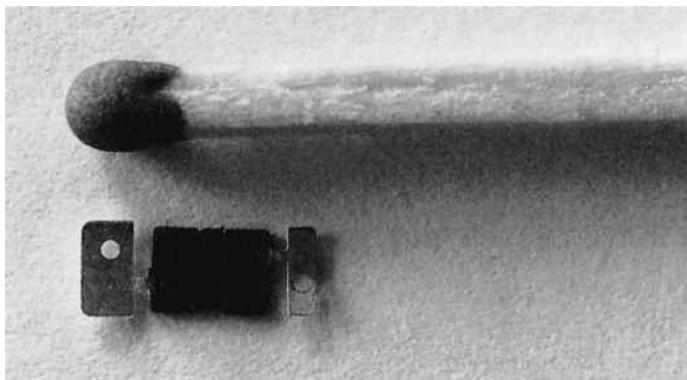
The structure of a lead-frame module is relatively simple. The contact leads, which are stamped from a sheet of gold-plated copper alloy, are held together by a plastic mold body. A chip is placed on the lead-frame module by a pick-and-place robot and wire-bonded to the backs of the leads. The chip is then covered by a protective blob of opaque epoxy resin, usually black. Figure 3.43 shows the structure of a lead-frame module, while Figures 3.44 to 3.46 on the following page show examples of these modules.



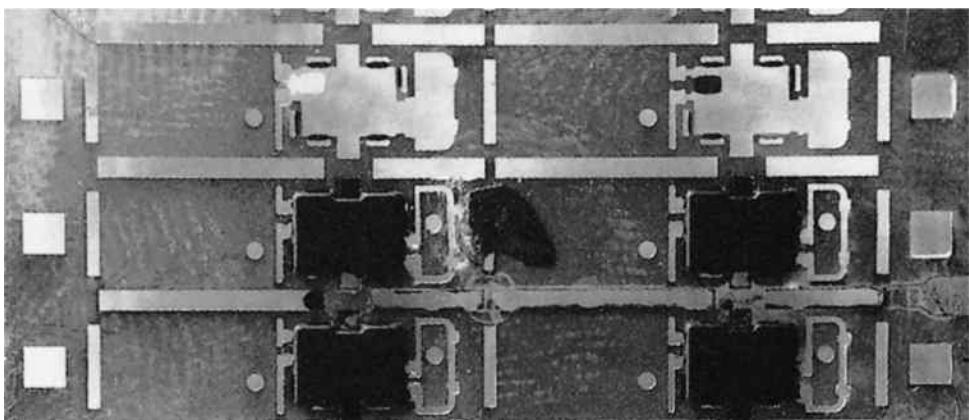
**Figure 3.43** Cross section of a chip module in lead-frame technology



**Figure 3.44** Lead-frame modules for contact smart cards, arranged in pairs on 35-mm tape



**Figure 3.45** A stamped lead-frame module with two coil contacts for a contactless smart card, shown next to a match for comparison



**Figure 3.46** Lead-frame modules for contactless smart cards, arranged in pairs on 35-mm tape. Two holes where modules have already been stamped out can be seen at the top



**Figure 3.47** A smart card with a special module for direct connection to a PC. Its four contacts have the same form as a USB plug. After the card is broken free from its carrier and placed in an adapter, it can be plugged into the USB port of a computer. This arrangement allows smart card technology to be used directly with computers, without using an intermediate terminal (Reproduced with permission from Giesecke & Devrient)

Lead-frame technology is currently one of the least expensive ways to produce chip modules, with no penalties in terms of the mechanical robustness of the modules.

### 3.6.5 Special modules

There are also various types of special modules for special applications. Chip modules can be designed to mate with a standard USB connector, including the four electrical connections. In combination with a suitable USB-compatible smart card microcontroller, they can be used to produce smart cards that can be used directly with a PC with no need for a terminal or adapter. Figure 3.47 shows an example of such a product.

# 4

## Electrical Properties

The electrical properties of smart cards depend solely on the embedded microcontroller, since it is the only component of the card with an electrical circuit. This situation will undoubtedly change in the future with the addition of other components to cards, such as displays, keypads and the like, but it will take some time before these new types of smart cards are widely used.

The application that from the very beginning has imposed many stringent requirements on the electrical properties of smart cards is mobile telecommunication in the GSM system. This system, in which an extremely large variety of terminal equipment from a large number of manufacturers must operate with a variety of card types that is probably even larger, has for a long time imposed extremely severe requirements. Due to the large number of smart cards used in the GSM system, the electrical characteristics specified for GSM cards have assumed the status of general requirements for all manufacturers of smart card microcontrollers. It can be assumed that nearly all new smart card microcontrollers comply with the general electrical parameters of the relevant GSM specifications, as otherwise they could not be marketed in the telecommunication sector.

In the early days of smart card technology, in many cases the primary concern was ensuring that the implanted microcontroller was functional, with less attention given to its general electrical properties, such as current consumption. At that time, almost all applications were closed applications using a single type of smart card together with a terminal designed to match the card. The electrical properties of the smart card were relevant only in the sense that they had to be consistent, since the terminal was designed to work with a particular type of microcontroller. However, the present situation is entirely different. With current large-scale applications in which different types of smart cards must work together with many different types of terminals, it is an unavoidable requirement that all of the cards used in the system are either electrically identical or at least have uniform electrical characteristics within clearly defined bounds.

The ISO/IEC 7816-3 standard and Amendment 1 of this standard form the general international basis for the electrical properties of smart cards. The amendment will be incorporated into the main body of the standard in the next major revision and thus disappear as a separate document. This standard specifies all of the fundamental electrical requirements for smart cards, such as voltage ranges, maximum current consumption, and the activation and deactivation sequences.

As often happens with international standards, ISO/IEC 7816-1 provides a range of options that in many cases is too large for practical use. This has allowed industry standards to become established in addition to the ISO/IEC standard, such as the EMV standard in the payment systems sector and TS 102 221 in the telecommunication sector. These two industry standards do not compete with ISO/IEC 7816-3, but instead complement it with useful restrictions arising from practical experience with smart card applications involving millions of issued cards.

The distinction between smart cards used in payment systems and smart cards used in telecommunication systems came about because certain requirements proved to have fundamentally different natures in these two application areas. For example, in the payment systems sector the current consumption and voltage range are noncritical because the terminals used in these applications are connected to the public power grid. The situation in the telecommunication sector is entirely different, since every milliwatt counts when the objective is to achieve the longest possible operating time with battery-powered mobile equipment. Consequently, the requirements for the least possible current consumption and low supply voltage are highly important in this application area.

Table 4.1 on the facing page provides a summary of the most important electrical requirements of the essential international standards and industry standards. More detailed information is provided in the following sections.

## 4.1 ELECTRICAL CONNECTIONS

Smart cards have six or eight contacts on the front, which form the electrical interface between the terminal and the microcontroller in the card. All electrical signals pass via these contacts. ISO/IEC 7816-2 specifies that two of the eight contacts (C4 and C8) are reserved for the auxiliary functions AUX1 and AUX2. These contacts can be used for purposes such as a USB interface<sup>1</sup> or for connecting an antenna for contactless data transmission. Some smart card modules have only six contacts because this yields slightly lower production costs than modules with eight contacts. However, they have the same functionality as modules with eight contacts.

The contacts are numbered sequentially from top left to bottom right. Figure 4.2 on page 64 shows the ISO designations and electrical assignments of the eight defined contacts, and Table 4.2 on page 65 describes the functions of the contacts.

Until the late 1980s, an external voltage source was necessary for programming (writing) and erasing the EEPROM because the microcontrollers used at that time did not have onboard charge pumps. Contact C6 was reserved for this purpose. However, since the early 1990s it has been standard practice to generate this voltage directly on the chip using a charge pump, so this contact no longer has a dedicated use. However, it is used now in the telecommunication sector for the Single Wire Protocol (SWP) interface for communication with an NCF component in the mobile telephone.<sup>2</sup>

## 4.2 SUPPLY VOLTAGE

Smart cards originally operated with a supply voltage of 5 V with a maximum tolerance of  $\pm 10\%$ . This voltage, which is the same as the supply voltage of conventional TTL circuits, was the standard value for all commercially available smart cards and all applications.

<sup>1</sup> See also Section 9.4, ‘USB Transmission Protocol’, on page 272

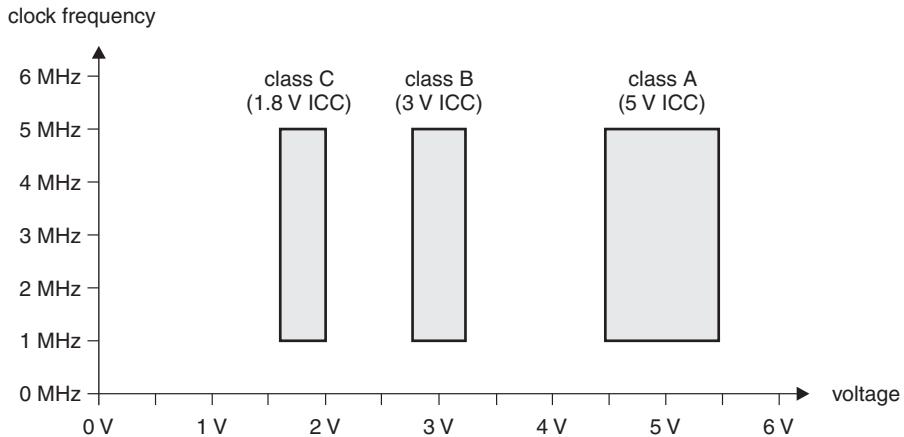
<sup>2</sup> See also Section 9.6, ‘Single-Wire Protocol’, on page 278

**Table 4.1** Overview of three electrical parameters (voltage, current and clock frequency) specified by the most important international standards for smart cards. The tolerances for the maximum current are stated in the relevant standards

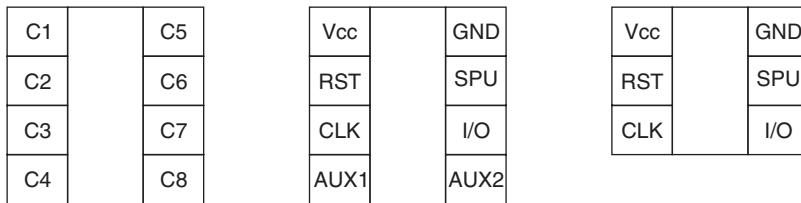
Standard and type	Voltage	Clock frequency	Maximum current
<b>ISO/IEC 7816-3</b>			
Class A	5 V ( $\pm 10\%$ ) ⇒ 4.5–5.5 V	1–5 MHz	60 mA at 5 MHz
Class B	3 V ( $\pm 10\%$ ) ⇒ 2.7–3.3 V	1–5 MHz	50 mA at 4 MHz
Class C	1.8 V ( $\pm 10\%$ ) ⇒ 1.62–1.98 V	1–5 MHz	30 mA at 4 MHz
Class A, B, and C with clock stopped			0.5 mA (clock stop)
EMV	5 V ( $\pm 10\%$ ) ⇒ 4.5–5.5 V	1–5 MHz	50 mA over the entire clock frequency range
<b>TS 102 221</b>			
Class A, from reset to application selection	5 V ( $\pm 10\%$ ) ⇒ 4.5–5.5 V	1–5 MHz	10 mA at 5 MHz (operating state) 200 µA at 1 MHz (idle state)
Class A, during an application-specific session			60 mA at 5 MHz
Class B, from reset to application selection	3 V ( $\pm 10\%$ ) ⇒ 2.7–3.3 V	1–5 MHz	7.5 mA at 5 MHz (operating state) 6 mA at 4 MHz (operating state) 200 µA at 1 MHz (idle state)
Class B, during an application-specific session			50 mA at 5 MHz
Class C, from reset to application selection	1.8 V ( $\pm 10\%$ ) ⇒ 1.62–1.98 V	1–5 MHz	5 mA at 5 MHz (operating state) 4 mA at 4 MHz (operating state) 200 µA at 1 MHz (idle state)
Class C, during an application-specific session			30 mA at 5 MHz

As with other semiconductor devices, increasingly smaller structure widths and the need for reduced current consumption have made it necessary to markedly reduce operating voltages. This tendency has been further strengthened by the mobile telecommunication sector. The market-driven demand for reducing the weight of mobile telephones required changing from 6-V batteries to 3-V batteries, and as all other components used in mobile telephones were available in 3-V technology, for a while the smart card was the only component in a mobile telephone that still needed 5 V. This meant that an extra voltage converter (with associate extra cost) was necessary to provide electrical power to the smart card.

Consequently, the international standard was revised to allow a voltage range of  $3 \text{ V} \pm 10\%$  for smart cards in addition to the 5 V range. This effectively yielded a range of 2.7 to 5.5 V. However, it quickly became apparent that this extension was not sufficient, so the revised



**Figure 4.1** Comparison of the voltage and clock frequency ranges specified in ISO/IEC 7816-3 and ISO/IEC 7816-3 Amd. 1 for the three voltage classes



**Figure 4.2** Electrical assignments and numbering of smart card contacts as specified by ISO 7816-2. See Figure 9.26 on page 273 for the contact assignments of a smart card with a USB interface and Figure 9.30 on page 278 for the contact assignments of a smart card with an SWP interface

version of ISO/IEC 7816-3 was again revised in relatively short order to allow the use of smart cards with a supply voltage of  $1.8 \text{ V} \pm 10\%$ .

This extended voltage range does not pose a problem for microprocessors or most types of memory, especially as most semiconductor devices fabricated in 0.13- $\mu\text{m}$  technology have a core voltage of 1.8 V. However, smart card microcontrollers also have integrated EEPROM or flash memory, and the charge pumps used to provide the voltage necessary for erasing EEPROM or flash memory cells posed the greatest obstacle in the development of smart cards with low operating voltages. This problem has now been overcome, and with a certain amount of additional circuitry it is possible to integrate EEPROM or flash memory and the associated charge pump in a microcontroller with an operating voltage range of 1.62 to 5.5 V.

The ISO/IEC 7816-3 standard specifies three operating voltage classes as shown in Figure 4.1: Class A for the  $5 \text{ V} \pm 10\%$  range, Class B for the  $3 \text{ V} \pm 10\%$  range, and Class C for the  $1.8 \text{ V} \pm 10\%$  range. All three voltage classes can be used individually or in any desired combination. For instance, if a smart card meets the requirements of both Class A and Class B, it can operate at 5 V as well as 3 V. However, with this combination it must be borne in mind that the interval between 3.3 and 4.5 V is outside the specified ranges, so the smart card is not guaranteed to operate properly in this interval. Nevertheless, most smart cards

**Table 4.2** Contact designations and functions as specified in ISO/IEC 7816 Parts 2, 3, 10, and 12

Contact	Designation	Function
C1	Vcc	Supply voltage
C2	RST	Reset
C3	CLK	Clock
C4	AUX1	Auxiliary 1; D+ for USB
C5	GND	Ground
C6	SPU	Standard or proprietary use; SWP in the telecommunication sector (formerly programming voltage Vpp)
C7	I/O	Input/output for serial communication
C8	AUX2	Auxiliary 2; D- for USB

can operate without any problems between the overall lower and upper limits of the specified voltage ranges.

The ISO/IEC 7816-3 standard imposes yet another very important requirement, which is that under no circumstances may the microcontroller of a smart card be damaged if the applied supply voltage is not supported by the microcontroller. This requirement is essential in order to ensure the downward compatibility of new smart cards with older types of terminals. The aim is to eliminate the possibility that using a 3-V card in a 5-V terminal, for example, could destroy the IC in the card. The three voltage ranges defined by ISO/IEC 7816-3 are not yet used for smart cards in the payment systems sector. In this sector, the original  $5\text{ V} \pm 10\%$  operating voltage still prevails because fixed terminals can easily supply this voltage.

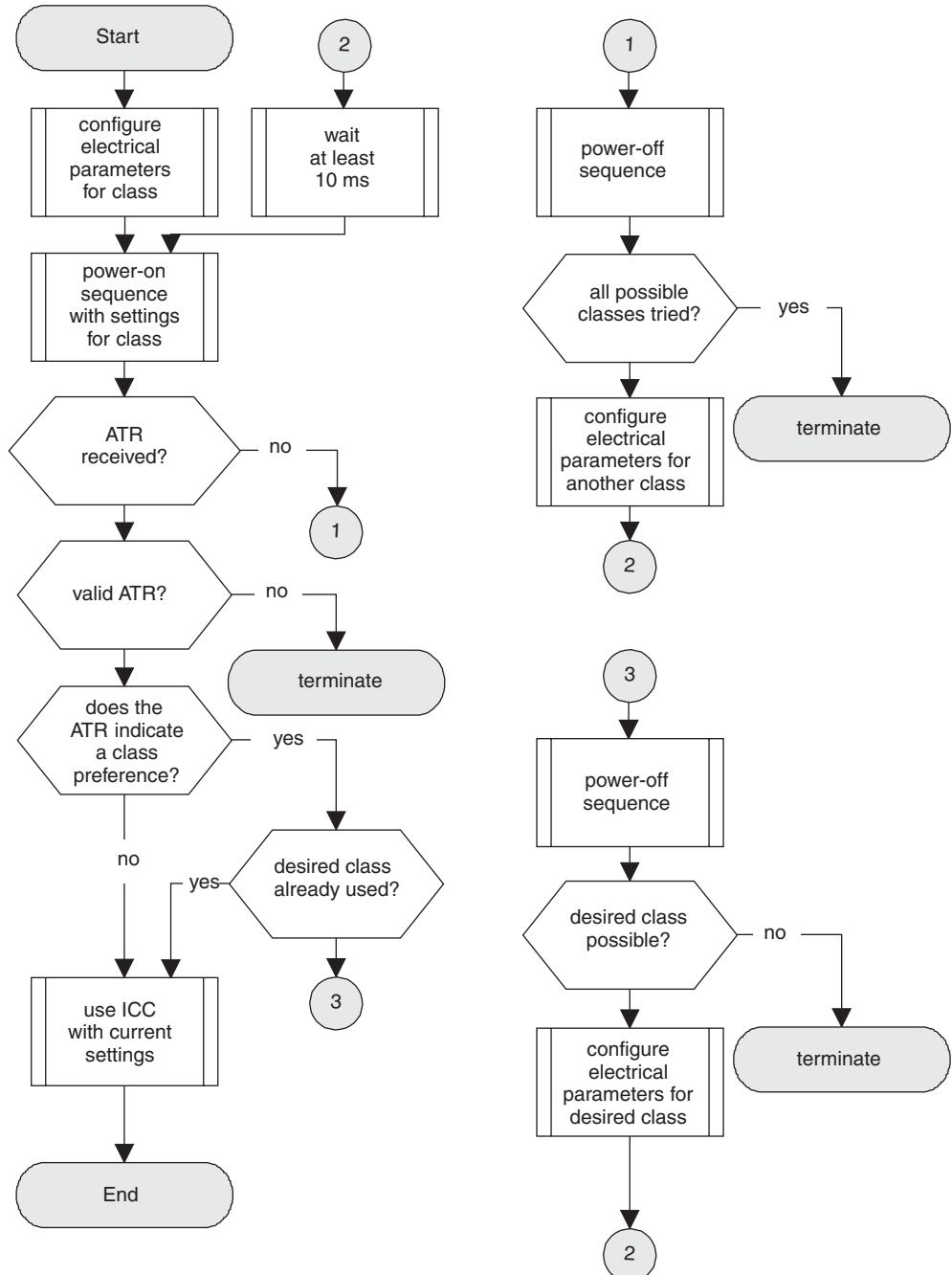
The situation is entirely different with smart cards used in the telecommunication sector. In this sector, smart cards that can only operate at 5 V have disappeared entirely. The standard supply voltage in GSM systems has been 3.3 V since the late 1990s. In UMTS mobile telecommunication networks, it is already clear that there is no longer any mobile equipment that supports 5 V. The standard voltage is 3 V, although some mobile equipment operates exclusively with a 1.8 V supply voltage.

The ISO/IEC 7816-3 standard stipulates a specific procedure for selecting the supply voltage, which essentially amounts to trying each voltage class in turn. As soon as the terminal can receive an ATR, it analyzes the ATR to see whether the smart card prefers a particular class. If it does, the terminal initiates a new activation sequence using the desired class. If the ATR does not include any information about the voltage range, the smart card is operated at the voltage with which the first valid ATR was received. The selection procedure is shown in the form of a flow chart in Figure 4.3 on the following page.

In the future, yet another voltage class (Class D) may be defined as standard, which will probably specify an operating voltage of 1.2 V. However, this additional voltage class is not urgently needed, so several years may elapse before it is included in the standard.

## 4.3 SUPPLY CURRENT

The microcontroller in the smart card obtains its supply voltage, and thus its supply current, via contact C1. This current must be held below a certain value to allow the hardware of the power source in the terminal to be designed accordingly. The first version of the



**Figure 4.3** The sequence of actions performed by a terminal to select a supply voltage according to the classes specified in ISO/IEC 7816-3 and ISO/IEC 7816-3 Amd. 1. This process usually begins with the lowest voltage class

ISO/IEC 7816-3 standard, published in 1989, specified a maximum current of 200 mA with a 5-V supply voltage and 5-MHz clock frequency, but even at that time this was too high. Since then, the value has been reduced significantly and made dependent on the voltage class.

In any case, it is important to know that the current consumption of a microcontroller is directly proportional to both the applied supply voltage and the clock frequency. It is also somewhat dependent on the temperature of the microcontroller.

The present version of ISO/IEC 7816-3 specifies a maximum current of 60 mA for voltage class A (5 V) at the maximum clock frequency of 5 MHz and maximum ambient temperature of 50 °C.

With regard to smart cards used in payment systems, the EMV specification reduces the maximum current to 50 mA in place of 60 mA as specified in ISO/IEC 7816-3, but there are no other significant differences. In the telecommunication sector, current consumption has been a critical issue since the very beginning. Consequently, in this sector there is a complicated formula that specifies the maximum current as a function of the clock frequency and the operating state of the smart card. The maximum current values specified for the different voltage classes are listed in Table 4.1 on page 63.

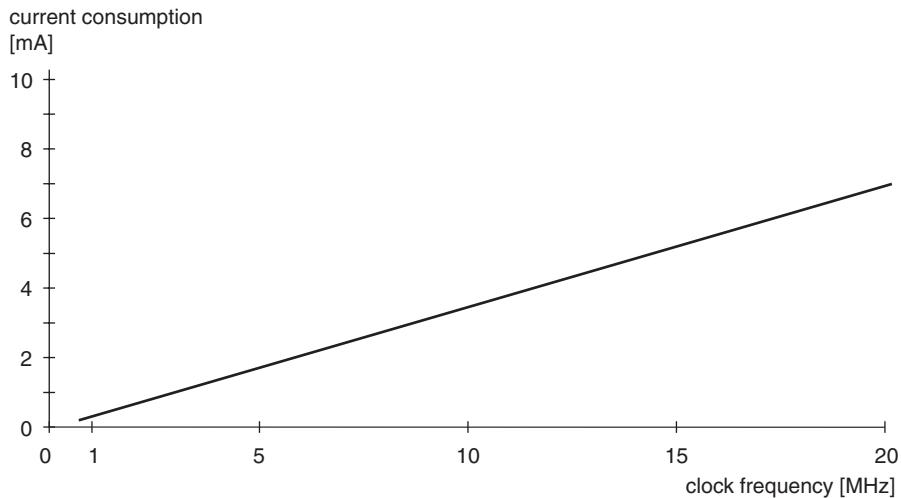
A technically interesting innovation has been introduced for smart cards that conform to the USIM specification. Here two different operational states are specified with respect to current consumption. The first state encompasses the time from the reset until the application is selected, while the second state comprises the subsequent application-specific session. The maximum allowable current consumption in the first state is significantly lower than in the second state. Furthermore, the mobile equipment can use an application-specific data object to determine how much current the selected application needs. This arises from the fact that the current consumption of a smart card is considerably higher when its numeric coprocessor or an internal frequency multiplier is enabled in order to boost its performance. In theory, this mechanism could be used to allow the mobile equipment to use only the smart card applications whose current consumption it can adequately support. Unfortunately, the relevant USIM specification does not include any procedure (similar to the PPS) that would allow the mobile equipment and the smart card to negotiate the maximum supply current. With the current version of the specification, the only option available to the mobile equipment if its smart card draws too much current is to deactivate the smart card.

Modern smart card microcontrollers have a current consumption of around 350 µA per megahertz of clock frequency, as shown in Figure 4.4 on the following page. Using this value, we can write the following formula for the current consumption of a microcontroller as a function of the clock frequency supplied to the chip or generated inside the chip:

$$I = \frac{f}{2.857} \frac{\text{mA}}{\text{MHz}}$$

The resulting value is useful for making initial estimates, but it must be remembered that the current consumption depends not only on the clock frequency, but also on the supply voltage, the temperature, and of course the chip type.

With a supply voltage of 5 V and an assumed current consumption of 5 mA, the power consumption of a smart card is 25 mW. This value is so low that there is no need to be concerned about chip overheating during operation, even though this power is dissipated over an area of approximately 10 mm<sup>2</sup>.



**Figure 4.4** Microcontroller current consumption versus clock frequency in normal operating mode (not sleep mode). The current consumption in sleep mode with the clock enabled is also linearly dependent on the clock frequency

All smart card microcontrollers have one or more reduced-current operating modes. The operating principle of these modes is based on de-energizing all components on the chip that are not currently being used. In principle, the only components that must remain energized are the interrupt logic of the I/O interface, the processor registers, and the RAM; the rest of the microcontroller can be de-energized. The RAM must remain energized during an active session in order to retain the current states. In practice, the processor often remains energized as well, but the ROM and EEPROM are de-energized. When the microcontroller is in this idle state or sleep mode, its current consumption drops dramatically because most of the chip's components are de-energized. In addition to the sleep mode, many smart card microcontrollers support another mode in which the applied clock can also be disabled, which is called clock stop mode. The main objective of this mode is to allow the clock generator in the terminal to be de-energized, which makes it especially attractive for battery-operated terminals in the mobile telecommunication sector.

According to ISO/IEC 7816-3, the maximum allowable current in sleep mode with the clock stopped is  $500 \mu\text{A}$  in all three classes. Even this relatively low value is too high for the mobile telecommunication sector. Accordingly, TS 51.011 specifies an upper limit of  $200 \mu\text{A}$  for 5-V smart cards at a clock frequency of 1 MHz.

There is another important consideration with regard to the supply current, which causes severe headaches for terminal manufacturers who ignore it. All microcontrollers in common use are based on CMOS technology. In these devices, large short-circuit currents may occur during transistor switching processes. These currents produce spikes with widths in the nanosecond range that are many times greater than the nominal operating current. These spikes can also occur when the charge pump for the EEPROM or flash memory is switched on. If the terminal cannot supply enough current during these brief intervals, the supply voltage will drop below the specified minimum value. This can cause a write error in the EEPROM or flash memory, and it may trigger the undervoltage detector in the chip.

For this reason, information regarding these spikes can now be found in practically every relevant standard and specification. For example, ISO/IEC 7816-3 requires power supplies for Class A (5 V) cards to be able to handle spikes with a maximum duration of 400 ns and a maximum amplitude of 100 mA. If this spike is assumed to be triangular, the equivalent charge that must be supplied is 20 nA s. A simple remedy to this problem is to connect a 100 nF ceramic capacitor between the supply voltage line and the ground line very close to the contacts for the smart card.

## 4.4 CLOCK SUPPLY

Smart card microcontrollers usually do not have internal clock generators, so an externally supplied clock signal is necessary. This clock signal is also the reference signal for data transmission. According to ISO/IEC 7816-3 as well as most other standards and specifications, the duty factor of the clock must be 50 %. The usual tolerance range for the duty factor is 40 % to 60 %.

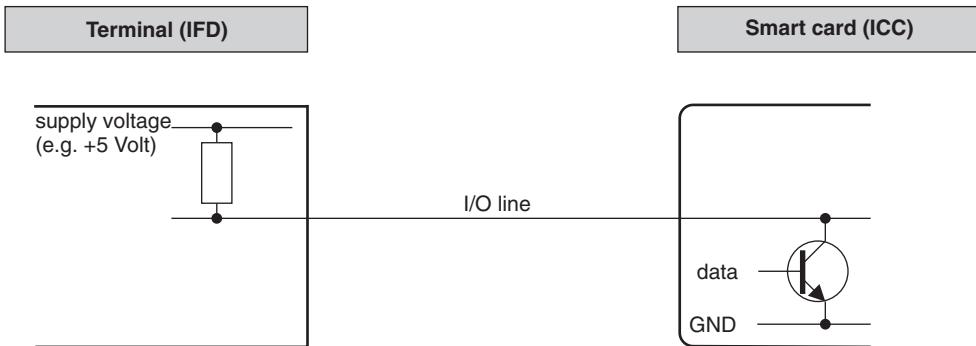
The frequency of the clock signal applied to the contact is not necessarily the same as the frequency of the internal clock signal for the processor. Some microcontrollers have a clock multiplier or divider that can optionally be interposed between the external and internal clock signals. The clock divider often has a division factor of 2, so the internal clock frequency is half the external frequency. Clock multipliers and dividers are used in part for reasons related to the chip hardware and in part because they allow oscillators already present in terminals to be used as clock sources for the chip.

Most smart card microcontrollers allow the clock to be stopped when the CPU is in sleep mode. Here stopping the clock means holding the clock line at a defined level. Depending on the semiconductor manufacturer, the preferred level may be high or low. As smart cards draw only a few microamperes of current from the clock line, stopping the clock may at first glance appear somewhat curious. However, the power savings in the terminal's clock generator may be significant, so stopping the clock can be beneficial in certain applications.

## 4.5 DATA TRANSMISSION WITH T = 0 OR T = 1

If an error occurs during data transmission, it can happen that the terminal and the card attempt to send data at the same time. This causes a data collision on the connecting I/O line. Apart from the resulting problems at the application level, this could result in currents on the I/O line that would be large enough to destroy the interface components.

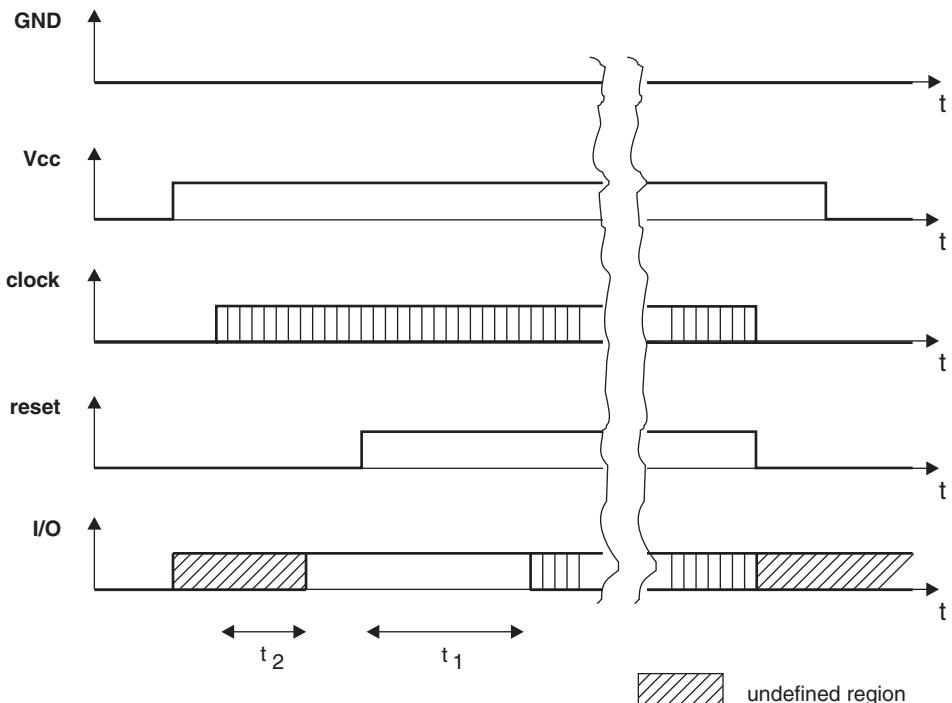
To prevent damage to the semiconductor devices in such cases, the I/O line is tied to the +5-V line in the terminal by a 20-k $\Omega$  pull-up resistor as shown in Figure 4.5 on the following page. In combination with the convention that an active 5-V level is never applied to the I/O line, this ensures that no damage will occur if both parties try to drive the I/O line at the same time. When the I/O line has to be set to the +5-V level in the communication process, the transmitting party switches its (tri-state) output to the high-impedance state and the line is raised to +5 V by the pull-up resistor. The same principle applies with a supply voltage of 3 V or 1.8 V.



**Figure 4.5** Circuitry of the I/O line between the terminal and the smart card

## 4.6 ACTIVATION AND DEACTIVATION SEQUENCES

All smart card microcontrollers are protected against electrostatic charges and out-of-range voltages on the contacts. In order to avoid undefined states, precisely specified activation and deactivation sequences are specified, and they must be strictly adhered to. This is also reflected in the relevant portion of ISO/IEC 7816-3. These sequences define the electrical



**Figure 4.6** Smart card activation and deactivation sequences according to ISO/IEC 7816-3. The times  $t_1$  and  $t_2$  are specified as  $(400/f) \leq t_1 \leq (40\,000/f)$  and  $t_2 \leq (200/f)$

aspects of activating and deactivating the card and have nothing to do with the sequence of events for mechanical contact with the card, which is actually not specified. However, it is good idea to make the connection to the ground contact first (before the other contacts) during the contacting process in order to ensure a well-defined electrical sequence, and to reverse this order during de-contacting.

As shown in Figure 4.6 on the preceding page, the ground potential must be applied first, followed by the supply voltage and then the clock signal. If the clock signal is applied before the supply voltage, the microcontroller will try to draw its entire supply current from the clock line. This can cause irreversible damage to the chip and render it useless. An incorrect deactivation sequence can have a similar effect on the microcontroller.

When the microcontroller is operating, it can be reset via the reset line. This requires the reset line to first be set to a low level and then high again. The actual reset is triggered by the rising edge. As in other computer systems, this form of reset during operation is called a warm reset. By contrast, a cold reset occurs when all of the supply lines are activated in accordance with the ISO standard.

# 5

## Smart Card Microcontrollers

From an informatics perspective, the key component of a smart card is the embedded microcontroller under the contacts. It controls, initiates, and monitors all of the card's electrical activities. Microcontrollers that have been developed specifically for this purpose are not simply externally addressable memory devices, but instead complete computers. This means that they incorporate a processor, memory, and an interface to the outside world. These components and the arrangement of the functional units on a silicon chip are shown in Figure 5.1 on the following page.

The major functional components of a typical smart card microcontroller are the processor (CPU), the address and data buses, and the various types of memory (RAM, ROM, and EEPROM or flash). The chip also has an interface unit that provides serial communication with the outside world. In the simplest case, the serial interface is only an address that is connected to the I/O contact and can be accessed by the CPU. With this minimal configuration and a suitably adapted operating system, it is certainly possible to implement a simple smart card application for use in payment or telecommunication systems.

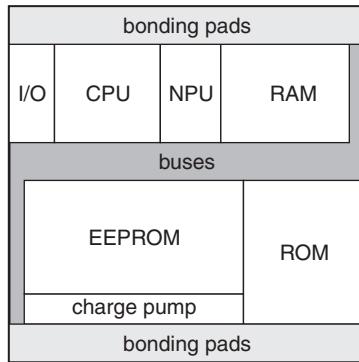
The other end of the scale in terms of the hardware configuration of a smart card microcontroller is illustrated in Figure 5.2 on the next page, which shows all of the functional units commonly present in modern microcontrollers. This also includes interfaces for other transmission methods and protocols, as well as special on-chip calculation units that are used as numerical coprocessors for cryptographic algorithms.

Figure 5.3 on page 75 and Figure 5.4 on page 76 show examples of smart card microcontroller chips. The semiconductor technology currently used for the fabrication of smart card microcontrollers lies in the range of 0.18 µm to 90 nm. Taking into account the different types of memory, this certainly lies in the realm of the smallest structure widths that can be achieved with current technology.

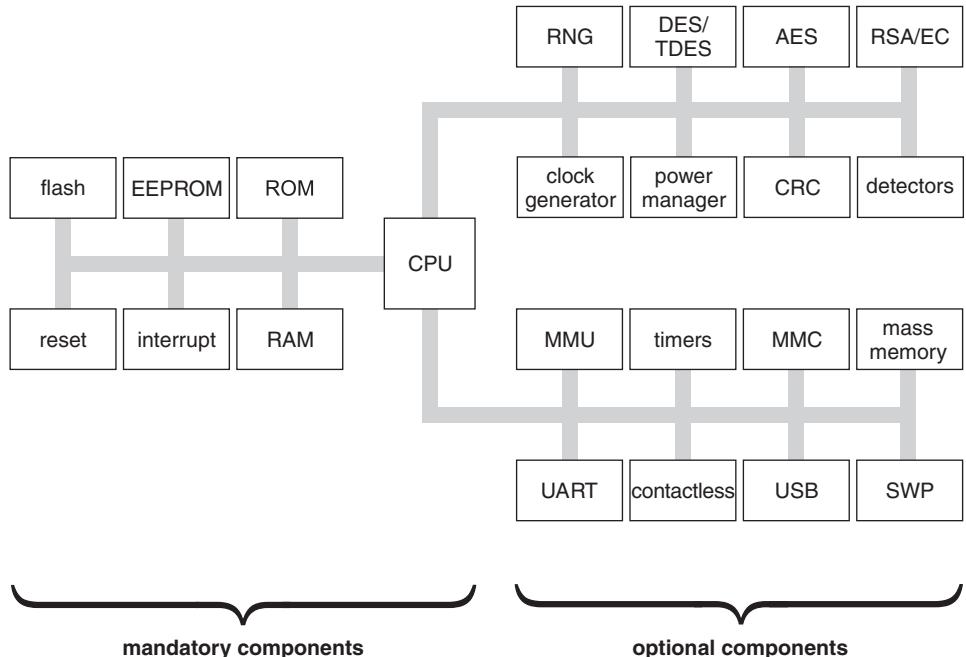
Microcontrollers used in smart cards are not standard, widely available components. Instead, they have been developed specifically for this purpose, and they are not used in other applications. There are several important reasons for this, which are described below.

### Manufacturing cost

Along with the structure width, the area of the microcontroller chip on the silicon wafer is a decisive manufacturing cost factor. Consequently, efforts are made to minimize the chip area.



**Figure 5.1** A possible arrangement of the essential functional components of a simple smart card microcontroller on the silicon die

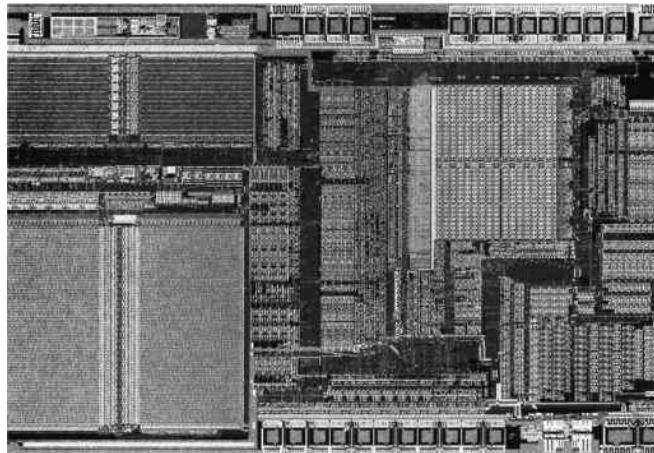


**Figure 5.2** The usual functional units of a high-performance smart card microcontroller

In addition, many standard commercial components include functions that are not necessary in smart cards. As these functions occupy additional space on the silicon, they are omitted in smart card microcontrollers. Although these area reduction measures decrease the manufacturing cost per chip only slightly, the total savings with large production volumes are significant and are large enough to justify the cost of changing the chip design.

### Functionality

Due to the need to integrate all of functional units of a computer on a single silicon chip, only a limited number of semiconductor devices are suitable for this application. Given the



**Figure 5.3** A PC 83 C 852 smart card microcontroller with the following functional units (from top left to bottom right): ROM, EEPROM, processor with coprocessor, and RAM. This chip has an area of  $22.3 \text{ mm}^2$  and contains 183 000 transistors. Although this chip is no longer produced, it clearly shows the arrangement of the functional units on the die (Reproduced with permission from Philips/NXP)

requirements of minimum chip area, a supply voltage range of 1.8 to 5 V and various special components on the chip, all standard components are essentially ruled out. In addition, the chip must include memory (EEPROM or flash EEPROM) that can be written and erased but does not require a constant source of power for data retention.

### Security

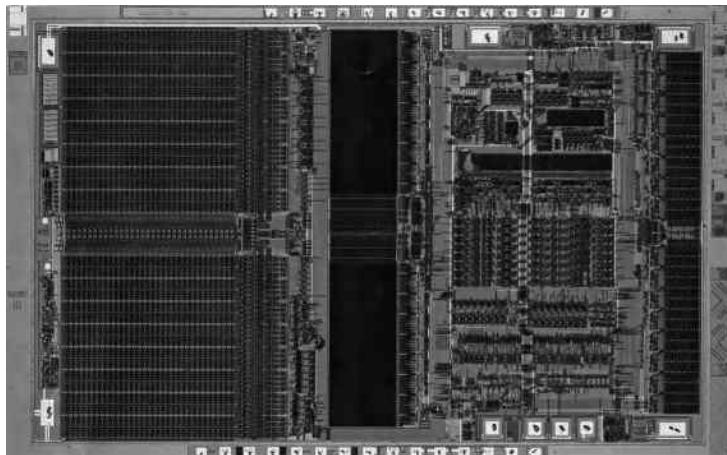
Smart cards are used primarily in security-related areas and thus need to have passive as well as active security features on the chip, so the only option is to use chips specifically developed for this purpose.

### Chip area

Along with the chip geometry, the size of the microcontroller chip is an essential factor with regard to the fragility of the die. The larger the die, the more likely it is to break as a result of the bending and torsion stresses in the card. For example, consider a smart credit card carried in a wallet: the resulting bending stress on the card and the embedded chip is substantial. Even the finest hairline crack in the chip is sufficient to render the smart card useless. Accordingly, most card producers impose an upper limit of approximately  $25 \text{ mm}^2$  on the chip area and insist that the chip layout be as nearly as possible square, in order to minimize the risk of breakage. In practice, typical smart card microcontrollers have an area of around  $10 \text{ mm}^2$ .

### Availability

Some card producers subscribe to the security policy of using microcontrollers that are not freely available on the market. This makes it considerably more difficult to analyze the chip hardware, since potential attackers do not have access to the hardware. However, this rationale has been weakened somewhat by the general availability of freely programmable smart cards such as Java Card, and it is generally no longer defensible for standard applications.



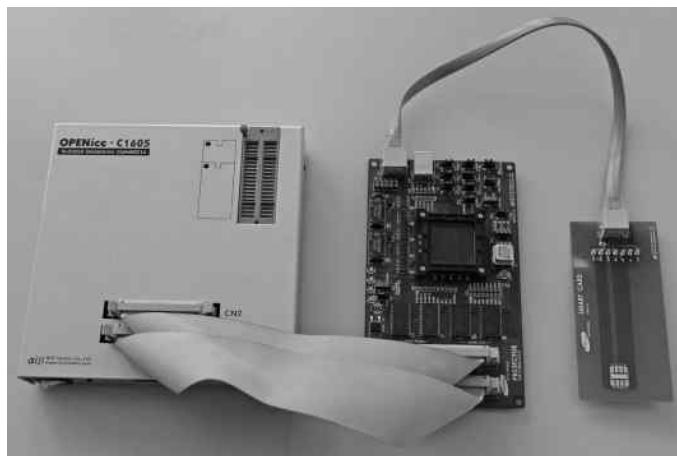
**Figure 5.4** An ST 16623 smart card microcontroller with the following functional units (from left to right): EEPROM, PROM, CPU, and RAM. Although this chip is no longer produced, it clearly shows the arrangement of the functional units on the die (Reproduced with permission from ST Microelectronics)

Restricting the selection to only a few microcontrollers and manufacturers also has the drawback that it makes the card producer highly dependent on its suppliers. If the semiconductor manufacturer experiences production problems or pursues an unattractive pricing policy, it is often not possible to change to a different component on short notice.

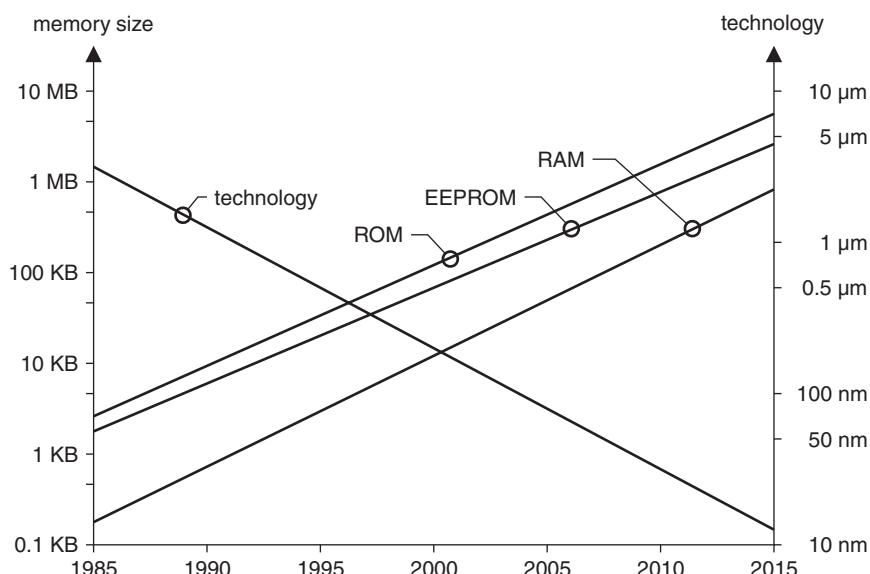
## 5.1 SEMICONDUCTOR TECHNOLOGY

The history of semiconductor technology is marked by constantly shrinking structure sizes combined with continual cost optimization, leading to constantly increasing integration density. The first published description of this phenomenon appeared in 1965 in the trade magazine *Electronics* in an article written by Gordon E. Moore, who at that time was the head of research at Fairchild Semiconductors. His assertion then, which is essentially still valid today, was that integration density doubles approximately every eighteen months. In honor of its discoverer, this relationship has been known since then as ‘Moore’s Law’, and it is one of the essential success factors and drivers of the semiconductor industry. Even cautious estimates currently assume that Moore’s Law will remain valid for the next fifteen years.

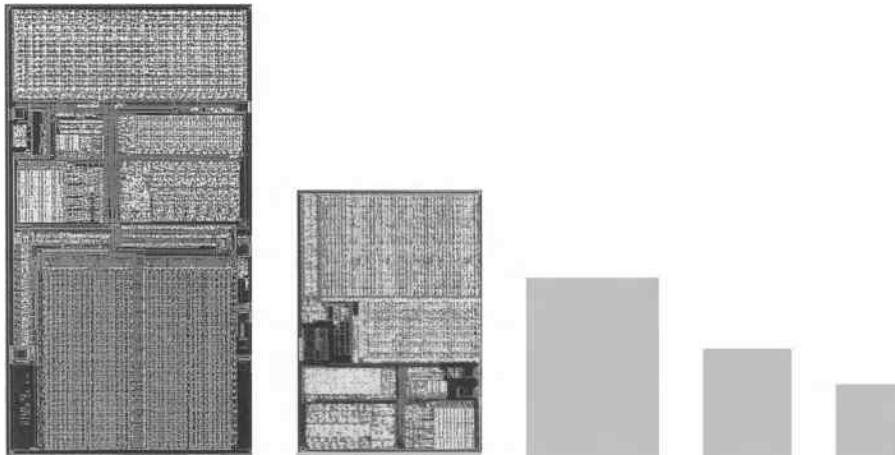
The consequences of Moore’s Law can also be seen in the evolution of smart card microcontrollers. If the course of evolution from the first smart card microcontrollers to currently available high-performance chips is plotted on a suitable chart, the relationship can be visualized quite clearly, as illustrated in Figure 5.6 on the facing page. The starting point here consists of the first two single-chip microcontrollers for large-scale use, which were first marketed in 1985 by Hitachi (now Renesas) and in 1988 by Motorola. The Hitachi HD65901 had dimensions of  $5.6 \times 5.7 \text{ mm}$  ( $31.9 \text{ mm}^2$ ) with 3 KB of ROM, 2 KB of EEPROM and 128 bytes of RAM, and it was fabricated in  $1.3\text{-}\mu\text{m}$  technology. The Motorola 68HC05SC21 (SC21 for short) had an area of  $18 \text{ mm}^2$  with 6 KB of ROM, 3 KB of EEPROM and 128 bytes of RAM, and it was fabricated in  $1.2\text{-}\mu\text{m}$  technology.



**Figure 5.5** An emulator used in the development of software for smart card microcontrollers. The unit on the left, which is the same for every chip family, is the adapter for connection to the development computer. The device in the middle is the probe board, which implements the specific characteristics of the target microcontroller in the form of a bond-out chip or FPGA emulation, and the device on the right is the paddle board, which provides the connection to the terminal



**Figure 5.6** Graphic depiction of Moore's Law as adapted to the characteristics of smart card microcontrollers. The trend lines represent the highest-performance smart card microcontrollers in each generation



**Figure 5.7** Relative sizes of functionally identical smart card microcontrollers before and after chip area reduction (shrink process). At the far left is an SLE 44C80 in 1- $\mu\text{m}$  technology with an area of 21.7  $\text{mm}^2$ . To its right is an SLE 44C80S in 0.8- $\mu\text{m}$  technology with an area of 10  $\text{mm}^2$ . The three grey rectangles indicate how large this smart card microcontroller would be if it were fabricated in 0.5- $\mu\text{m}$ , 0.35- $\mu\text{m}$ , or 0.13  $\mu\text{m}$  technology (Infineon)

Figure 5.7 shows the result of constantly shrinking structure sizes. This makes it possible to put the same functionality in an increasingly smaller area of silicon. As the manufacturing cost is almost directly proportional to the chip area, it decreases accordingly. Naturally, the cost of module packaging and electrical bonding are independent of the semiconductor manufacturing cost. These additional factors prevent the cost from decreasing without limit.

The extent to which integration density has increased can be illustrated by the following actual figures for an actual smart card microcontroller. The original technology was 0.13  $\mu\text{m}$ , and after the first shrink to 0.11- $\mu\text{m}$  technology the microcontroller occupied only 80 % of the original chip area. The next shrink step to 90-nm technology reduced the required chip area to 55 % of the original area. Incidentally, the usual rule of thumb for logic functions is 90 000 gates/ $\text{mm}^2$  with 0.13- $\mu\text{m}$  technology or 120 000 gates/ $\text{mm}^2$  with 90-nm technology. Examples of packing densities with different technologies are listed in Table 5.1 on the facing page.

To compensate for constantly declining prices as a result of Moore's Law, the industry is constantly increasing functionality and memory size. As long as the increases in integration density and functional scope remain in balance, customers receive hardware with constantly increasing performance at prices that remain roughly constant.

Naturally, increasing integration density also reduces the distances between interconnecting conductors on the chip. To ensure an adequate breakdown voltage margin, an on-chip voltage converter is used to reduce the core voltage accordingly. For example, the core voltage is typically 1.5 V with 0.13- $\mu\text{m}$  technology and drops to 1.2 V with 90-nm technology. An advantage of this approach is that the voltage converter allows the standardized smart card supply voltages (5, 3 and 1.8 V) to be supported without any difficulty.

A good overview of semiconductor technology can be found in a book by Ulrich Hilleringmann [Hilleringmann 04].

**Table 5.1** Typical packing densities of different types of memory cells as a function of technology size. For comparison, the dots in the punctuation of this book have a diameter of  $\approx 400 \mu\text{m}$  and an area of  $\approx 125\,660 \mu\text{m}^2$

Technology size	Memory type	Value
0.18- $\mu\text{m}$ technology	EEPROM	$5 \mu\text{m}^2$
0.15- $\mu\text{m}$ technology	EEPROM	$3 \mu\text{m}^2$
0.13- $\mu\text{m}$ technology	EEPROM	$1.3 \mu\text{m}^2$
0.13- $\mu\text{m}$ technology	ROM	$0.29 \mu\text{m}^2$
0.13- $\mu\text{m}$ technology	RAM	$3.5 \mu\text{m}^2$
0.13- $\mu\text{m}$ technology	typical core voltage	1.5 V
0.13- $\mu\text{m}$ technology	typical wafer size	8 inch (20.5 cm); sometimes 6 inch (15.4 cm)
0.11- $\mu\text{m}$ technology	EEPROM	$0.9 \mu\text{m}^2$
90-nm technology	NOR flash	$0.228 \mu\text{m}^2$
90-nm technology	ROM	$0.2 \mu\text{m}^2$
90-nm technology	RAM	$1.3 \mu\text{m}^2$
90-nm technology	typical core voltage	1.2 V
90-nm technology	typical wafer size	8 inch (20.5 cm); rarely 12 inch (30.7 cm)

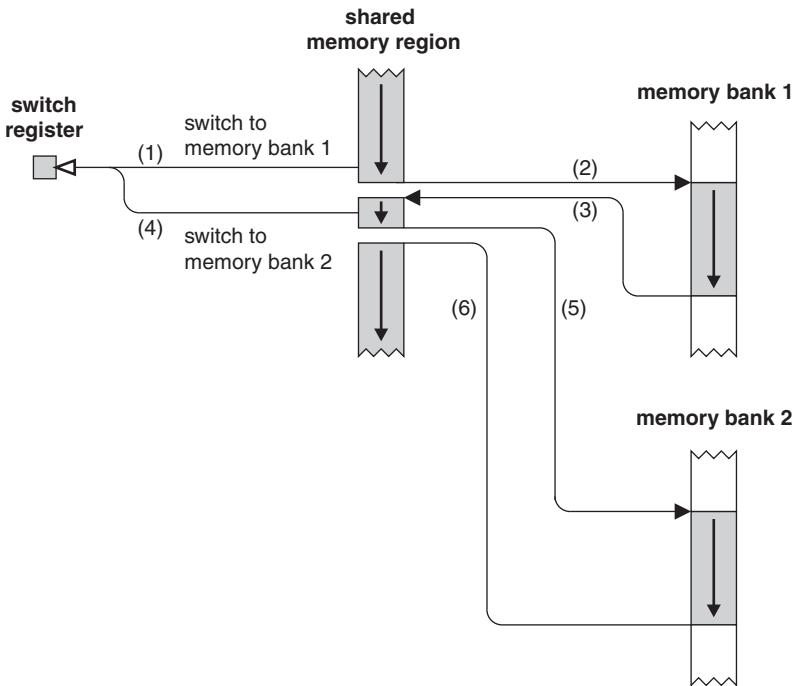
## 5.2 PROCESSOR TYPES

The processors used in smart cards are not special designs, but instead proven components that have been used in other areas for a long time. In this industry, it is not common practice to develop new processors for special application areas because this is generally too expensive. It would also yield a fully unfamiliar processor, for which no suitable function libraries or development tools would be available from operating system producers.

In addition, processors for smart cards must be extremely reliable, for which reason older types of processors that have been proven in practice are regarded as considerably less risky than the latest products of the semiconductor industry. For the same reason, only components that are one or two generations behind the current state of the technology are used in the aerospace industry, in which functional reliability is a prime concern.

The transistor was invented at the Bell Labs in 1947. Intel launched the first commercial microprocessor in 1971, with the type designation 4004. It contained 2300 transistors and operated at a clock frequency of 108 kHz with 45 machine instructions, 604 bytes of address space, and a 4-bit data bus, yielding a processing capacity of 0.06 MIPS. Since then, the development of integrated microprocessor components has made huge strides. This is clearly shown by recent products, such as the quad-core Nehalem in the Intel Penryn family with 731 million transistors on an area of  $190 \text{ mm}^2$  fabricated in 45-nm technology and operating at 3 GHz. However, leading-edge processors are not used in the smart card realm for the previously mentioned reasons. Here a transistor count of 200 000 roughly corresponds to a chip at the lower end of the performance scale.

Smart card microcontrollers at the lower end of the performance scale have a total memory capacity of around 50 to 100 KB. Under these conditions, using an 8-bit processor does not create any significant limitations. The processors usually have a CISC (complex

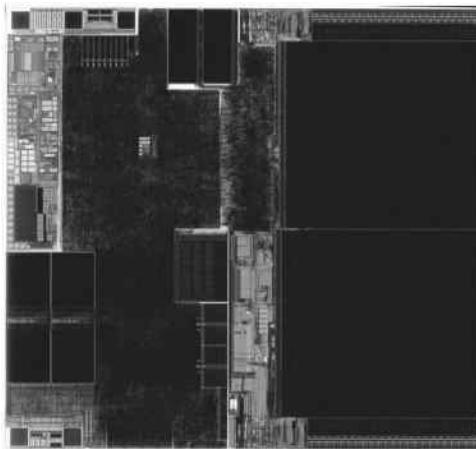


**Figure 5.8** Basic structure and program flow with a memory divided in to several memory banks. This example shows two subroutines located in different memory banks, which are called from a common memory region. The numbers in parentheses indicate the sequence of events in the process

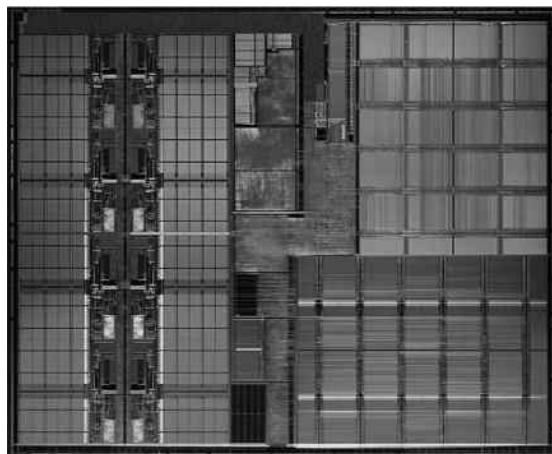
instruction set computer) architecture, which means that they need several clock cycles to execute each machine instruction and usually have a very large instruction set. The address range of 8-bit processors is generally 16 bits, which is sufficient to address up to 65 536 bytes. The instruction sets of these processors are primarily oriented toward the Intel 8051 architecture with a few enhancements, which depend on the individual semiconductor manufacturer. These enhancements usually involve additional capabilities for extended memory addressing, which is only available in rudimentary form in the original instruction set.

Processors in 8-bit families are also available with enhancements that enable them to address additional memory banks in order to surmount the 64-KB limit. Access to these memory banks is controlled by special registers that map them into a specific address space where they can be accessed by the processor, as shown in Figure 5.8. However, this nonlinear memory addressing has substantial drawbacks. For instance, the complexity of the program code is increased considerably by the relatively complicated distribution of program code over several memory banks. The probability of errors also rises with increasing complexity, and additional memory is needed for bank switching. Consequently, memory space expansion using memory banks is primarily a stopgap solution that precedes the transition to processors with larger bit widths.

The 16-bit processor sector includes derivatives of the conventional 8051 architecture, such as the 80251 architecture, as well as proprietary designs such as the H8 and AE4, the NXP XA, and the Samsung CALM. Figure 5.9 shows an example of a 16-bit microcontroller chip. Incidentally, for a long time the H8 was the only 16-bit processor for smart card



**Figure 5.9** An SLE 76CF3601P smart card microcontroller fabricated in 0.13- $\mu\text{m}$  technology with 360 KB flash, 8 KB RAM, and a coprocessor for triple DES and AES (Infineon)



**Figure 5.10** An SLE 88CX720P smart card microcontroller fabricated in 0.22- $\mu\text{m}$  technology with 240 KB ROM, 80 KB EEPROM, and 8 KB of RAM (Infineon)

microcontrollers based on a RISC-like architecture with a corresponding instruction set ('RISC' stands for 'reduced-instruction-set computer').

At the upper end of the performance scale for smart card microcontrollers, 32-bit types are also available now. The development trend is quite clearly heading in the direction of 32-bit processors, which are urgently needed in this performance class in order to manage large memories above the 64-KB limit, and especially to quench the enormous processing power appetites of modern interpreter-based smart card operating systems such as Java Card. The key selection criteria for processors include code density, power consumption, and resistance to attacks. Figure 5.10 shows an example of a 32-bit microcontroller chip.

In the 32-bit processor area, proprietary types such as the Renesas AE5 and the Infineon SLE8 are again establishing a market position, although a processor core originating from

**Table 5.2** Typical chip area distribution of a smart card microcontroller

Component type	Area
CPU, NPU	20 %
ROM	10 %
EEPROM	45 %
RAM	15 %
Other	10 %

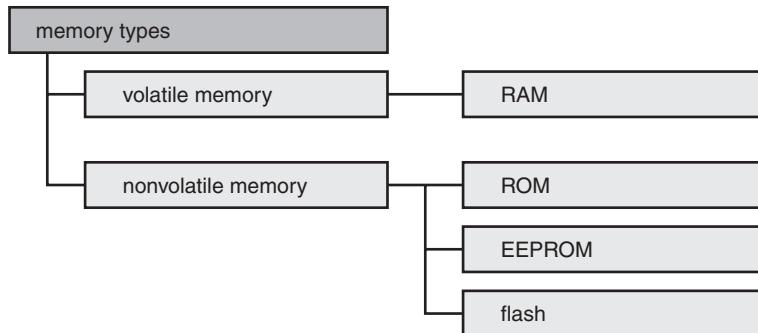
the typical industrial controller environment has also made its way into the smart card realm. This consists of ARM processors [ARM] in both variants: ARM 7 and ARM Cortex. This processor, which has a RISC architecture, has achieved considerable success outside the smart card world in recent years and is widely used.

The first steps into the 32-bit realm were initiated in 1993 with the European CASCADE (Chip Architecture for Smart Card and Portable Intelligent Devices) project. Among other things, this project had the objective of providing a high-performance processor for smart cards. The selected processor was the ARM 7M, a RISC processor that requires approximately 33 000 gates for implementation on a chip. In 0.18-µm technology, the ARM 7 processor occupies an area of 0.6 mm<sup>2</sup>. The code density in Thumb mode is one of the highest of all currently available processors. With its supervisor and user modes, the CPU also supports the separation of program code from the smart card operating system and applications. The ARM 7 processor is available in the (SC 100) version optimized for smart cards ('SC' stands for 'secure core'), which several semiconductor manufacturers use as the processor for their smart card microcontrollers. The next evolutionary step is a further optimized version with the designation ARM Cortex M3. With performance figures similar to the ARM 7, it occupies 0.21 mm<sup>2</sup> in 0.13-µm technology and has a power consumption of only 0.07 mW/MHz. In the next shrink step to 90-nm technology, the area will drop to 0.13 mm<sup>2</sup> with a power consumption of 0.047 mW/MHz.

Although 32-bit processors, with their broad bus structures and more elaborate internal components, occupy somewhat more space on the die than 8-bit processors in the same technology, they will play a growing role in future smart cards. The processing power they can provide is urgently needed for future smart card applications, so the drawbacks of higher power consumption and increased chip area can be accepted as the price of performance. Of course, it is unlikely that 8-bit processors will disappear in the foreseeable future, since they provide a solid basis for inexpensive microcontrollers at the lower end of the performance scale.

### 5.3 MEMORY TYPES

Beside the processor, the most important components of a microcontroller are various types of memory, which serve to store program code and data. Smart card microcontrollers are complete computers and accordingly have a characteristic division of memory into volatile and nonvolatile memory as shown in Figure 5.11 on the facing page. The exact division depends very strongly on the chip's ultimate application area. In any case, an effort is always



**Figure 5.11** Memory types used in smart card microcontrollers. PROM and EPROM are normally not used in modern microcontrollers. FRAM is presently used relatively rarely in smart cards

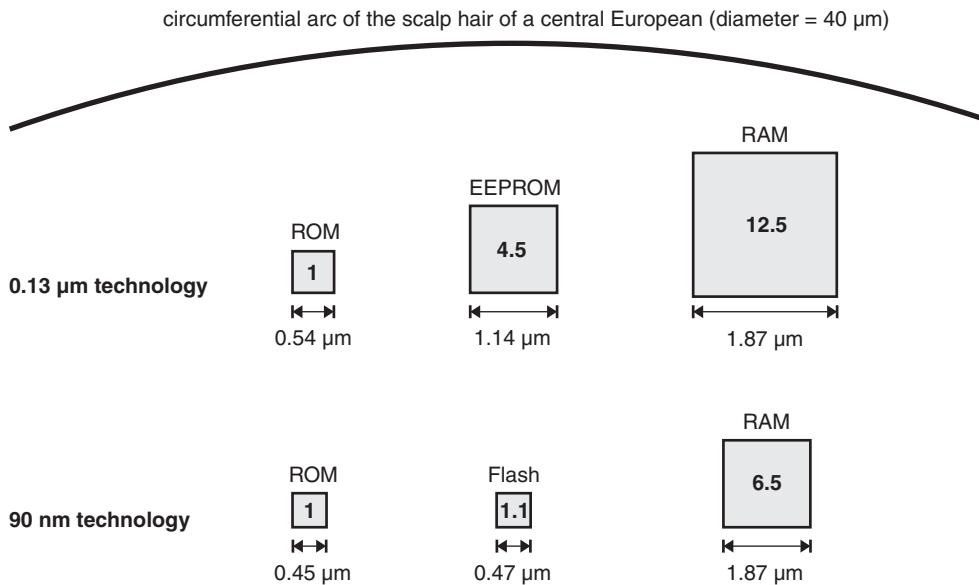
made to keep the RAM and EEPROM as small as possible because they require more space per bit. The basic read/write characteristics of the various memory types are listed in Table 5.3 on the following page.

In multiapplication smart cards, which can manage several applications concurrently, most commonly used chips have roughly twice as much ROM capacity as EEPROM capacity in order to provide enough space to hold the complex operating system. Microcontrollers whose EEPROM capacity is only slightly larger than the volume of the application data are used in smart cards that support only one application. This allows all variable application data and some parts of the operating system to be stored in EEPROM while optimizing EEPROM utilization, since EEPROM occupies a relatively large area on the die and is thus expensive.

Integrating three different types of semiconductor memory on a single silicon die is technically difficult and requires a considerable number of production steps and exposure masks. The different types of memory also have distinctly different storage densities (and thus areas) due to their different structures and operating principles. As a rule of thumb, a RAM cell occupies about four times as much space as an EEPROM cell, which in turn occupies four times as much space as a ROM cell. This is why smart card microcontrollers have so little RAM, with 4 KB of RAM already considered to be large. If you consider that 16 KB of EEPROM or 64 KB of ROM can be put in the same area, you can understand why. Figure 5.12 on the following page shows the relative sizes in graphic form.

A new type of memory technology for smart cards has become available relatively recently. It is called flash EEPROM, and it enables write access times that are much shorter than with previously available types of EEPROM. Depending on the structure width, the cell size can easily approach that of a ROM cell. This creates the very attractive option of replacing mask-programmed ROM as well as EEPROM with flash memory. A book by Ashok Sharma [Sharma 03] provides very detailed descriptions and explanations of the various types of memory at the semiconductor level.

The following numeric examples and Figure 5.12 on the following page illustrate the size relationships. A simple laser printer works at a resolution of 600 dpi (dots per inch), which means that the minimum possible dot size is  $42.6 \mu\text{m}$ . The dot at the end of this sentence has a diameter of  $400 \mu\text{m}$ . The upper limit on the recording density of high-performance hard disk drives is  $15.5 \text{ Gbit/cm}^2$  with longitudinal recording or approximately  $110 \text{ Gbit/cm}^2$  with perpendicular recording. With the latter figure and assuming that each bit occupies a square area on the disk surface, the calculated bit area is  $0.0009 \mu\text{m}^2$ . For comparison, the area



**Figure 5.12** Relative die area of single bit cells with different types of memory. The dimensions shown here are approximate and relate to 0.8-μm technology. For comparison, the diameter of the first planar transistor in 1959 was 764 μm [Buchmann 96, Stix 96], the diameter of the dots in the punctuation of this book is ≈ 400 μm, the resolution limit of the human eye is 40 μm, the size of a bacterium is 0.4–2 μm, and the size of a DNA double helix is ≈ 0.1 μm

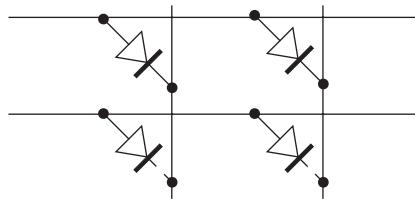
**Table 5.3** Memory types used in current smart card microcontrollers

Memory type	Number of write/erase cycles	Write time per memory cell
ROM	0	—
Flash	10 000–100 000	10 μs
EEPROM	100 000–1 000 000	2–10 ms
FRAM	$\approx 10^{10}$	$\approx 100$ ns
RAM	$\infty$	$\approx 70$ ns

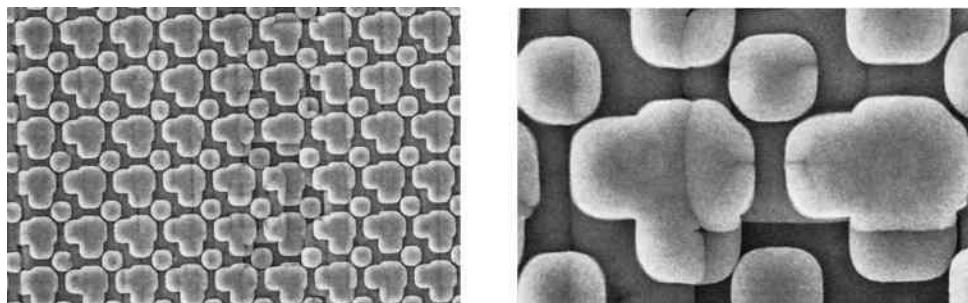
occupied by a ROM cell or flash memory cell in a smart card microcontroller fabricated in 90-nm technology is 220 times as large. The storage density of a CD-ROM is 7.3 MB/cm<sup>2</sup>, which corresponds to an edge length of 1.3 μm with a square bit cell. This is around 60 times less than the area occupied by a ROM cell in 0.8-μm technology. Digital versatile discs (DVDs) have a storage density of 50.5 MB/cm<sup>2</sup>, which means that each bit occupies 0.5 μm<sup>2</sup> and is 400 times smaller than one cell of a ROM fabricated in 0.8-μm technology.

### 5.3.1 ROM (read-only memory)

As the name implies, this type of memory can only be read and cannot be written. No supply voltage is needed to retain the data, since it is ‘hard-wired’ in the memory. Figures 5.11 and 5.14 show the functional structure of ROM cells and photographs of typical ROM cells.



**Figure 5.13** Basic functional structure of a ROM. The connection to the diode is broken in the two cells at the bottom



**Figure 5.14** A ROM cell shown at 1000 $\times$  magnification at left and 11 200 $\times$  magnification at right (Reproduced with permission from Giesecke & Devrient)

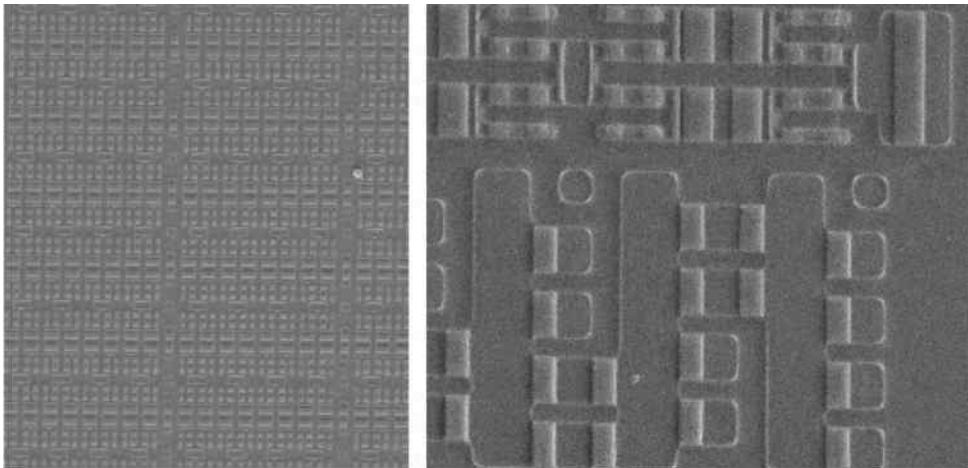
A smart card's ROM contains most of the operating system routines, as well as various test and diagnostic functions. These programs are built into the chip by its manufacturer when it is fabricated. This is done by preparing a ROM mask from the program code and using this mask to incorporate the program in the chip using lithographic processes. This data, which is the same for all chips of a production batch, can only be entered into the ROM during manufacturing.

### 5.3.2 EPROM (erasable read-only memory)

EPROM was often used in the early years of smart card technology, since at that time it was the only type of memory that could retain data without a supply voltage and could also be written (although only once per bit). However, EPROM can only be erased by UV light, so it cannot be erased in a smart card. This is also why EPROM no longer has any practical significance. The only meaningful use for EPROM would be to permanently store a chip number during semiconductor production, but this can now be done using a special type of nonerasable EEPROM.

### 5.3.3 EEPROM (electrically erasable read-only memory)

EEPROM, which is technically more complex than ROM or RAM, is used in smart cards for all data and programs that need to be modified or erased at some time. Functionally, an



**Figure 5.15** An EEPROM cell shown at 1000 $\times$  magnification at left and 4000 $\times$  magnification at right (Reproduced with permission from Giesecke & Devrient)

EEPROM corresponds to the hard disk of a PC, since it retains data in the absence of power and the data can be altered as necessary. This makes EEPROM a form of nonvolatile memory (NVM). Figure 5.5 show photographs of typical ROM cells.

In principle, an EEPROM cell is a tiny capacitor that can be charged or discharged. The charge state can be interrogated by sensing logic. A charged capacitor represents a logic 1, while a discharged capacitor represents a logic 0. In order to store one data byte, eight of these small capacitors are needed, along with suitable sensing circuitry.

The erased state of the EEPROM cell is the critical factor with regard to writing data to the cell. In most types of EEPROM, the erased state is 1. An EEPROM has the property that an individual cell can only be programmed from its erased state to its unerased state, which in this example is 0. If an EEPROM cell is already in the 0 state, an entire EEPROM page must be erased in order to restore the associated bit to the 1 state. The algorithm that is usually used for the EEPROM write routine is shown in pseudocode form in Table 5.4 on the next page.

Unlike ROM and ROM, nonvolatile memory in EEPROM and flash technology is not fully linear, but instead has an internal structure as illustrated in Figure 5.16 on page 88. This type of memory is organized in pages, which is also called the granularity of the memory. Typical memory page sizes are 8, 32, 64 and 128 bytes. For example, an EEPROM with a granularity of 32 has a page size of 32 bytes. The next higher structural level consists of sectors, which are groups of memory pages. Sectors typically have a size of 2 to 8 KB.

The data content of a sector can be erased in a single operation, which yields a considerable increase in speed compared with erasing individual memory pages, since both operations take the same time. The contents of a memory page can usually be written bytewise and erased as a group. The typical write or erase time of EEPROM memory is 2.5 ms per page.

In order to understand how an EEPROM cell works, you need to understand the underlying semiconductor technology. Figure 5.17 on page 88 shows the cross section of an EEPROM cell. The actual structure is somewhat more complicated, but this simplified diagram is a very useful aid to understanding.

**Table 5.4** Pseudocode of a routine for writing a complete EEPROM page. If several pages or only part of a page must be written, this routine should be nested in a higher-level routine. A similar procedure should be used if a write retry routine must be called in the event of an error. Here the erased state of the EEPROM is ‘FF’ and the written state is ‘00’

---

### WRITE BINARY

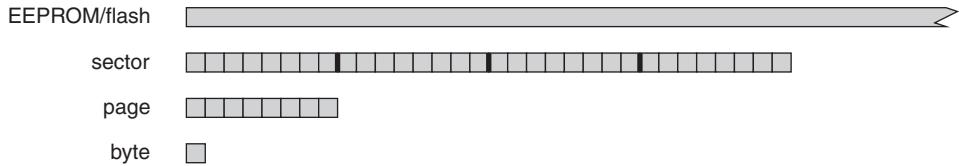
---

UpdateEEPROM:	Entry point for writing data to an EEPROM page
// NewData: data to be written	
// StoredData: stored data	
IF (NewData = StoredData) THEN. (GOTO UpdateEEPROM_Exit)	If the data already stored in the EEPROM page is the same as the new data, exit the function.
WorkData := NewData XOR StoredData	After the XOR operation, the differences between the stored data and the new data appear as set bits in the variable WorkData.
WorkData := WorkData AND NewData	The AND operation causes the variable WorkData to be nonzero if the EEPROM page must be erased before the write process.
IF (WorkData <> 0) THEN (erase EEPROM page) IF(StoredData <> 'FF') THEN. (GOTO UpdateEEPROM_Error_Exit))	If the variable WorkData is nonzero, the EEPROM page must be erased before the write operation. After this operation, verify that the EEPROM page was successfully erased.
Write NewData to the EEPROM page IF (StoredData <> NewData) THEN. (GOTO UpdateEEPROM_Error_Exit)	The EEPROM page can now be written. Afterwards, verify that the data was successfully written to the EEPROM.
UpdateEEPROM_Exit: RETURN	The function completed successfully.
UpdateEEPROM_Error_Exit: RETURN	An error occurred during the execution of the function.

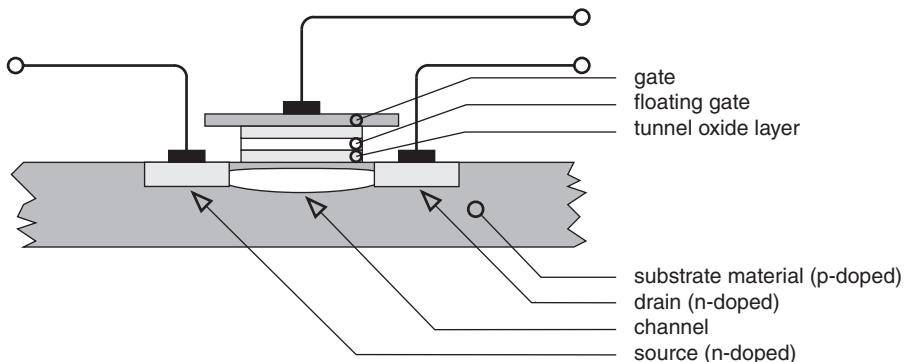
---

In its simplest form, an EEPROM cell is essentially a modified field effect transistor (MOSFET) fabricated on a silicon substrate. A MOSFET is formed by first creating a source and a drain in the substrate and then placing a control gate between them. The current flowing from the source to the drain can be controlled by applying a potential to this gate. If no potential is present on the gate, no current can flow, since there are two diode junctions ( $n-p$  and  $p-n$ ) between the source and the drain. If a positive voltage is applied to the gate, electrons are drawn toward it from the substrate, forming an electrically conducting channel between the source and the drain. The FET is then conductive, and a current can flow.

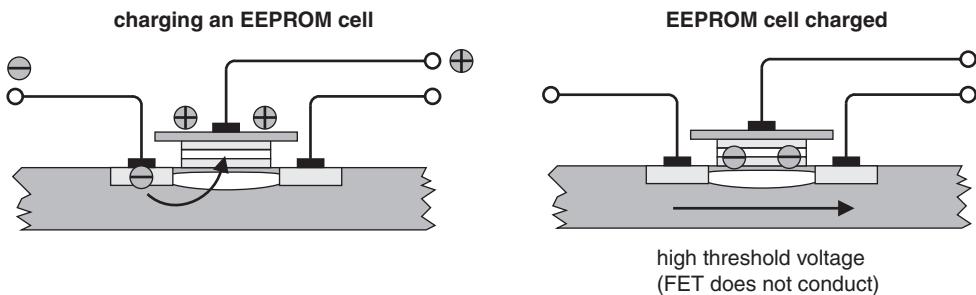
In an EEPROM cell, an additional floating gate is located between the control gate and the substrate as illustrated in Figure 5.17 on the following page. It is not connected to any external voltage source, and the distance between it and the substrate is very small (around 10 nm). Depending on the EEPROM technology, the floating gate can be charged or discharged using the tunnel effect (Fowler–Nordheim effect), which enables charge carriers to penetrate thin oxide layers. This requires a sufficiently large potential difference (around 10 MV/cm) across the oxide layer, which is called the tunnel oxide layer. The conductivity of the channel between



**Figure 5.16** Typical memory organization of EEPROM and flash nonvolatile memory. A sector can be erased as a unit but not written as a unit. A page can be erased or written as a unit. Certain types of EEPROM also allow bytewise writing, but not bytewise erasing



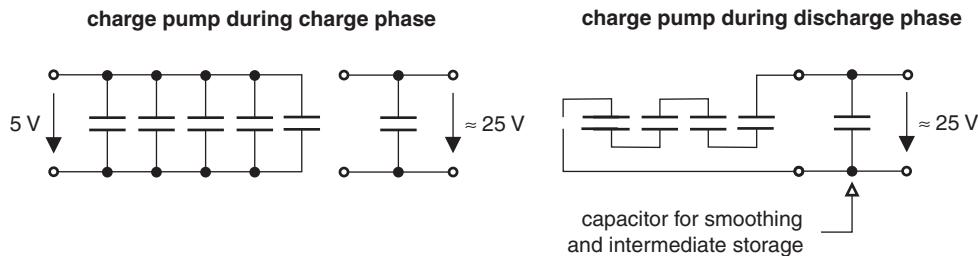
**Figure 5.17** Cross section of the semiconductor structure of an EEPROM cell



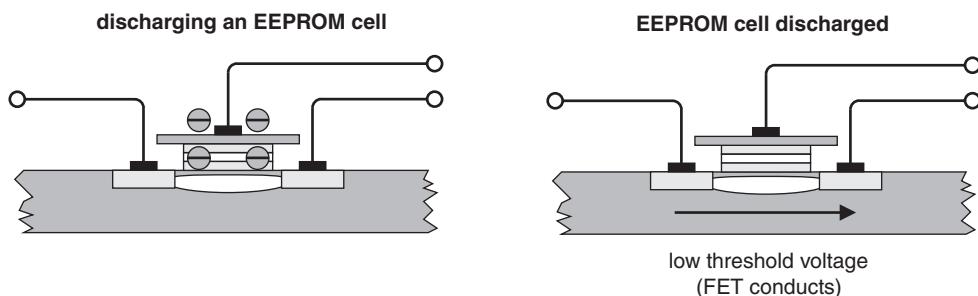
**Figure 5.18** Charging an EEPROM cell

the source to the drain is controlled by the charge on the floating gate. This means that the state of this gate can be interpreted as a logic 0 or a logic 1 according to the conductivity of the cell.

To charge the floating gate, a high positive voltage is applied to the control gate as shown in Figure 5.18. This creates a large potential difference between the substrate and the floating gate, which in turn causes electrons to tunnel through the oxide layer to the floating gate, with a corresponding current in the picoampere range. This produces a negative charge on the floating gate and results in a high threshold voltage between the source and the drain, so the field effect transistor is cut off. No current can flow between the source and the drain in this state. Storing electrons in the floating gate is thus equivalent to storing data.



**Figure 5.19** Basic circuit of a charge pump for charging (left) and discharging (right). These operations are repeated at a high frequency to yield a DC voltage with a small ripple component at the output of the charge pump



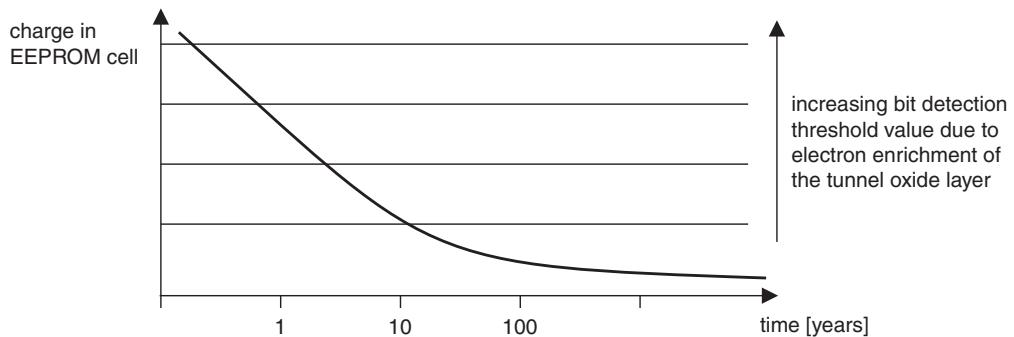
**Figure 5.20** Discharging an EEPROM cell

The voltage needed to charge the EEPROM cell is approximately 17 V on the control gate, which is reduced to approximately 12 V on the floating gate by capacitive division. However, smart card microcontrollers operate with a supply voltage of only 1.8 to 5 V, so a charge pump (essentially a cascaded voltage multiplier circuit) is needed to generate the necessary voltage. It produces an output voltage of approximately 25 V from the input voltage, which yields a voltage close to the necessary level of 17 V after stabilization. Depending on the cell structure, charging an EEPROM cell requires 2 to 10 ms per memory page (1–32 bytes). Figure 5.19 illustrates the operating principle of the charge pump.

To erase an EEPROM cell, a negative voltage is applied to the control gate. This causes electrons to leave the floating gate and return to the substrate. As a result, the EEPROM cell is discharged and the threshold voltage between the source and the drain is low, so the FET conducts. Figure 5.20 shows the basic operation of this process.

The floating gate can also be discharged by heat or energetic radiation, such as X-rays or UV light, which causes it to return to its secure state. This state is significant for the design of smart card operating systems, since security barriers can be breached by deliberately altering ambient conditions if the secure state of the EEPROM is not used to store critical data. Depending on the technical implementation of the EEPROM cell, the secure state can correspond to a logic 0 or a logic 1. This is specific to each type of smart card microcontroller, and it should be confirmed with the manufacturer if necessary.

EEPROM is a type of semiconductor memory that has a limited number of access cycles. It can be read any number of times, but it can be programmed only a limited number of times. The reason for this limitation can be found in its semiconductor structure. The life expectancy



**Figure 5.21** Displacement of the EEPROM cell discharge curve as a function of the number of executed write or erase cycles

of an EEPROM depends strongly on the nature, thickness and quality of the tunnel oxide layer between the floating gate and the substrate. As this layer must be produced very early in the fabrication process, it is exposed to high stresses in subsequent fabrication steps. These stresses may cause damage to the oxide layer, which in turn affects the useful life of the EEPROM cell.

During fabrication and every time the EEPROM cell is written, the tunnel oxide layer absorbs electrons that are not subsequently released. These trapped electrons are close to the channel between the source and the drain, and once they reach a certain number they have a stronger effect on the threshold potential than the charge on the floating gate. When this happens, the EEPROM cell has reached the end of its useful life. Although it can still be written, the charge on the floating gate has only a minimal effect on the characteristics of the channel between the source and the drain, so the threshold voltage remains the same. The number of possible write/erase cycles depends strongly on the semiconductor structure of the cell. Typical values range from 100 000 to 1 000 000 cycles over the entire range of operating temperature and supply voltage. At room temperature with an optimal supply voltage, the number of possible write/erase cycles can easily be 10 to 50 times greater than the guaranteed number.

The data retention time of an EEPROM cell decreases when it is approaching the end of its useful life. The data retention time can range from hours to minutes or even seconds. The more exhausted the EEPROM becomes, which means the more electrons that have been absorbed by the tunnel oxide layer, the shorter is the retention time.

A charged floating gate loses charge over time due to insulation losses and quantum mechanical effects, which causes the bit detection threshold to shift as shown in Figure 5.21. The time required for this to become noticeable can range from 10 to 100 years. In this regard, it is interesting to note that a charged floating gate holds approximately 30 000 electrons, depending on the cell implementation. Currently, all semiconductor manufacturers guarantee data retention for 10 years. To increase this value, the contents of EEPROM cells can be refreshed periodically by reprogramming. However, this is only worthwhile when the data must be stored for a long time.

### 5.3.4 Flash memory

Flash EEPROM memory, which is now simply called flash memory, shares the property of nonvolatility with regular EEPROM, which means that it retains its contents when the supply

voltage is removed. Flash memory is very similar to EEPROM in terms of functionality and semiconductor structure. The main differences relative to EEPROM are a smaller distance between the floating gate and the substrate and the nature of the write operation, which is based on hot electron injection instead of the tunneling effect. The erase operation utilizes the tunneling effect, just as in EEPROM.

Hot electron injection utilizes fast electrons that are produced by a high potential difference between the source and the drain. A positive charge on the control gate causes some of these electrons to penetrate the tunnel oxide layer and be stored on the floating gate. This reduces the write time to typically 30 µs, which is considerably less than the 2.5-ms write time of normal EEPROM cells. The name ‘flash’ comes from this extremely short write time. The erase time is similar to or somewhat longer than the erase time of regular EEPROM memory, and depending on the technology and the manufacturer it is typically 4 ms for a block or 80 ms for a full sector. Another advantage of flash memory is that the programming voltage is only 12 V, compared with 17 V for EEPROM.

There are two types of flash memory, which have different internal connections. NOR flash allows free read access to individual cells, which makes it suitable for storing program code because the processor needs to access memory at will, depending on the executed machine code. A drawback of NOR flash is its extensive internal interconnections, which result in a relatively large area per memory cell.

By contrast, NAND flash has a significantly higher storage density, but this comes at the price of a significant drawback, which is that NAND flash allows only blockwise access for reading. In practice, this means that it is not possible to read individual bytes as desired; instead the memory must always be read in full blocks. For this reason, NAND flash memory is not suitable for storing program code that is executed directly by the processor, since this requires random access. In order to further increase the storage density, the block size of NAND flash is significantly larger than with NOR flash; the block size of NAND flash memory is typically 2 to 64 KB. For these reasons, NAND flash is used primarily for data storage.<sup>1</sup>

The original version of flash memory, which was invented in 1984 and has been commercially available since 1988, can store only one bit per cell. This type of cell is called a single-level cell (SLC). By contrast, a multilevel cell (MLC) can store more than one bit per cell. Multilevel flash technology was invented in 1992, and commercial products have been available since 1997. Currently available products can store four bits per cell, which enables a significant increase in storage density. However, in the smart card microcontroller world the market is dominated by flash memory with single-level cells.

More and more smart card microcontrollers with NOR flash memory are coming on the market, with this memory being used primarily as a replacement for mask-programmed ROM. This can reduce the development time of a smart card project by several months because it eliminates the need to generate a ROM mask.

It is difficult to fabricate semiconductor devices with EEPROM and flash memory on the same chip, so in practice microcontrollers with flash EEPROM usually do not have EEPROM. It is replaced by flash memory with the smallest possible page size (around 32 bytes) in order to minimize the impact on the smart card operating system. The page size of flash memory used in place of ROM is generally significantly larger (such as 64 or 256 bytes), since write

<sup>1</sup> Up to now, NAND flash has had a sort of niche existence in the smart world, where it used exclusively for data storage in relatively uncommon two-chip solutions. Consequently, for the sake of simplicity the term ‘flash’ is generally used in this book to mean NOR flash; otherwise the term ‘NAND flash’ is used explicitly

operations on the routines stored in this memory are rare. During chip fabrication, a boot loader is stored in a small ROM to enable the smart card producer to subsequently load program code and data into memory.

Current NOR flash cells have a guaranteed data retention time of at least 10 years with at least 100 000 write/erase cycles and typical page sizes of 8 to 128 bytes.

There are single-chip smart card microcontrollers available with unusually large memories, often in the range of 1 to 2 MB. They are always fabricated using flash memory with a page size of up to 64 KB. This yields considerable savings in terms of the area occupied by the address and control lines, allowing memories of this size to be implemented on chips with the maximum allowable area of 25 mm<sup>2</sup>.

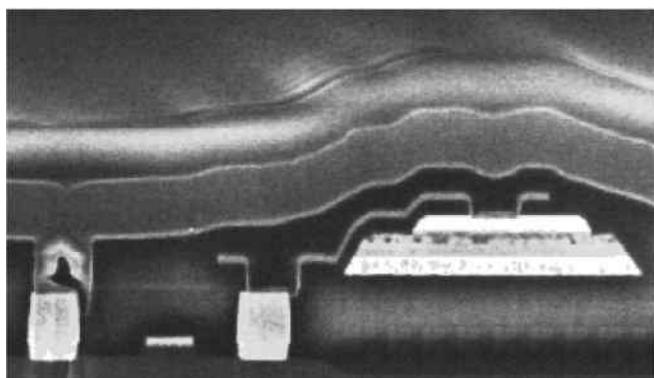
### 5.3.5 RAM (random-access memory)

In smart cards, RAM is used to hold data that is stored and/or altered during a session. The number of accesses is unlimited. RAM needs a supply voltage in order to operate. If the voltage is switched off or fails temporarily, the content of the RAM is undefined.

A RAM cell consists of several transistors, connected such that they operate as a bistable flip-flop. The state of this flip-flop represents the stored value of one bit in the RAM. The RAM used in smart cards is always static RAM (SRAM), which means that its contents do not have to be refreshed periodically. This allows the clock to be stopped, which would not be possible with dynamic RAM (DRAM) because it would cause the stored data to be lost.

### 5.3.6 FRAM (ferroelectric random-access memory)

Despite its name, FRAM not a form of volatile memory like RAM, but instead retains its contents without a supply voltage. Data storage in FRAM is based on the properties of



**Figure 5.22** Cross section of a FRAM cell in 0.5-μm technology. The two light rectangles on the left are the control lines on the chip. The trapezoidal horizontal area at the right is the actual FRAM cell, and the darker layer between the two electrodes is the ferroelectric material. The width of the FRAM cell is approximately 2 μm (Reproduced with permission from Fujitsu)

ferroelectric materials. Its cell structure is similar to that of EEPROM, but with a ferroelectric material located between the control gate and the floating gate.

FRAM is potentially well suited for use as smart card memory because it has many of the properties of an ideal data storage medium. It requires only 5 V for programming, the write time is around 100 ns, and the number of guaranteed write cycles is around  $10^{10}$ . The integration density is similar to that of normal EEPROM. However, FRAM has certain drawbacks that have limited its use up to now. For example, the read operation is destructive, which means the read data must be rewritten immediately. Another drawback is more serious. The fabrication of FRAM requires process steps and materials that are not commonly used in current semiconductor technology, for which reason very few semiconductor manufacturers include smart card microcontrollers with FRAM in their product lines.

## 5.4 SUPPLEMENTARY HARDWARE

There are some requirements specific to smart cards that cannot be satisfied using conventional microcontroller hardware and cannot be fully satisfied using software, so they must be satisfied by supplementary hardware. Consequently, manufacturers of smart card microcontrollers offer a wide range of supplementary functions in the form of on-chip hardware.

The most commonly used components for supplementary functions are described below. It is generally not necessary for all of these components to be present in a particular microcontroller. The components that are present depend strongly on the target application, among other things. For example, it would be economically unsound to include an RSA coprocessor in a microcontroller whose target application uses only symmetric cryptographic algorithms. Nevertheless, there are a few commercially available microcontrollers that include nearly all of the components described below.

Another aspect of supplementary functions for smart card microcontrollers relates to the general subject of security. Chapter 16 on page 667 provides extensive descriptions of supplementary functions implemented in hardware that are primarily intended to counter possible forms of attack. Consequently, here we describe only those components whose primary purpose is not enhancing security against attacks.

### 5.4.1 Communication with T = 0 or T = 1

Communication between a smart card and the outside world typically consists of data transmission over a bidirectional serial interface using the T = 0 or T = 1 transmission protocol, and this is usually the only communication option available. Originally, data transmission and reception over this interface were controlled exclusively by operating system software, without any hardware support. The complexity of the necessary software is not inconsiderable, and it creates additional potential sources of software errors. However, the main problem is that the speed of software-based data transmission is limited because the speed of the processor is limited. With current processors, the upper limit is represented by a divisor value (clock rate conversion factor) of around 30, which yields a data transmission rate of approximately 115 kbit/s with a 3.5-MHz clock.

If higher transmission rates are desired or required, it is necessary to use either internal clock multiplication or a universal asynchronous receiver transmitter (UART) component.

As the name suggests, this is a general-purpose component for transmitting and receiving data, independent of the processor. It is not limited by the speed of the processor, nor does it need software for communication at the byte level. Of course, the upper layers of the data transmission protocol must still be implemented in software resident in the smart card, but the lowest layer can be implemented in hardware in the UART.

Most current UARTs can operate with all standard ISO/IEC 7816-3 divisor values smaller than 372, and some of them can transmit and receive data with divisor values as small as 4. There are many different implementations of this functionality. Some UARTs can transmit and receive only single bytes, while some also support the byte retransmission function of the T = 1 protocol in case of a transmission error. With such UARTs, the processor only has to supply the necessary data to the UART on time and read it from the UART on time. The UART can signal reception of a complete byte to the processor by a flag or an interrupt. More advanced types of UARTs can easily transmit or receive several bytes in succession. The highest functionality possible with current technology is provided by UARTs that can use direct memory access (DMA) to transmit data directly from RAM or store received data in RAM without the intervention of the processor and in parallel with the other processor activities. With such UARTs, the transmission protocol logic must always be implemented in software.

It has been technically feasible to implement UARTs in smart card microcontrollers since the early days of smart cards, but until the end of the 1990s transmission and reception routines implemented in ROM software required less physical area on the silicon than a functionally equivalent UART component. As chip area is a decisive cost factor with smart card microcontrollers, for a long time almost all semiconductor manufacturers rejected the hardware approach. However, conditions have changed with increasing integration density. Almost all smart card microcontrollers now include a UART as a standard component.

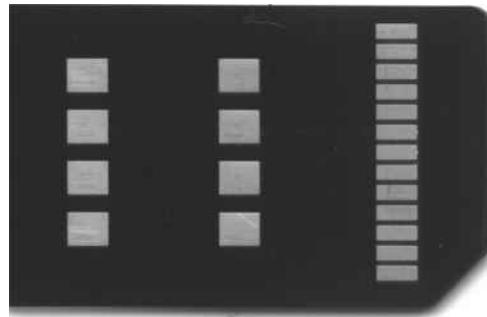
#### 5.4.2 Communication with USB

Some microcontrollers have a USB interface<sup>2</sup> component as supplementary hardware. This lets the microcontroller use the USB protocol to exchange data with a terminal. Depending on the chip configuration and application, this may occur in parallel with data transmission on another communication channel, which requires relatively strong hardware support.

The timing requirements of USB are rather rigid and cannot always be achieved satisfactorily with on-chip circuitry. In some cases it is necessary to use an external quartz crystal located in the module and connected to the chip. With this crystal, the frequency stability necessary for USB communication can be achieved.

Depending on the implementation, the USB component on the chip may support Low Speed mode (1.5 Mbit/s) and Full Speed mode (12 MB/s), with a tendency to omit support for Low Speed mode in recent products. The USB hardware usually provides a control endpoint and several data endpoints (such as four) for the on-chip software. Input and output buffers with a typical capacity of 64 bytes each are associated with the endpoints. The CRC calculations necessary for protecting data exchange are performed independently by the USB hardware. The hardware also performs the NRZI coding/decoding and bit stuffing/destuffing necessary for USB communication.

<sup>2</sup> See also Section 9.4, ‘USB Transmission Protocol’, on page 272



**Figure 5.23** A smart card module with the usual eight-contact interface and a supplementary MMC interface with eight data lines

The USB hardware enables the higher-level software layer to select a large number of configuration settings for the driver software. In many cases, the chip manufacturer provides the operating system producer with a complete driver in source code, so that the latter can modify it as necessary and use it in the operating system.

### 5.4.3 Communication with MMC

The MultiMediaCard (MMC) interface<sup>3</sup> was originally designed as an inexpensive communication interface for memory cards containing NAND flash memory. The protocol specified for this purpose is relatively static and is controlled precisely by a state machine, with the focus on data transfer. There are a few smart card microcontrollers that support the MMC protocol. In addition to the two control lines, the interface may have one, two, or more rarely eight data lines. In any case, this requires a special module with a number of additional contacts, depending on the configuration, as illustrated in Figure 5.23.

Like USB, MMC entails a large amount of additional hardware in the smart card microcontroller. In some implementations, the functionality includes not only transmitting APDUs via the MMC interface, but also direct write access to specific memory regions without any involvement of the processor. This enables the smart card to be used in exactly the same way as a memory card.

To simplify the relatively complex integration of the MMC component in the smart card operating system, chip manufacturers usually provide the source code of a sample MMD driver implementation for the microcontroller.

### 5.4.4 Communication with SWP

The Single Wire Protocol (SWP)<sup>4</sup> can be used for communication between a SIM and an NFC controller in a mobile telephone concurrently with the main communication process using the

<sup>3</sup> See also Section 9.5, ‘MMC Transmission Protocol’, on page 277

<sup>4</sup> See also Section 9.6, ‘Single-Wire Protocol’, on page 278

T = 0 protocol. Due to the potentially large market, most large chip manufacturers have at least one microcontroller in their product line that supports SWP with a dedicated hardware unit.

With SWP, data is sent digitally from the smart card using voltage modulation and received digitally using current modulation. The protocol supports full-duplex operation, which means that data can be sent and received concurrently. Each transmission frame has a CRC for protection against transmission errors. These requirements can only be fulfilled in a smart card microcontroller if they have massive hardware support.

A typical smart card microcontroller configuration has four transmit buffers and four receive buffers for SWP, each with a capacity of 32 bytes (large enough to hold a complete SWP frame). Buffer use is controlled by a state machine to ensure continuous transmission and reception capability. At this level of the communication process, the only task of the processor is to supply data to the buffers and retrieve data from the buffers, both at a sufficiently high rate. The hardware unit also calculates the necessary CRC checksums independently. Additional layers implemented in software to control the higher-level protocol mechanisms are built on top of this hardware-intensive communication layer.

### 5.4.5 Communication with I<sup>2</sup>C

Another widely used interface for communication between electronic components is the I<sup>2</sup>C bus.<sup>5</sup> This two-wire bus is also often used for communication with independent card components in a smart card. Some typical examples are driving a display in a super smart card or integrating a supplementary memory chip.

To support the I<sup>2</sup>C bus, the microcontroller has supplementary on-chip hardware, which usually acts as an I<sup>2</sup>C bus master. With the associated transmit and receive buffers and the option of different transmission rates, the functionality is comparable to that of a normal UART. In combination with the simple protocol, this makes integration of I<sup>2</sup>C-compatible communication relatively easy.

### 5.4.6 Timer

A timer in a smart card microcontroller is connected to the internal processor clock or UART clock (which counts etus) via a configurable divider. It usually has a counting range of 16 bits, or more rarely 32 bits. A timer can count the number of clock pulses from the start command to the end command without involvement of the processor. Most timers can also be operated in reloadable mode, in which they count down from a predefined value and trigger an interrupt when the count reaches zero, after which the counter is automatically reset to the initial value and continues counting. Timers are most often used by the operating system in connection with data transmission for timeout detection and the More Time command.<sup>6</sup>

Many microcontrollers also include a watchdog. In principle, a watchdog is a timer that must be reset regularly by an explicit processor instruction in order to prevent it from timing out after a configurable interval and triggering a reset. A watchdog allows the processor to be

<sup>5</sup> See also Section 9.2.2, ‘I<sup>2</sup>C bus’, on page 251

<sup>6</sup> See also Section 9.3.1, ‘The T = 0 transmission protocol’, on page 255

reset to a defined state after a definable maximum interval if it becomes trapped in an endless loop. The primary typical application for watchdogs is in the automatic controller environment, where they are highly useful. However, they are not of much use for smart cards, partly because the software is (hopefully) extremely reliable and partly because the terminal can always reset the microcontroller if it gets stuck in an endless loop. Consequently, watchdogs are generally not used in smart card microcontrollers.

#### 5.4.7 CRC (cyclic redundancy check) calculation unit

Calculating a CRC code is still a standard method for protecting data or programs by means of an error detection code. Calculating CRC codes in software is relatively slow due to the large number of bit manipulations required, but the calculation can be readily implemented in silicon in the microcontroller hardware. For this reason, many smart card microcontrollers have a hardware-based CRC calculation unit. The usual generator polynomials and initial values can be configured by the user with these units.

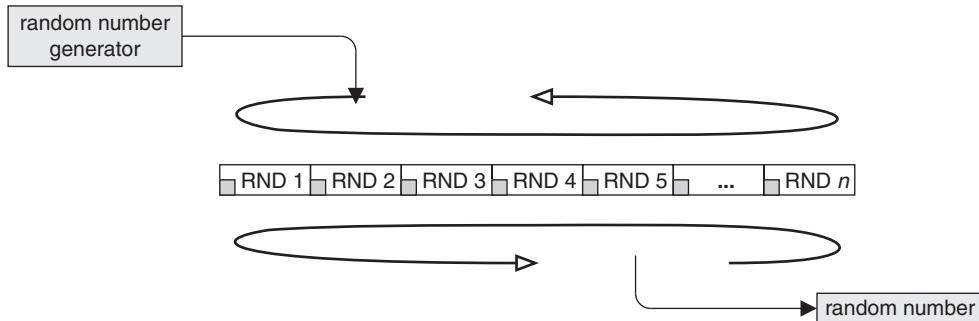
#### 5.4.8 Random number generator (RNG)

Random numbers are often needed in smart cards for generating keys and authenticating smart cards and terminals. For reasons of security, they should be genuine random numbers rather than pseudorandom numbers, which are commonly produced by typical software-based random number generators. All modern smart card microcontrollers have hardware random number generators that generate true random numbers.

The quality of these generators must not be impaired under any circumstances by external physical properties such as temperature or supply voltage. The random number generator may utilize these external parameters to generate random numbers on the chip, but this must be done in a manner that does not allow the predictability of the random numbers to be affected by purposefully manipulating one or more of these parameters.

This is very difficult to implement in silicon, so a different approach is taken. The random number generator uses various logical states of the processor, such as the clock signal or the memory contents, as the input values of a linear feedback shift register (LFSR) clocked by a signal that is also generated using several different parameters. For example, the frequency of the clock signal may be a multiple of the processor clock frequency. If the CPU reads this random number register at an arbitrary time, it obtains a relatively good random number that cannot be ascertained from outside in a deterministic manner. The quality of the true random numbers obtained in this manner can be improved by supplementary methods and algorithms. However, the essential aspect here is that the hardware-based random number generator must provide essentially good random numbers that can pass the usual tests, such as FIPS 140-2 and AIS 31.<sup>7</sup>

<sup>7</sup> See also Section 7.3.2, ‘Testing random numbers’, on page 163 for a brief discussion of the quality of random numbers



**Figure 5.24** Example of a random number generator whose outputs are continually written to a ring buffer, from which they can be requested as necessary. The grey rectangles mark random numbers that have already been read once and cannot be requested again. This sort of buffer is often used with low-speed random number generators

#### 5.4.9 Clock generation and clock multiplication

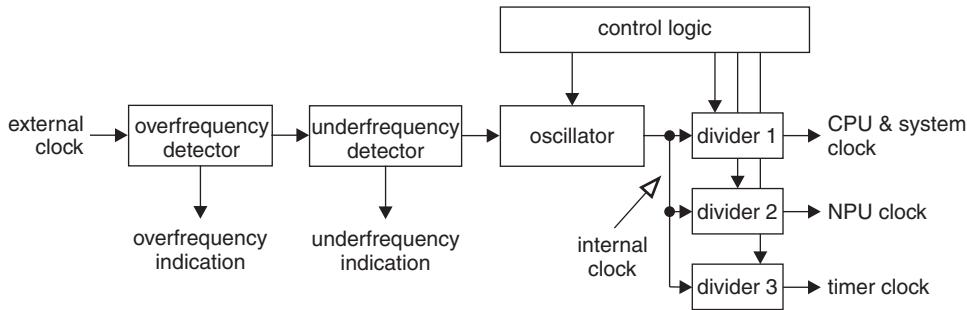
The demands on the processing power of smart cards are constantly increasing. This applies to the processor as well as components that support cryptographic algorithms. One way to meet these demands is to simply increase the frequency of the microcontroller clock signal, since the processing power is proportional to the clock frequency. Doubling the clock frequency thus doubles the performance of the processor. However, for reasons of compatibility with applicable standards, it is generally not possible to raise the clock frequency above 5 MHz.

To circumvent this limitation, the external clock can be multiplied by a fixed factor or a separate clock signal can be generated on-chip. This can be implemented using a phase-locked loop (PLL) circuit or an RC oscillator, as illustrated in Figure 5.25 on the facing page. Despite their high frequency stability, PLL circuits are rarely used because their settling time after the clock is re-enabled exceeds the quite strict requirements of the relevant standards.

With a separate oscillator or a clock multiplier, a smart card with an external 5-MHz clock signal can operate with a higher-frequency internal clock, such as 20 MHz. This can yield significant advantages, such as considerably reducing the computation time of a complex cryptographic algorithm or accelerating a Java virtual machine. However, there is an aspect that must always be borne in mind here: the data transmission rate always depends on the external clock signal, never the internal clock signal. This means that the external clock signal is always the reference for the UART.

Another aspect that must not be overlooked with clock multiplication or internal clock generation is that the current consumption is proportional to the clock frequency. The relationship is usually linear, which for example means that the current consumption increases by a factor of four if the clock frequency is quadrupled. Particularly with battery-operated terminals such as mobile equipment, increased current consumption is undesirable.

An elegant solution to this problem is provided by intelligent power management in the microcontroller, in which the operating system communicates the maximum allowable current consumption to the control logic of the oscillator. This logic then independently adjusts the oscillator to operate in a frequency range that avoids exceeding the specified maximum current consumption. For example, if a power-hungry NPU is energized while this control



**Figure 5.25** Block diagram of a possible internal clock multiplication circuit using a PLL oscillator followed by dividers to supply clock signals to the various microcontroller components. The clock signal for the coprocessor (NPU) is often taken directly from the PLL without any division. If the microcontroller has several timers, each one usually has its own configurable divider

loop is active, the internal clock frequency is automatically reduced to prevent the current consumption from rising above the allowable value.

Unfortunately, there is a small difficulty with this solution, which is that the specifications for smart cards used in GSM and UMTS telecommunication systems prohibit the use of free-running oscillators in smart card microcontrollers. This prohibition is based on the risk of interference to other circuitry in the mobile telephone. As long as the relevant portions of the specifications remain in force, it is not possible to use a continuously adjustable internal clock frequency or an oscillator that is fully independent of the external clock signal. However, these specifications do allow the use of a clock rate multiplier if the internal and external clock frequencies have a fixed ratio controlled by a multiplication factor.

Processor speed is not the only bottleneck in a smart card. The data transmission rates specified in the standards and the EEPROM or flash memory write and erase times do not benefit from increased clock frequencies, which somewhat limits the advantages of this approach. For some applications, using smart cards with a higher internal clock frequency can yield distinct benefits, especially considering the relatively small amount of additional circuitry (and thus area) required on the chip. For this reason, nearly all new types of smart card microcontrollers have internal clock multiplication capability.

#### 5.4.10 DMA (direct memory access)

DMA components have been used for a long time in the PC realm. DMA makes it possible to copy or shift data between memory regions at high speed, independent of the processor and in part in parallel with the other processor activities. It is often also possible to independently fill a memory region with a defined value. The main benefits of a DMA unit are that it offloads the processor and allows some program routines to be simplified. However, high-performance DMA components have only been available in rudimentary form in smart card microcontrollers up to now.

### 5.4.11 Memory management unit (MMU)

Modern smart card operating systems allow executable machine code to be downloaded to the card.<sup>8</sup> This code, which can be run using a special command, can be used for purposes such as executing a cryptographic function known only to the card issuer. However, it is in principle not possible to prevent such downloaded code from including a function for reading out secret data from the memory. This conflicts with the interests of operating system producers, who take great care to maintain the confidentiality of their operating system architecture and program code. Similar considerations apply to the secret keys and algorithms of the applications in the card.

An administrative solution would be to have the downloadable program evaluated by an independent body, but even this cannot ensure complete security, since a program that differs from the evaluated program might be downloaded instead, or the program code may be so confidential that its knowledge must be limited to the application provider.

One acceptable solution to this impasse is to equip the smart card microcontroller with a memory management unit (MMU). A MMU constantly monitors the memory boundaries of the currently running application program. The allowed memory region is specified by an operating system routine before the application is started, and it cannot be altered by the application program while it is running. This ensures that the application is completely encapsulated and cannot access memory areas forbidden to it. The barriers formed in this manner are called firewalls in analogy to fire protection walls in buildings. If an application attempts to access another memory region from (MMU), whose operating principle is illustrated in Figure 5.26 on the facing page. It will not be able to do so within a region delimited by a firewall, and this usually triggers an interrupt so that the operating system can take suitable action.

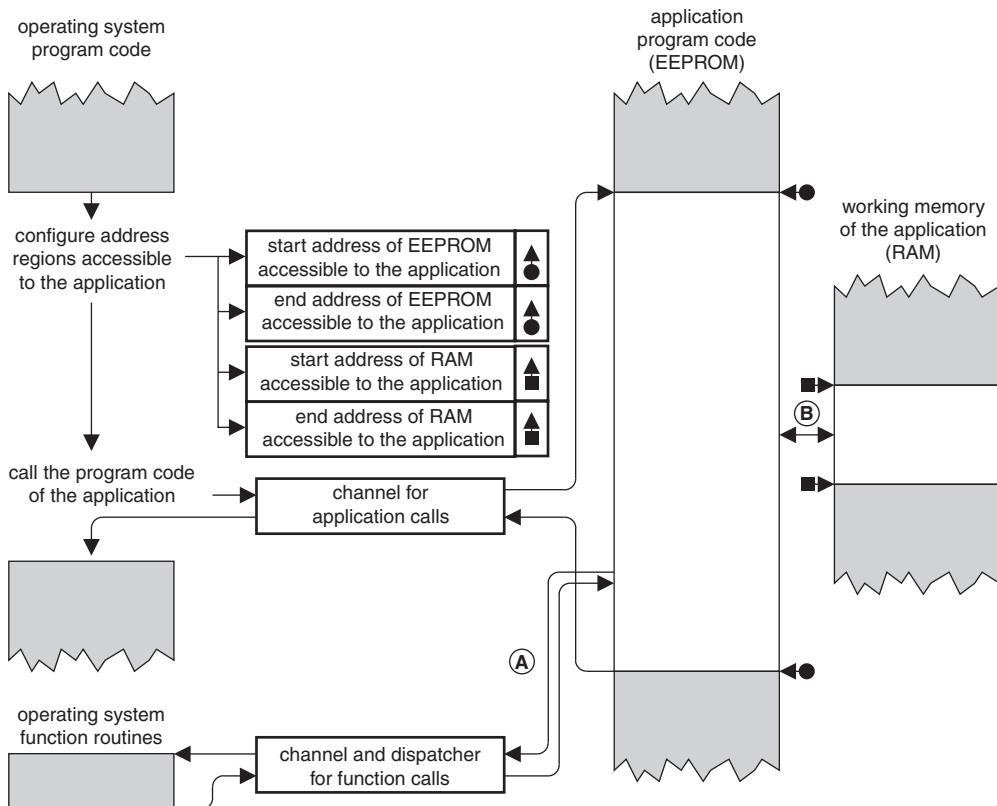
Although MMUs have been used in many areas for a long time, at present only a few smart card microcontrollers have a MMU. However, this supplementary hardware will become much more important in the future because it is the only practical way to isolate several applications reliably in a single smart card.

Another aspect of MMUs is their ability to relocate physically addressable memory regions to any desired address range in the logical memory space of the processor, as illustrated in Figure 5.27 on page 102. In part, this considerably simplifies the memory management functions of the smart card operating system, and it enables strict separation of the memory spaces of individual applications. If downloadable native code is used, the MMU can also be used to relocate it to the appropriate memory region, thus eliminating the need for manual relocation of the program code with the aid of the operating system.

However, there is a critical aspect that must be borne in mind when using an MMU in a smart card operating system. With the current state of technology, every MMU in a microcontroller is tailored to a specific type of microcontroller. Although this allows the operation and the area requirement of the MMU to be optimized, it comes at the price of the portability of the program code. In practice, the type of MMU that is used has a significantly greater impact on the operating system than the type of processor that is used. Consequently, MMUs have been used with considerable reserve up to now in smart card operating systems for large-scale applications, which are compelled to support a variety of hardware platforms.

---

<sup>8</sup> See also Section 13.14, ‘Downloadable Program Code’, on page 491

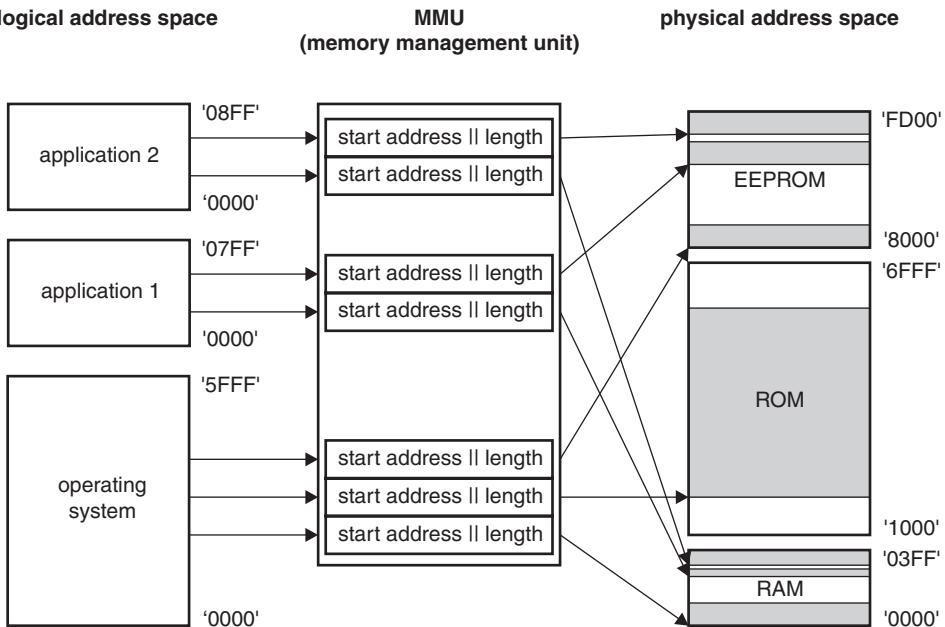


**Figure 5.26** Schematic representation of the operating principle of a hardware-based memory management unit (MMU) in a smart card microcontroller. Process A shows a call to an operating system function that is channeled by the MMU and controlled by a dispatcher. Process B is an example of a write/read access to an application memory area delimited by the MMU

### 5.4.12 Java accelerator

Java Card has become an industry standard for executable program code in smart cards. The Java virtual machine (VM) must interpret the bytecode instead of executing it directly, leading to an unavoidable loss of execution speed compared with native machine instructions, which are executed directly by the processor. Due to the widespread use of Java in smart cards, semiconductor manufacturers are interested in finding ways to get around this execution speed problem. There are two basic approaches to this.

In the first approach, a large part of the Java VM is incorporated in the smart card microcontroller in the form of dedicated hardware components that supplement the regular processor. The technological trend with this approach is in the direction of picoJava, which means a real IC that can process Java bytecode directly. This solution has two drawbacks, which are that the Java processor occupies additional space and that a full implementation of a Java VM is relatively costly. The advantage of this solution is its high execution speed.



**Figure 5.27** Schematic representation of the operating principle of a hardware memory management unit (MMU) in a smart card microcontroller for allocating logical address space to physical address space. This example shows an operating system and two applications that share the physically available memory via the MMU, which translates the logical address range starting at '0000' for each application into the appropriate physical address range

In the second approach, the native instruction set of the processor is extended to include typical Java machine instructions. This allows bytecodes supplied by the software VM to be processed immediately by the extended processor. This option is implemented using a processor lookup table containing a sequence of CPU microinstructions for each bytecode to be emulated. The advantage of this solution relative to the first one is that it requires less additional space on the chip, although its execution speed is somewhat lower.

Both solutions have the drawback that their implementations up to now have been specific to individual smart card microcontrollers. This considerably increases the effort of porting smart card operating systems, with the result that Java accelerators are rarely used. The situation here is similar to that with the highly useful DMA function, which is also rarely used due to a lack of hardware compatibility between different microcontroller products.

#### 5.4.13 Coprocessor for symmetric cryptographic algorithms

Up to now, DES has been used as the standard cryptographic algorithm for payment systems and telecommunication applications. This large market potential makes it worthwhile for semiconductor manufacturers to equip smart card microcontrollers with their own DES calculation units. This is not especially difficult in principle because DES was originally designed with hardware implementation in mind.

The benefits of DES calculation units for smart card microcontrollers can be seen quite easily by examining their performance figures. With a 3.5-MHz clock, they can achieve processing times of around 75 µs for a simple DES operation or 150 µs for a triple-DES operation with two keys.

There are also an increasing number of coprocessors for the AES algorithm available in smart card microcontrollers. They usually support all three possible key lengths (128, 196, and 256 bytes). In terms of technical feasibility, they are comparable to DES coprocessors because the AES algorithm is also relatively easy to implement in hardware.

With coprocessors of this sort, increasing the clock frequency yields a proportional reduction in the calculation time. Furthermore, an integrated DES or AES calculation unit in the microcontroller does not require significantly more chip area than the ROM or flash memory space necessary for a DES or AES algorithm implemented in software, so it does not increase the overall space requirement on the die.

#### 5.4.14 Coprocessor for asymmetric cryptographic algorithms

For calculations in the realm of public-key algorithms such as RSA and elliptic-curve algorithms, there are specially designed calculation units that are located on the silicon along with the usual functional components of a smart card microcontroller. These calculation units are limited to performing some basic calculations that are necessary for these types of algorithms: exponentiation and modulo calculations on large numbers. Both of these operations are essential elements of public-key encryption algorithms, such RSA and elliptic-curve algorithms. The speed of these components, which are optimized for these two arithmetic operations, results from their very broad architectures and high clock rates. In their specific application area, some of them can even outperform a high-performance PC.

The calculation unit is called by the processor, which either passes the data directly or passes a pointer to the data and then issues an instruction to start the calculation. After the task has been completed and the result has been stored in RAM, control of the chip is returned to the processor. In general, these coprocessors can handle all key lengths up to 1024 bits for the RSA algorithm, and in the medium term this will increase to 2048 bits. With elliptic curves, capacities in the range of 160 to 256 bits are presently common.

#### 5.4.15 Error detection and correction for nonvolatile memory

The essential limitation on the useful life of a smart card is imposed by the EEPROM or flash memory, with its technically limited number of possible write/erase cycles. One way to relax this limitation is to use software to calculate error correction codes for certain heavily used regions of nonvolatile memory, so that errors can be corrected. It is possible to implement the error correction method in the form of hardware on the chip. In this way, memory errors can be detected and corrected (as long as they are not too extensive) in a manner that is transparent to the software.

Naturally, additional nonvolatile memory space is needed to store the error detection and correction codes. Good error correction codes occupy a relatively large amount of memory space, so designers are confronted with a strategic dilemma. Effective error detection requires extra memory space, which may be as much as 50% of the memory area to be protected,

and the memory space needed for error correction mechanisms can only be used for this purpose. Lower-performance error correction requires less additional memory space, but its utility is questionable.

There are a few microcontrollers on the market that have EEPROM or flash error detection and correction implemented in hardware, but under certain conditions this may require half as much memory space as the memory region to be protected. As a result, the amount of EEPROM or flash memory available to the user is not especially large, although the useful life of EEPROM or flash memory with this error correction mechanism is several times longer than the usual value. This hardware extension is used only with EEPROM and NOR flash memory in the smart card realm, since other methods such as a flash translation layer<sup>9</sup> must anyhow be used with NAND flash memory.

#### 5.4.16 Mass memory interface

The amount of memory usually available in smart card microcontrollers is distinctly too small for some application areas. In such cases, NAND flash memory<sup>10</sup> can be used to expand the memory capacity of the smart card. This is currently implemented using two-chip solutions, but the first single-chip solutions are already foreseeable. Figure 2.12 on page 25 illustrates the general approach.

A special functional unit is included in the microcontroller to control the expansion memory. The main task of this unit is to provide the electrical interface to the memory chip. This is implemented as a bus to the external memory, which is fabricated using NAND flash technology. The width of the data bus is only eight bits in most cases, in order to minimize the number of connections between the microcontroller and the memory. Control lines for the bus and chip selection are also necessary, as well the supply voltage and ground lines.

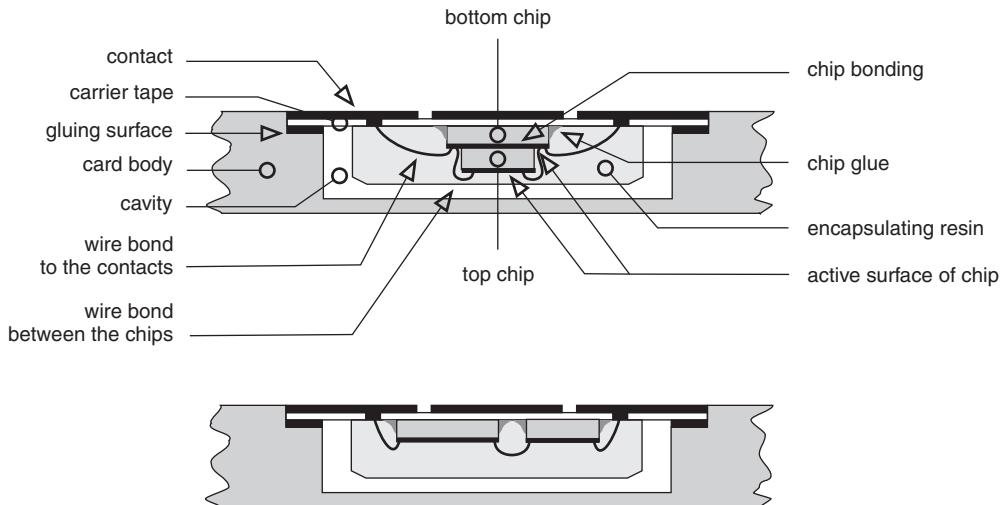
Some of these interface components can support up to four NAND flash memory chips with single-level cell (SLC) or multilevel cell (MLC) technology. Configuring different wait times for write operations is usually possible as well. The range of configuration options is typically large enough to enable almost any type of NAND flash memory component to be accessed.

The essential functions at the logic level are reading data from the NAND flash memory and writing data to it. This data can be transferred via a FIFO buffer, which for example may have a size of 16 bytes. In many cases, it is also possible to perform DMA transfers directly to or from the microcontroller RAM. The completion of a DMA transfer is signaled to the processor by a flag bit or an interrupt. Some microcontrollers also have additional functionality for calculating error detection and correction codes during write or read operations. In this case, the checksums are calculated blockwise for the individual sectors.

Data stored in a separate NAND memory can easily be read by unauthorized parties if they can establish electrical contact with the memory, so all of the stored data must be encrypted. This function is also provided by the mass memory interface. For this purpose, the corresponding key is loaded in a register, after which all data sent to the mass memory is symmetrically encrypted and all data coming from the mass memory is decrypted. This ensures the security of the data stored in the mass memory.

<sup>9</sup> See also Section 13.6.3, ‘Flash memory management’, on page 461

<sup>10</sup> See also Section 5.3.4, ‘Flash memory’, on page 90



**Figure 5.28** Cross section of a chip module containing two different chips electrically connected by bonding wires. In the upper drawing, the two chips are stacked, while in the lower drawing they are arranged side by side

### 5.4.17 Multichip module

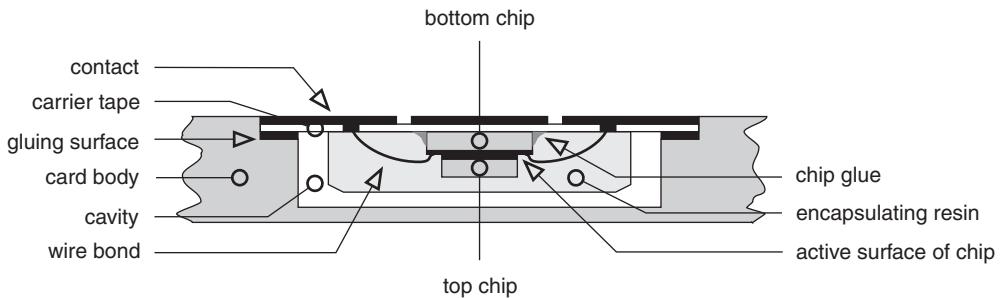
If the chip hardware must be extended for some reason, this entails a considerable expenditure of time and effort on the part of the chip manufacturer. There are only two ways to implement customer-specific hardware: building new hardware on top of the silicon of an existing chip family, or developing a two-chip solution.

In order to achieve an acceptable solution to this problem, a compromise approach that combines the features of these two approaches has been devised. A chip holding the new hardware unit can be glued directly to the existing chip and electrically connected to it by bonding wires, or a second chip can be glued in the module next to the microcontroller chip. Figure 5.28 shows both options.

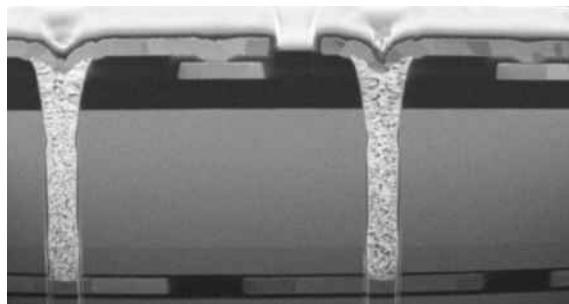
An alternative to using bonding wires for the connections between the two chips is to use face-to-face bonding. With this arrangement, both chips have raised contacts on their faces, and these contacts provide the electrical connections when the chips are glued together. Figure 5.29 on the following page illustrates this method.

This solution benefits from the fact that many smart card microcontrollers have several I/O lines that can be used for communication with the additional chip. The thickness of the resulting sandwich construction is not significantly greater than that of a normal chip, since the silicon substrates are ground thinner than usual in this case. A sandwich chip can thus be packaged in a standard module.

This method is well suited to satisfying customer-specific desires for additional hardware without making extensive or expensive modifications. It can be used to combine an existing chip with a new unit, which for example could provide a special serial interface for checking the security features of other chips. It also provides a means to incorporate a special ASIC with a secret cryptographic algorithm in a smart card. Another option is to connect a NAND flash memory chip that serves as mass memory for a smart card microcontroller. In



**Figure 5.29** Cross section of a chip module with two chips electrically connected by face-to-face bonding



**Figure 5.30** Cross-section of a VSI stack. The two vias connecting the stacked dice are clearly visible (Reproduced with permission from Giesecke & Devrient)

this case, the microcontroller chip must already have a suitable interface for accessing the flash memory.

In case of large production volumes in the range of several million, this method is only suitable for very high-end (and correspondingly expensive) chips, since the cost/benefit situation often favors the development of special chips in case of mass production. However, sandwich chips are often a good solution for small to medium production volumes. They are typically used in security modules for terminals or smart cards for pay-television decoders.<sup>11</sup>

#### 5.4.18 Vertical system integration (VSI)

Another technique used to extend chip hardware by combining semiconductor technologies that are incompatible on a single chip is vertical system integration (VSI), in which two or more dies that have been ground thin are bonded together mechanically to form a stack, and the individual chips are electrically interconnected by through contacts (vias) formed using semiconductor fabrication processes. Figure 5.30 shows an example of vias in a VSI stack.

VSI provides a very elegant means to increase the available chip area in units of the original area. With two stacked dice, twice the original area is available, and with four dice there is

<sup>11</sup> See [Kuhn 97]

four times as much area. In addition to allowing the available memory capacity to be increased considerably, this can significantly enhance chip security, since it presently impossible to access a chip sandwiched between two other chips using the analytical equipment commonly employed in the semiconductor industry without destroying the cover chips.

A simpler variant of VSI, which in principle can be scaled to as many layers as desired, is face-to-face bonding of two chips. Here the electrical connections are made by extremely precise positioning of two chips with their upper surfaces (faces) touching.

VSI and face-to-face bonding both enable significantly better chip hardware extensions than what can be achieved by connecting two chips with wire bonds.

## 5.5 EXTENDED TEMPERATURE RANGE

Smart card microcontrollers are also employed in areas that do not involve their direct use by people. Some examples are applications in automated equipment, telemetry equipment, and the broad field of the automotive industry. These application areas are collectively designated machine-to-machine (M2M) applications. The functionality and security requirements on the microcontrollers usually remain unchanged in such applications, but the operating temperature range is significantly larger. The temperature ranges specified for the telecommunication sector, which serve as examples for other application areas, are listed in Table 5.5. Additional form factors, such as SOP8<sup>12</sup> (small outline package) and QFN (quad flat pack, no leads) are also used to reduce the installed volume and increase reliability.

The only difference between normal smart card microcontrollers and microcontrollers used in M2M applications is that the latter have an extended temperature range, so they can be used with the same software as normal smart card microcontrollers without any need for software modifications. Semiconductor manufacturers typically offer M2M versions of several microcontrollers in their product lines. The extended temperature range is achieved by means of an adapted chip design and different process parameter values during semiconductor fabrication. Due to this additional effort, microcontrollers for M2M applications are somewhat more expensive and are produced specifically for this purpose.

**Table 5.5** Operational and storage temperature ranges of UICCs as specified in TS 102 221

Temperature class	Temperature range
A	-40 °C to +85 °C
B	-40 °C to +105 °C
A	-40 °C to +125 °C

<sup>12</sup> See also Figure 3.14 on page 37

# 6

# Information Technology Foundations

In contrast to all other types of cards, the specific properties of a smart card are determined by the microcontroller implanted in the card. The main function of the plastic card body is to hold the microcontroller. Of course, other components may be present in addition to the microcontroller, but they are not essential for the actual smart card functions. A basic understanding of certain aspects of information technology is necessary to understand the characteristics of these small computers and the information technology mechanisms based on them.

Our intention here is not to convey expert knowledge. This is anyhow not necessary for understanding the basic aspects of the techniques and methods used with smart cards. The basic knowledge presented in this chapter delves into the technical details only to the extent necessary to understand the fundamental relationships.

## 6.1 DATA STRUCTURES

Transmitting data unavoidably requires an exact definition of the data in question and its structure. Only then is it possible to recognize and interpret the data elements. Fixed-length data structures with rigidly defined sequences are a frequent cause of system failures. The best example of this was the conversion of the many different European currencies to the euro at the start of 2002. All systems and data structures with fixed currency definitions had to be upgraded at considerable cost. The same difficulties prevail in many smart card applications. Fixed data structures sooner or later lead to considerable effort and expense when they have to be modified.

However, the problem of structuring data has been around for a long time, and there are many intelligent solutions to this problem. One method that is very popular in the world of smart cards, and which is coming into more general use in information technology systems, comes from the field of data transmission. It is called Abstract Syntax Notation 1, or ASN.1 for short. ASN.1 is an encoding-independent description of data objects, originally developed for transmitting data between different computer systems. As an alternative to ASN.1, Extensible

**Table 6.1** Some ASN.1 data types frequently used with smart cards

Data type	Sort	Meaning
BOOLEAN	primitive	Boolean value: yes/no
INTEGER	primitive	negative and positive integers
OCTET STRING	primitive	byte sequence 1 byte = 8 bits
BIT STRING	primitive	bit sequence
SEQUENCE	constructed	several components combined to form a new data type

Markup Language (XML) could be used to structure data, but up to now it has not gained a foothold in smart card applications.

In principle, ASN.1 is a sort of artificial language that is suitable for describing data and data structures, rather than software programs. The syntax is standardized in ISO/IEC 8824, and the encoding rules are defined by ISO/IEC 8825. Both of these standards were developed from CCITT Recommendation X.409.

Describing ASN.1 in detail would require a book on its own, so here we only summarize a few essential aspects in order to give a general idea of how it works. For further information, we suggest you consult the relevant literature, such as Walter Gora [Gora 98].

ASN.1 has a collection of basic data types known as primitive data types, as shown in Table 6.1, and constructed data types (combinations of primitives), as shown in Figure 6.3 on page 114. Macros can also be used to extend the syntax of ASN.1 in order to obtain any desired extensions. Tables 6.2 and 6.3 on the facing page, and 6.4 on page 112 show a simple example of how ASN.1 can be used for tasks such as defining and encoding data.

ASN.1 objects are encoded using the classic TLV structure, where 'T' (tag) is the data packet identifier, 'L' (length) is the length, and 'V' (value) is the actual data.

The first field of a TLV structure is the tag of the data object in the following V field. To avoid the need for user-defined tags, which would open the door to incompatibility, there are standards that define tags for a variety of frequently used data structures. ISO/IEC 7816-6, for example, defines tags for objects used in general industrial applications, ISO/IEC 7816-4 defines tags for secure messaging, and EMV defines several other tags. Although specific tags are by no means used everywhere for the same types of data elements, efforts are being made to achieve standardization.

The basic idea of encoding data using ASN.1 is to prefix each data object with unique tag and length information. The rather complex syntax of the description language also enables users to define their own data types and to nest data objects. The original idea, which was to create a generally valid syntax that would form the basis for data exchange between fundamentally different computer systems, is scarcely used in smart cards. Currently, only a very small part of the available syntax is used in this area, mainly due to the very limited memory capacity of smart cards.

The Basic Encoding Rules (BER) for ASN.1 are defined in the ISO/IEC 8825 standard. Data objects created according to these rules are called BER-TLV encoded data objects. A BER encoded data object has a tag, a length field and the actual data part, with an optional end marker. Certain bits of the tag are predefined by the encoding rules. This is shown in detail in Figure 6.1 on page 112. The Distinguished Encoding Rules (DER), which are a subset of the BER, specify the encoding of the length field, which can have a size of one, two, three, or four bytes. A basic summary of the BER and DER can be found in Burton Kaliski [Kaliski 93].

**Table 6.2** A simple example of data type definition using ASN.1

---

SC_Controller ::= SEQUENCE {	Definition of a new data type for SC_Controller
Name IA5String,	The name of the microcontroller is an ASCII string
CPUPower CPUPower,	CPUPower refers to the definition of CPUPower
NPU BOOLEAN,	Boolean value (yes/no) indicating whether a coprocessor (NPU) is present
EEPROMSize INTEGER,	The size of the EEPROM is an integer value
RAMSize INTEGER,	The size of the RAM is an integer value
ROMSize INTEGER }	The size of the ROM is an integer value
CPUPower ::= ENUMERATED {	Definition of a new data type for CPUPower as an enumerated type.
8bit (8),	Selectable value for 8-bit CPU type
16bit (16),	Selectable value for 16-bit CPU type
32bit (32) }	Selectable value for 32-bit CPU type
SuperXS SC_Controller ::= {	Specific instance of data type SC_Controller with attributes of SuperXS
Name "XS 8-bit",	The microcontroller name is 'XS 8 bit'
CPUPower 8,	It has an 8-bit CPU
NPU true,	A coprocessor (NPU) is present
EEPROMSize 1024,	The EEPROM size is 1024 bytes
RAMSize 256,	The RAM size is 256 bytes
ROMSize 8192 }	The ROM size is 8192 bytes

---

**Table 6.3** The data definition from Table 6.2 filled with the data of a particular microcontroller

---

SuperXS SC_Controller ::= {	Specific instance of data type SC_Controller with attributes of SuperXS
Name "XS 8 Bit"	The microcontroller name is 'XS 8 Bit'.
CPUPower 8,	It has an 8-bit CPU
NPU true,	A coprocessor (NPU) is present
EEPROMSize 1024,	The EEPROM size is 1024 bytes
RAMSize 256,	The RAM size is 256 bytes
ROMSize 8192 }	The ROM size is 8 192 bytes

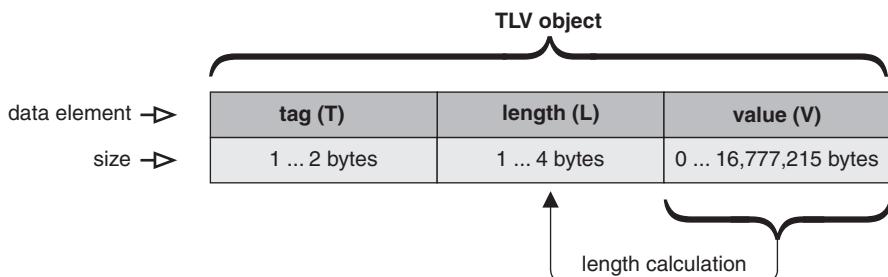
---

The two most significant bits of the tag encode the class of the subsequent data object. The class indicates the general type of the data object. The ‘universal’ class includes general data objects, such as integers and character strings. The ‘application’ class indicates that the data object belongs to a particular application or standard, such as ISO/IEC 7816-6. The other two classes, ‘context-specific’ and ‘private’, fall under the heading of nonstandardized applications. This is shown in Table 6.5 on the next page.

The bit following the two class bits indicates whether the tagged object is constructed from other data objects. The five least significant bits are the actual tag code. As it can only assume values in the range of 0 to 30 due to the limited address space, it is possible to point to the following byte by setting all five bits to 1. All values from 31 to 127 are allowed in the second byte. Bit 8 of the second byte is a pointer that is reserved for future use, so it cannot presently be used.

**Table 6.4** The data of a particular microcontroller from Table 6.3, encoded using the ASN.1 BER

'30 1C'	Tag '30' for a string with a length of 28 bytes ('1C')
'16 08 58 53 20 38 20 42 69 74'	Tag '16' for an IA5 string with a length of 8 bytes ('08') and content '58 53 20 38 20 42 69 74' ("XS 8-bit")
'0A 01 08'	Tag '0A' for an enumerated data type with length 1 byte ('01') and content '08'
'01 01 FF'	Tag '01' for a Boolean data type with length 1 byte ('01') and content 'FF', which corresponds to the value 'true'
'02 02 04 00'	Tag '02' for an integer data type with length 2 bytes ('02') and content '04 00' (1 024)
'02 02 01 00'	Tag '02' for an integer data type with length 2 bytes ('02') and content '01 00' (256)
'02 02 20 00'	Tag '02' for an integer data type with length 2 bytes ('02') and content '02 00' (8 192)

**Figure 6.1** The principle of BER-based TLV encoding according to ASN.1**Table 6.5** ASN.1 tag encoding

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
Byte 1								
0	0	...	...	...	...	...	...	universal class
0	1	...	...	...	...	...	...	application class
1	0	...	...	...	...	...	...	context-specific class
1	1	...	...	...	...	...	...	private class
...	0	...	...	...	...	...	...	primitive data object
...	1	...	...	...	...	...	...	constructed data object
...	...	X	X	X	X	X	X	tag code (0 ... 30)
...	...	1	1	1	1	1	1	pointer to the following byte (byte 2), which specifies the tag code
Byte 2								
0	X	X	X	X	X	X	X	tag code (31 ... 127)

**Table 6.6** Structure of the BER length field in ASN.1

Number of length bytes	First byte	Subsequent bytes	Encoded length	Meaning
1 byte	'00' ... '7F'	—	0 ... 127	One byte is used for these length values
2 bytes	'81'	'00' ... 'FF'	0 ... 255	Two bytes are used for these length values
3 bytes	'82'	'00 00' ... 'FF FF'	0 ... 65 535	Three bytes are used for these length values
4 bytes	'83'	'00 00 00' ... 'FF FF FF'	0 ... 16 777 215	Four bytes are used for these length values
5 bytes	'84'	'00 00 00 00' ... 'FF FFF FF'	0 ... 4 294 967 295	Five bytes are used for these length values

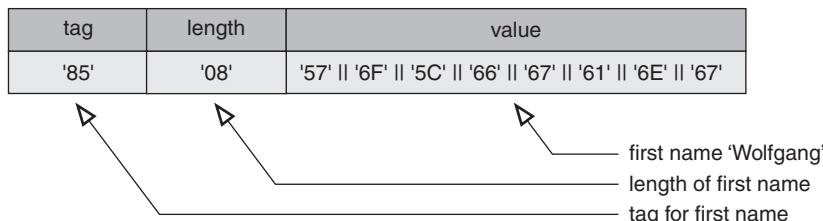
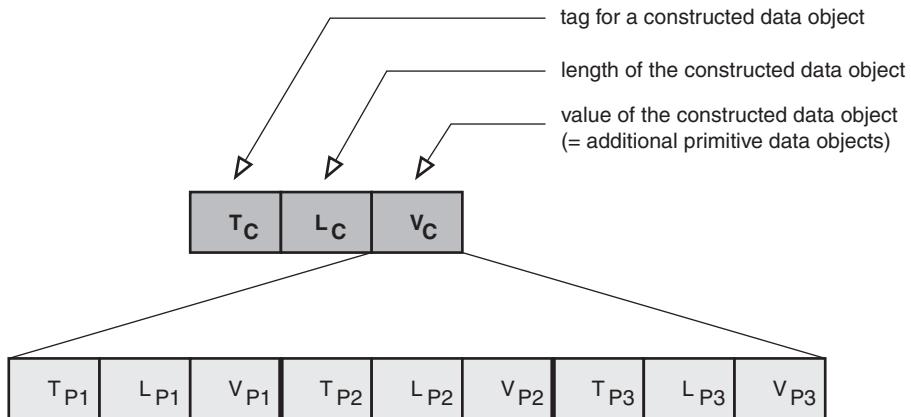
**Figure 6.2** TLV encoding of the name 'Wolfgang'

Table 6.6 lists the number of length bytes necessary for a TLV object. Here again, the encoding structure is flexible. The number of bytes needed to indicate the length of the data part of a TLV object depends on the size of the data part. A single-length byte is sufficient for data volumes up to 127 bytes, while correspondingly longer length fields are defined for greater data volumes.

The standard also defines the term 'template'. A template is a data object that acts as a container for other data objects. ISO/IEC 7816-6 defines tags for data objects used in smart card applications that extend across sector boundaries. ISO 9992-2 covers the smart card financial transactions sector.

This data encoding method has several characteristics that are particularly beneficial with smart cards. As the available memory space is generally never enough, using data objects based on ASN.1 can yield considerable space savings because TLV encoding makes it possible to transmit and store variable-length data without major complexity. This allows memory to be used very economically. This is illustrated in Figure 6.2, which shows the TLV encoding of a name.

Subsequent extensions to data structures can be undertaken very easily with ASN.1, since all that is necessary is to add additional TLV-coded data objects to the existing data structure. Full compatibility with the previous version is retained as long as the previous TLV objects are not deleted. The same is true of new versions of data structures in which changes have been made with respect to the previous encoding. This is a straightforward process that only requires modifications to the tags. It is equally easy to represent the same data using different



**Figure 6.3** Basic scheme for forming constructed TLV-coded data structures from several primitive TLV-coded data objects. The subscripts ‘C’ and ‘P’ stand for ‘constructed’ and ‘primitive’

	T	L	V	T	L	V	T	L	V
option 1	'85'	'07'	"Manfred"	'87'	'05'	"Braun"	'84'	'04'	"Ing."
<hr/>									
option 2	T	L	V	T	L	V	T	L	V
	'84'	'04'	"Ing."	'85'	'07'	"Manfred"	'87'	'05'	"Braun"
<hr/>									
option 3	T	L	V	T	L	V	T	L	V
	'87'	'05'	"Braun"	'85'	'07'	"Manfred"	'84'	'04'	"Ing."

**Figure 6.4** An example of sequence independence in a TLV structure. The tags used here have been selected arbitrarily and do not correspond to any relevant standard

encodings. Collectively, these advantages explain why the ASN.1 syntax, based on TLV encoding, is particularly popular in the smart card industry.

The main disadvantage of ASN.1 data objects is that the administrative overhead is rather high if the volume of user data is small. For example, if the user data is only one byte, two additional bytes (tag and length) are still needed for its administrative data. However, the relative overhead decreases as the volume of user data increases. ASN.1-structured data in the German health insurance card is a good example of this. Each data object has 70 to 212 bytes of user data. The total volume of administrative data for each data object is 36 bytes, which means that the administrative overhead is 17 to 51 %.

This is further illustrated and summarized in Figure 6.4. Suppose we wish to store surnames, given names and titles in a file with a transparent data structure. Regardless of the specific ASN.1 description, the TLV-encoded data will have the illustrated structure.

When evaluating this data structure, the computer compares the first tag with all tags known to it. If it finds a match, it recognizes the first object as a given name. It reads the length of this object from the next byte. The subsequent bytes are then the actual object (the given name). This is followed by the next TLV object, whose first byte is the tag for a surname. The computer recognizes this using exactly the same process as for the first object.

If it becomes necessary to extend the data structure, such as by adding a name affix, a new type of data object can simply be added to the existing structure. The position of the new object in the structure is unimportant. The extended structure remains fully compatible with the previous structure, since the new type of data object has its own tag and is thus unambiguously identified. Programs that only know the old tags will not be upset by the new one, since they do not recognize it and thus automatically skip it. Other programs that do know the new tag can evaluate it, but even if the old structure is used, they will not experience any problems.

## 6.2 ENCODING ALPHANUMERIC DATA

Alphanumeric data in the files and data objects of smart cards can be stored in a wide variety of formats. In part, this is a result of intensive memory space minimization measures and a lack of coordination between the various applications and specifications, and in part it is due to the success of smart cards in countries outside Western Europe, which have their own alphabets. Accordingly, the original seven-bit and eight-bit character sets must be replaced by more powerful encoding schemes for alphanumeric data.

### 6.2.1 Seven-bit code (ASCII)

A total of 128 ( $2^7$ ) characters can be represented with a seven-bit code. The most widely used international seven-bit encoding scheme, which is commonly known as the ASCII (American Standard Code for Information Interchange) code, is specified in ISO/IEC 646. The importance of ASCII has been declining steadily for many years because the number of characters it can represent is much too small.

A seven-bit character set based on the ASCII set is used in the GSM system to represent alphanumeric data. It is defined in the TS 23.038 specification, and it has become very important in the mobile telecommunication sector.

### 6.2.2 Eight-bit code (PC ASCII)

The most commonly used eight-bit code with 256 characters ( $2^8$ ) is derived from seven-bit ASCII and is standardized in ISO/IEC 8859. It consists of two seven-bit code tables that specify control characters as well as printable characters. The low-order table is identical to the seven-bit ASCII table and is always the same. Various high-order tables can be used to accommodate country-specific character sets. Probably the best-known high-order code table is Latin 1, which contains the characters specific to the countries of Western Europe. Latin 2, by contrast, contains special characters for East European countries. ISO/IEC 8859 consists

of 16 parts in total, which define a set of high-order code tables for the character sets of a wide variety of languages.

The characters of the Latin 1 code table in ISO/IEC 8869 are also found with slightly modified encoding in DOS as Code Table 850 according to the IBM register, in the form of PC ASCII, and as the ANSI code in Windows.

EBCDIC (extended binary coded decimal interchange code), which is still widely used with mainframe computers, is not used with smart cards.

### 6.2.3 Sixteen-bit code (Unicode)

Codes with a width of 16 bits allow 65 546 ( $2^{16}$ ) characters to be represented. The only example of such a code is Unicode, which was developed as a private initiative by the Unicode Consortium [Unicode] as an industry standard.

The first 256 Unicode characters are identical to ISO/IEC 8859 Latin 1, so there is at least upward compatibility in this part of the character code. Although the number of characters that can be encoded using a sixteen-bit code is sufficient to represent the characters of the most important living languages, it is unfortunately not sufficient to represent all existing characters.

To compensate for this, a sort of escape sequence (surrogate pairs) is incorporated in the sixteen-bit character code of the current version of Unicode (3.0). This allows a supplementary byte to be used, so that up to one million characters can be represented.

### 6.2.4 Thirty-two-bit code (UCS)

Unicode was originally limited to 65 536 characters. Although this limitation does not often cause problems, it can be avoided by using an extended character encoding scheme. ISO/IEC 10646 specifies a 32-bit code called Universal Character Set (UCS), which can be used to represent 4 294 967 296 ( $2^{32}$ ) characters, although only half of them (2147 483 648) are actually used.

The four bytes of the UCS are called (in decreasing order of significance) group, plane, row, and cell. UCS thus consists of 256 groups of 256 planes, each of which has 256 rows of 256 cells. A plane specifies 65 536 characters. The lowest plane, which is Group 0, Plane 0, is called the basic multilingual plane (BMP) and is identical to Unicode. The lowest row, which is Group 0, Level 0, Row 0, automatically corresponds to the character set of ISO/IEC 8859 Latin 1, and the first 128 characters are identical to the ASCII code.

This can be illustrated by the simple example shown in Figure 6.5 on the next page. The letter 'A' is encoded as '30' in 7-bit ASCII and 8-bit ISO/IEC 8859 Latin 1. As the first 256 characters of Unicode are identical to ISO/IEC 8859 Latin 1, the letter 'A' is encoded as '00 30' in 16-bit Unicode or '00 00 00 30' in 32-bit UCS.

UCS is the only character encoding scheme that allows all characters of all living and dead languages to be represented by unique numerical values. Consequently, UCS is the most important encoding scheme for future applications, despite its memory requirement of four bytes per character.

There are three commonly used schemes, called UCS transformation formats (UTFs), for translating the codes of 32-bit UCS and 16-bit Unicode characters. UTF-8 translates characters into variable-length byte strings whose least significant seven bits correspond to ASCII.

<b>7-bit code (ASCII)</b>	7	1
<b>8-bit code (ISO/IEC 8859)</b>	8	1 8 1
<b>16-bit code (Unicode)</b>	8	1 8 1 8 1
<b>32-bit code (UCS)</b>	8 group 1 8 plane 1 8 row 1 8 cell 1	32 25 24 17 16 9 8 1

**Figure 6.5** Comparison of the most commonly used international 7-, 8-, 16- and 32-bit encoding schemes for alphanumeric characters

UTF-16 uses 16 bits for encoding and thus corresponds to the BMP of UCS and 16-bit Unicode. UTF-16 is also called UCS-2 because it uses two bytes for character encoding. UTF-32 corresponds to the usual four-byte representation of UCS, for which reason it is also called UCS-4.

## 6.3 SDL NOTATION

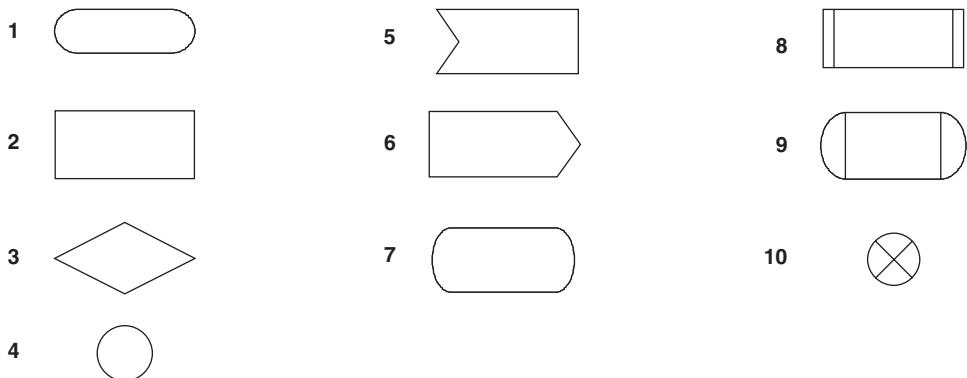
In addition to the usual UML notation, SDL notation is used in this book to describe states and state transitions. This notation is frequently used in the smart card domain to describe state-oriented mechanisms, such as those used in communication protocols. ‘SDL’ stands for ‘Specification and Description Language’, and it is described in detail in detail in CCITT Recommendation Z.100.

SDL notation is similar to conventional flowchart notation, but it describes states and state transitions instead of program flows. SDL diagrams are constructed using standardized individual symbols interconnected by lines. The flow is always from top left to bottom right, so the lines connecting individual symbols do not need arrowheads to identify their start and end points.<sup>1</sup> Figure 6.6 on the following page shows the symbols used in this book. They are only a subset of a much larger set of symbols defined in Z.100, but they suffice as a basic set for use with smart cards.

In simplified form, the notation can be regarded as a description of a system comprising a certain number of processes, where each process is a state machine. When a state machine is in a stable state, it can receive an external signal. Depending on the data it receives, the machine may then attain a certain new state. Additional actions may occur between the initial and new states, such as receiving and transmitting data or computing a value.

The Start symbol (1) denotes the start of a process. Most SDL diagrams begin with this symbol. The Task symbol (2) indicates a specific activity, which is described by text inside the box. With this symbol, there is no additional detailed description in the form of a subroutine. The Decision symbol (3) allows a query for the state transition, to which the answer may be ‘yes’ or ‘no’. The Label symbol (4) marks a link to another SDL diagram and is primarily used to divide large diagrams into several smaller diagrams.

<sup>1</sup> See Section 9.3.1, ‘The T = 0 transmission protocol’, on page 255 for a detailed example of an SDL diagram



**Figure 6.6** The SDL notation symbols (as defined in CCITT Z.100) used in this book: Start (1), Task (2), Decision (3), Label (4), Input (5), Output (6), State (7), Subroutine (8), Subroutine Start (9), and Subroutine End (10)

The Input (5) and Output (6) symbols represent interfaces to the outside world. The exact input and output parameters are described inside the symbol. The State symbol (7) is used to describe a state. The state attained at each stage is indicated by this symbol.

The final three symbols describe subroutines. The Subroutine symbol (8) indicates that the content of the box is described in more detail elsewhere. The Subroutine Start (9) and Subroutine End (10) symbols delimit the detailed description of a subroutine.

## 6.4 STATE MACHINES

A state machine is a type of automaton. A common example of an automaton is a vending machine, into which you insert a coin and then press a button. After this, you can open a compartment and remove your selection.

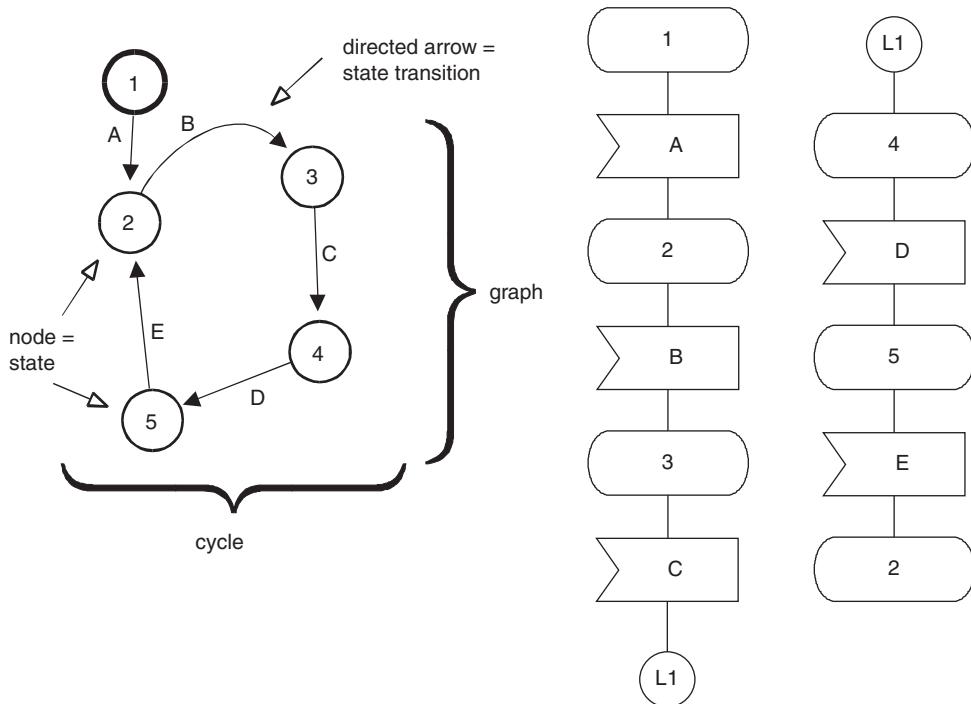
In slightly more abstract terms, this automaton defines a chain of events involving various state transitions. In the initial state, the automaton waits for money to be inserted. Any other action, such as pressing a button, will not cause anything to happen. Only the insertion of a coin causes a transition from the initial state to the ‘money inserted’ state. The next transition occurs as a result of pressing a button, following which the automaton allows a compartment to be opened.

In informatics, state machines can be very effectively visualized using graphs or Petri networks. These are not only useful for modeling state machines; they can also be used to investigate certain properties of the systems they describe. The objectives are to identify any deadlocks that may occur in the process and to ensure correct command processing.

### 6.4.1 Basic theory of state machines

Our objective here is to provide an introduction to the state diagrams used to describe smart card applications and a general explanation of how to interpret these diagrams.

A state diagram is a type of graph that represents a set of states and the interrelationships of these states. As illustrated in Figure 6.7, the states are shown as nodes, and their relationships



**Figure 6.7** Examples of two different representations of state machines. On the left is a directed state diagram, and on the right an equivalent SDL diagram

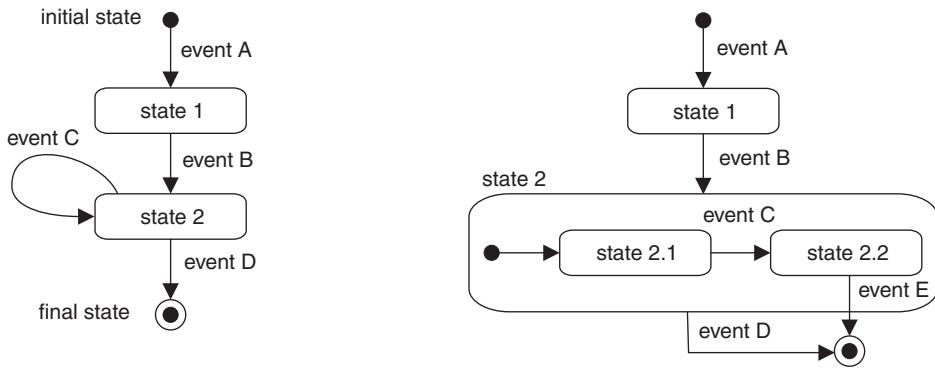
**Table 6.7** Tabular form of the state diagrams shown in Figure 6.7

To state	From state 1	2	3	4	5
1	—	—	—	—	—
2	A	—	—	—	E
3	—	B	—	—	—
4	—	—	C	—	—
5	—	—	—	D	—

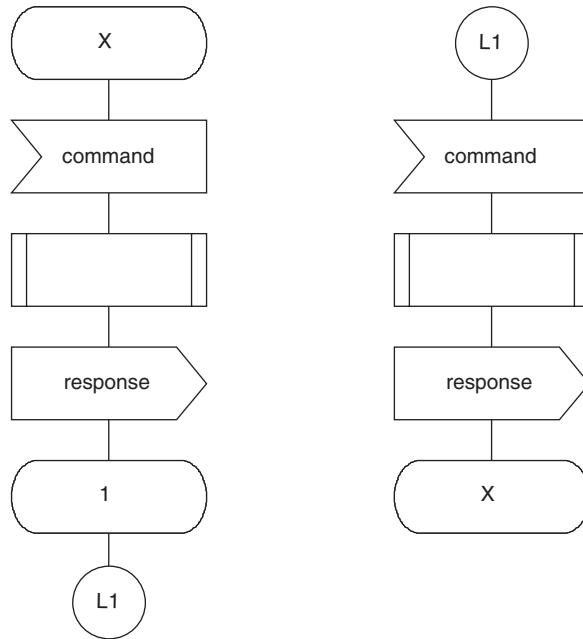
are shown as lines. If a line indicates a direction, which means that it has an arrowhead at one end, it is called a directed line and the graph is a directed graph. The arrowhead indicates the direction in which a state transition can take place. The layout of the nodes and lines in the graph plays no part in the interpretation of the diagram. A sequence of nodes connected by lines is called a path. If the first and last nodes are the same and there is more than one node, the path is called a loop.

State machines can also be described in tabular form, as illustrated in Table 6.7, or in UML notation as illustrated in Figure 6.8. Figure 6.9 shows an example of a simple state machine.

This is only a very small part of graph theory, but it is essentially all we need in order to describe states and their associated state machines in smart card applications.



**Figure 6.8** Examples of state diagrams in UML notation. The diagram on the left shows a simple sequential state flow, while the diagram on the right shows a state diagram with substates



**Figure 6.9** An example of a simple smart card state machine with two states: X and 1

#### 6.4.2 Practical applications

An advantage of processor cards relative to simple memory cards is that individual command sequences can be defined. It is thus possible to precisely specify the parameters and execution sequences of all commands. This provides additional protection against unauthorized access, along with object-oriented authorization privileges for file access. However, the capabilities offered by smart cards in this respect vary greatly. Simple operating systems usually cannot manage state machines, but with modern operating systems it is even possible to define application-specific state machines that utilize command parameters.

A typical example of a simple state machine is provided by the two commands needed to authenticate a terminal. The first command is GET CHALLENGE, which requests a random number from the card. This activates a state machine that accepts only the authentication command EXTERNAL AUTHENTICATE as the next command. If the card receives this command, the process completes and all other commands are allowed. Otherwise (if the card receives any command other than the expected authentication command) the state machine generates an error message and the process is aborted. The command sequence must then be restarted from the beginning.

Such simple state machines have several major advantages in smart cards. As only a very small number of commands in a rigidly defined sequence are defined, they require little memory space and program code. In many applications, it is sufficient to protect file contents by using object-oriented access mechanisms, without imposing any other restrictions on command sequences. Only a few processes, such as authentication, must follow prescribed sequences. This can be implemented with very little memory using simple state machines.

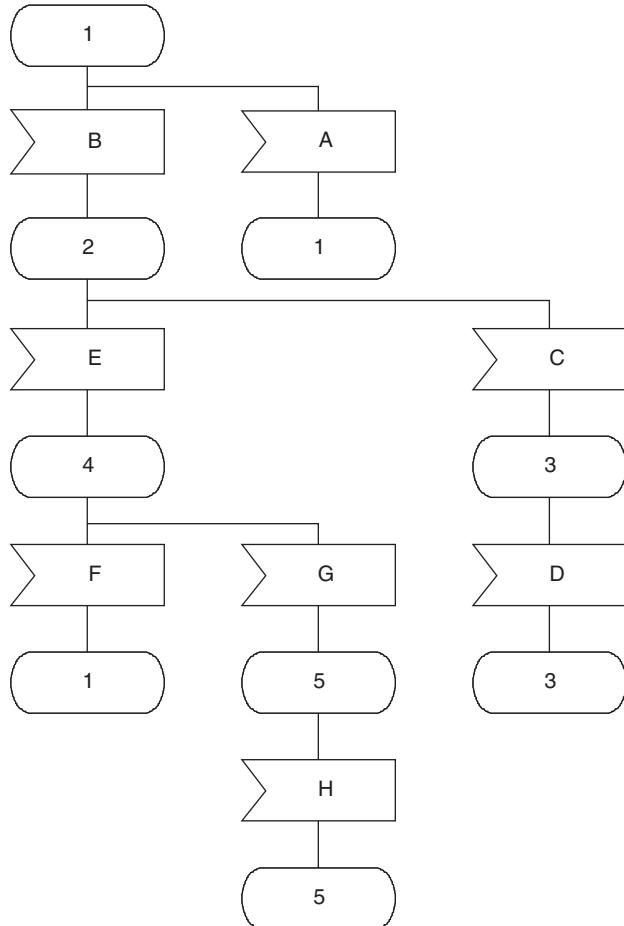
These simple state machines can be extended to check all commands and command parameters in a defined graph before they are executed. Depending on the implementation of the state machine, under certain conditions it is even possible to dispense with object-oriented file access protection because the state machine can perform all the necessary checks before a command is actually executed. Of course, an error in the state diagram could have fatal consequences for the security of the system. As it is very difficult to prove the complete absence of errors in the state diagrams of complex state machines, file access protection is still used in practice. Generating a correct description of all the processes and commands in a smart card is very laborious, so it is often determined empirically in part.

Now that we have described the benefits of state machines, in fairness we must mention their drawbacks. Implementing a state machine with the required capabilities is very time-consuming in terms of both design and subsequent programming. In addition to the program code, memory space must also be allocated for the graph of the state machine. The amount of memory space naturally depends on the complexity of the graph to be executed. The amount of information contained in a graph having many states and a corresponding number of transitions can easily be very large relative to typical smart card memory sizes. State machines for smart cards are addressed by the ISO/IEC 7816-9 standard. It describes access control descriptors (ACDs), which specify the commands that are permitted in a specific state and their associated parameters. The smart card operating system can use these ACDs to monitor the defined state machines.

In order to illustrate the capabilities of state machines in capsule form, Figure 6.10 on the following page shows a state diagram for a small application. Its operation is described below.

After a reset, the smart card is in the initial state, denoted by 1. In this state, every file in the directory can be selected with the SELECT command. This does not cause a state transition. All other commands except PIN verification (VERIFY) are prohibited, and the card responds to such commands with an error message. After successful PIN verification, the state machine reaches state 2.

Two commands are permitted in state 2. The first path leads via file selection (SELECT) to state 3, where the selected file can be read. The second path originating from state 2 leads to state 4 after the terminal requests a random number from the card (GET CHALLENGE). From here, every command except EXTERNAL AUTHENTICATE leads back to the initial state (1). If the terminal has been successfully authenticated, the card reaches state 5. According to the diagram, in this state files can be selected and written using the SELECT and UPDATE BINARY commands.

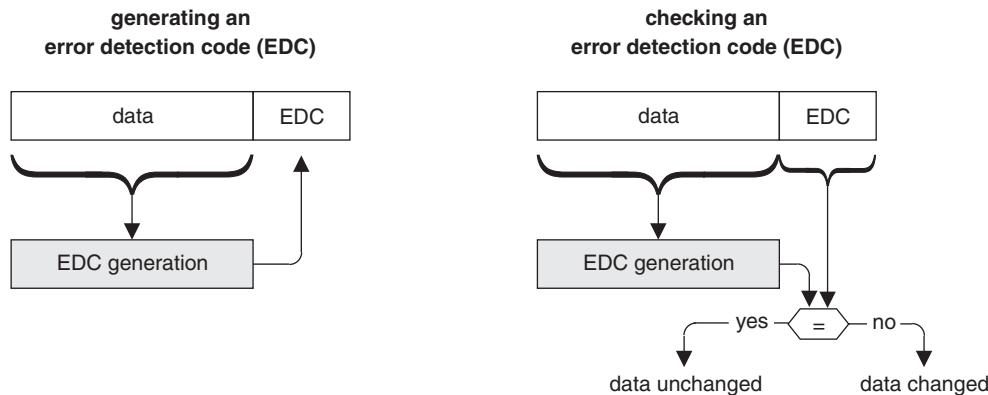


**Figure 6.10** Example of a smart card state machine with the following states and transitions: initial state (1); intermediate states (2 & 4); final states (3 & 5); SELECT (A & C); VERIFY (B); READ BINARY (D); GET CHALLENGE (E); all commands except EXTERNAL AUTHENTICATE (F); EXTERNAL AUTHENTICATE (G); SELECT/UPDATE BINARY (H)

In this diagram, states 3 and 5 cannot be exited during a session, so they represent the two final states. A transition to state 1 is only possible via a card reset. This is not shown in the diagram because the awareness of any state machine is limited to the current session. No information at all is transferred in a state machine from one session to the next.

## 6.5 ERROR DETECTION AND CORRECTION CODES

Whenever data is transmitted or stored, it should be possible to detect any changes to the data. In particular, stored programs must be protected against corruption because a single altered bit of program code could corrupt the program or modify its execution to such an extent that it no longer provides the necessary functions. In particular, the nonvolatile memory used in



**Figure 6.11** The basic processes for using an error detection code (EDC) to protect data

smart cards (EEPROM and flash memory) is relatively sensitive to external influences such as heat and voltage fluctuations. Consequently, the security-related regions of these memories must be protected so that undesired changes can be recognized by the operating system and their negative effects can be avoided.

Error detection codes (EDCs) are used to protect sensitive memory contents. They allow changes in the protected memory region to be detected with a certain probability, which depends on the type of code used.

Error correction codes (ECCs) are an extension of error detection codes. They make it possible not only to detect errors in the protected data, but also to correct errors to a limited extent and with a certain probability of accurate correction.

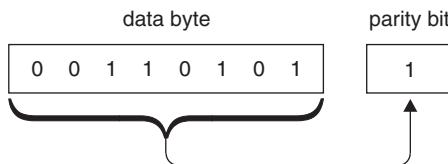
All of these codes operate on the principle of associating a checksum with the protected data. The checksum is usually stored along with the protected data. It is computed using a generally known algorithm, not a secret one. The calculated EDC can be used as necessary to check the data for changes. This is done by comparing the stored checksum with one computed anew. Figure 6.11 shows the basic processes.

A wide variety of mathematical methods are used for error detection and correction. Some of them provide better protection for the more significant bits of the data in order to minimize adverse effects on numerical values. However, in most cases the additional size and complexity of the program code vastly exceeds the benefits of such algorithms. Consequently, the most commonly used methods are byte-oriented methods in which error detection does not distinguish between the more significant and less significant parts of a byte.

Error detection and correction codes are very similar to message authentication codes (MACs) and cryptographic checksums (CCSs).<sup>2</sup> However, there is a fundamental difference. EDC and ECC checksums can be computed and checked by anyone. By contrast, computation of a MAC or CCS requires a secret key because these codes are designed to provide protection against intentional manipulation of the data instead of accidental corruption of the data.

The most widely known type of error detection code is doubtless the parity bit, which is appended to each byte in many data transmission protocols and some types of memory modules. Before a parity bit can be calculated, it is necessary to specify whether even or

<sup>2</sup> See also Section 7.1.4, ‘Message authentication code and cryptographic checksum’, on page 155



**Figure 6.12** Example of error detection using a supplementary odd parity bit

odd parity is to be used. With even parity, the value of the parity bit is chosen such that the total number of bits with a value of 1 in the combination of the data byte and the parity bit is an even number. With odd parity, the total number of bits with a value of 1 must be an odd number.

The parity mechanism can reliably detect a single incorrect bit in each byte, as illustrated in Figure 6.12. However, error correction is not possible because the parity bit does not provide any information on the position of the changed bit.

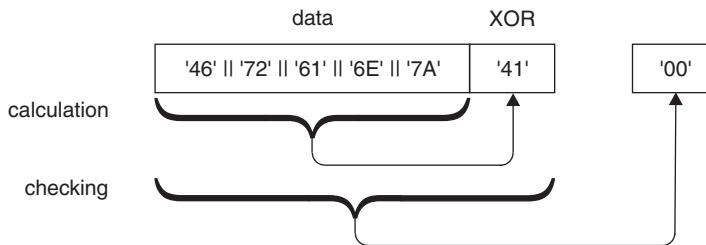
If two bits in a byte are simultaneously wrong, the parity will not change and no error will be detected. Another drawback of parity-based error detection is the relatively large overhead of one parity bit for every eight data bits. This represents an additional memory requirement of 12.5 %. Furthermore, it is difficult to use supplementary parity bits with byte-structured memory because it considerably increases the complexity of the software. This is why parity bits are not used for error detection in smart card memories. CRC checksums and Reed–Solomon (RS) codes are more suitable for this purpose.

### 6.5.1 XOR checksums

XOR checksums, which are also known as a longitudinal redundancy checks (LRC) due to the computation method that is used, can be calculated very easily and quickly. These are both important criteria for error detection codes used in smart cards. In addition, the algorithm can be implemented extremely easily. XOR checksums are only occasionally used to protect memory contents; they are typically used in data transmission (such as with the ATR in the T = 1 transmission protocol).

An XOR checksum is computed by performing consecutive logical XOR operations on all of the data bytes. In other words, byte 1 is XORed with byte 2, the result of this is XORed with byte 3, and so on. If the checksum is appended to the data to be checked and a new checksum is computed for the combination of the data and the original checksum, the result is '00'. This is the simplest way to check whether the data and the checksum still have their original values and are thus uncorrupted. These processes are illustrated in Figure 6.13.

The primary advantage of XOR checksums is that they can be computed quickly using a simple algorithm. The algorithm is so simple that the necessary code in assembly language is only 10 to 20 bytes. One reason for this is that the XOR operation is directly available in all processors as a machine instruction. In addition, an algorithm for XOR checksum computation must be implemented in almost every smart card operating system due to the requirements of various ISO standards for data transmission using the T = 1 protocol. This algorithm can be used for other purposes without any additional effort.



**Figure 6.13** Computing and checking an XOR checksum

Unfortunately, XOR checksums also suffer from several serious drawbacks, which significantly limit their practical use. They are in principle not very secure. For example, they do not allow the interchange of two bytes in the data to be detected. Also, multiple errors can occur at the same position in several bytes and cancel each other out. The consequence of all this is that XOR checksums are mainly used for data transmission; they are used only to a very limited extent to verify the consistency of memory contents. XOR checksums can be computed very quickly, since specific machine instructions for this purpose are generally available. With a typical 8-bit smart card microcontroller clocked at 3.5 MHz, a rate of 1  $\mu$ s/byte can be achieved, which corresponds to a throughput of 1 MB/s.

### 6.5.2 CRC checksums

The cyclic redundancy check (CRC) method also comes from the field of data communication, but it is significantly better than the XOR method. CRC checksums are also only error detection codes; they cannot be used for error correction. The CRC method has been used for a long time in data transmission protocols such as Xmodem, Zmodem and Kermit, and a hardware implementation of the method is widely used in hard-disk drive controllers. It is based on the CCITT V.41 recommendation. An additional standard for CRC checksums is ISO/IEC 13 239.

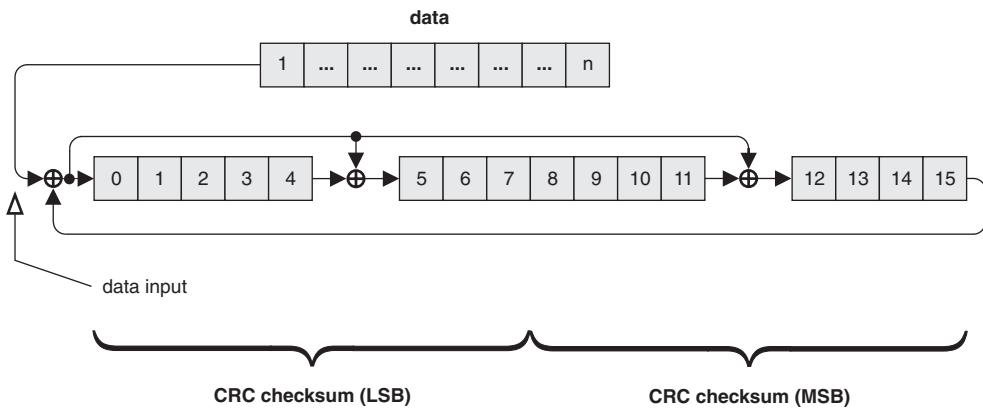
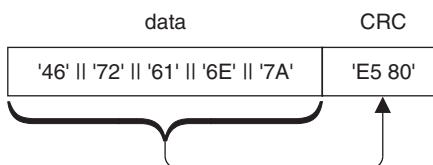
A CRC-16 checksum is generated by a 16-bit cyclic feedback shift register, while a 32-bit shift register is used to generate a CRC-32 checksum. The following description refers only to the CRC-16 checksum, since this is the most commonly used CRC method for smart cards. The feedback in the shift register is determined by a generating polynomial. In mathematical terms, the data to be checked is treated as a large number that is divided by the generating polynomial. The remainder of this division is the checksum. The CRC-16 method (that is, a 16-bit CRC) should only be used with data volumes up to 4 KB because the error detection probability drops sharply beyond this point. However, this restriction can easily be circumvented by dividing the data into blocks no larger than 4 KB. Alternatively, a CRC-32 method (32-bit CRC checksum) can be used, which allows single-bit errors to be detected in up to 4 GB of data.

With a CRC checksum, it is always necessary to know the generating polynomial as well as the initial value of the shift register, as otherwise the computation cannot be reproduced. The initial value of the shift register is zero in most cases (such as methods compliant with ISO 3309), but several data transmission protocols (such as CCITT Recommendation X.25) set all bits to 1. Some commonly used generator polynomials are listed in Table 6.8.

Computation of a CRC checksum, as illustrated in Figure 6.14 on the next page, proceeds as follows. First the 16-bit CRC register is set to its initial value. Then the data bits are fed into the

**Table 6.8** Commonly used generator polynomials for CRC-16 computation

Designation	Generator polynomial
CRC CCITT V.41, ISO/IEC 3309	$G(x) = x^{16} + x^{12} + x^5 + 1$
CRC-16 (CRC for USB data packets)	$G(x) = x^{16} + x^{15} + x^2 + 1$
CRC-12	$G(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1$
CRC-64	$G(x) = x^{64} + x^4 + x^3 + x + 1$
CRC for Bluetooth	$G(x) = x^5 + x^4 + x^2 + 1$
CRC for USB tokens	$G(x) = x^5 + x^2 + 1$

**Figure 6.14** Calculating a CRC-16 checksum with a generator polynomial  $G(x) = x^{16} + x^{12} + x^5 + 1$ . The data and the CRC register are both shown as bits**Figure 6.15** Operating principle and sample computation of a CRC checksum using the generator polynomial  $G(x) = x^{16} + x^{12} + x^5 + 1$  and an initial value of '0000'

feedback shift register one after the other, starting with the least significant bit. The feedback (which represents the polynomial division) is provided by bitwise logical XOR operations on the CRC bits. The computation is complete after all the data bits have been fed into the register, and the 16-bit value remaining in the register is the desired CRC checksum. Figure 6.15 shows an example of the result of CRC calculation with actual data.

Data with a CRC checksum is checked by recalculating the CRC checksum of the data and comparing the result with the checksum provided with the data. If they match, it follows that the data and the checksum have not changed.

The major advantage of CRC checksums is that they provide reliable error detection, even with multiple errors. In addition, in contrast to the XOR method CRC checksums allow

interchanged data bytes to be detected because the byte order is significant in the process of generating the checksum using a feedback shift register. However, it is very difficult to specify exact error detection probabilities because they are very dependent on the positions of the errors in the individual bytes. Here a comment based on experience is in order. Although the CRC algorithm is not especially complicated and can be clearly specified using only a few parameters, it is always a good idea to perform a sample computation before using the method. This is a simple way to test the implementation of the method.

The CRC algorithm is relatively simple, so the amount of code needed to implement it matches the capacities of small smart card memories. However, it has the drawback that the computation is relatively slow because the algorithm requires the data to be shifted bit by bit. The CRC checksum algorithm was originally designed for hardware implementation, which is a disadvantage when it is implemented in software. The throughput of a CRC-16 routine is lower than that of an XOR checksum routine by a factor of around 200. A typical figure is 0.2 ms/byte with a 3.5-MHz clock frequency, which corresponds to a throughput of 5000 byte/s. Computing the CRC checksum of the contents of the 10-KB memory of a smart card microcontroller would thus take around 2 s.

Some types of microcontrollers have integrated components for hardware-assisted generation of the CRC checksums of freely definable memory regions. The rates that can be achieved with this approach can be as high as one byte per clock cycle. With a microcontroller clock frequency of 5 MHz, this yields a throughput of 5 MB/s. At this rate, the CRC checksum of a 10-MB memory block could be calculated in 2 ms.

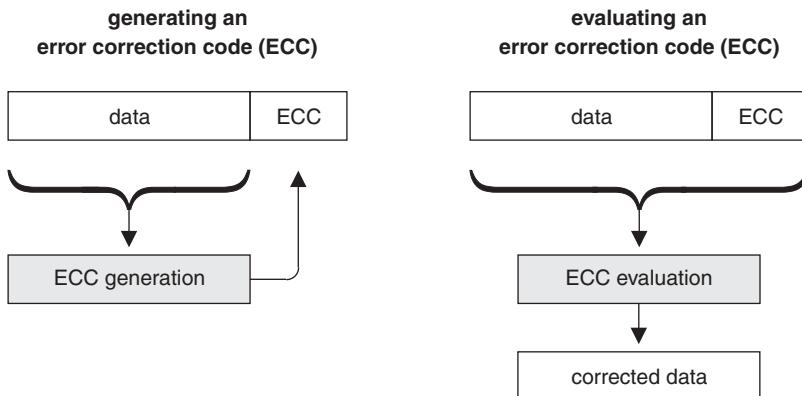
### 6.5.3 Reed–Solomon codes

In 1960, the mathematicians Irving S. Reed and Gustave Solomon published an article with the title ‘Polynomial Codes over Certain Finite Fields’, which forms the basis for what has become one of the most widely used error detection and correction methods. Reed–Solomon codes (RS codes), which are named after their inventors, are used for error detection and correction in data storage (such as barcodes, CDs and DVDs) and data transmission (such as DSL, satellite communication and space probes).

Reed–Solomon codes are block-oriented error correction codes (ECCs) that can also correct burst errors. The computation is based on the arithmetic of finite bodies (Galois fields, or GF). The characteristics of Reed–Solomon codes, in terms of the number of detectable and correctable errors for a specific data length, can be adapted to a particular application by selecting a suitable generator polynomial.

For some years, Reed–Solomon codes have been used in various smart card operating systems to detect errors in data stored in EEPROM or flash memory. The generator polynomials used for this purpose are matched to the properties of EEPROM and flash memory (occurrence of burst errors) to yield significantly more reliable error detection than what is possible with the CRC method.

For example, with a RS code based on a  $2^8$  GF and two supplementary bytes in addition to the data to be protected, it is possible to detect two incorrect bytes and correct one incorrect byte. With three supplementary bytes, three incorrect bytes can be detected and one incorrect byte can be corrected, and with four supplementary bytes, four incorrect bytes can be detected and two incorrect bytes can be corrected. The size of the program code in 8051 assembly language is approximately 100 bytes, and the computation speed is approximately 10 ms/byte



**Figure 6.16** Basic error correction code (ECC) processes. Here it must be noted that there is always a certain probability of erroneous data correction, depending on the ECC method that is used

at 3.57 MHz. RS coding can also be readily implemented in hardware, although this is not supported by commonly used smart card microcontrollers.

#### 6.5.4 Error correction codes

If it is necessary to not only detect changes in memory regions but also correct them in case of errors, error correction codes must be used. The basic principle of this is illustrated in Figure 6.16. Since computing such codes is costly in terms of program code and processing power, it is often difficult to use them to protect smart card memory. In addition, the algorithms concerned are usually designed to correct only low error rates. As EEPROM or flash memory in smart cards has a page-oriented structure and an entire page often fails in the event of an error, only methods that can correct burst errors are worthwhile. Consequently, other means are often used for error correction.

The technically simplest solution is to store the data in multiple, physically separate memory pages and to use a majority-vote procedure when reading the data. Triple storage is commonly used, together with a 2-of-3 vote. A less memory-intensive alternative is to store the data in two locations with EDC checksum protection for each location. The occurrence of a memory error can be detected by checking the two EDC values. This also allows the memory region where the error occurred to be identified. The region with no detected error must then contain the valid data, which can be restored to the faulty region.

Of course, a significant amount of extra memory is needed with these error correction methods, but for small data volumes it is still well within acceptable limits. The main advantage is that evaluation does not require a complicated, code-intensive algorithm. As an alternative to protection by multiple data storage, it is possible to use error correction algorithms such as Reed–Solomon. This is particularly suitable for handling burst errors, which sometimes occur in smart cards due to page faults in EEPROM or flash memory. The required storage space in program memory is a few hundred bytes if the algorithm is programmed in assembly language, and the volume of the ECC data depends primarily on the required probability of error detection and/or accurate correction.

Several basic remarks are in order here with regard to using error correction methods in smart cards. At first glance, it may seem tempting to use these methods to correct errors that occur in EEPROM or flash memory. However, the presumed data security comes at the price of several drawbacks. The necessary memory space is significant, and the time required to write data to memory increases considerably because the data must be stored in multiple locations. Algorithms that can correct burst errors on the scale typically seen with page-based EEPROM or flash memory are elaborate and require a certain amount of memory space for the EDCs. However, the fundamental drawback is even more severe. Even if error correction algorithms are used, errors can still be present in the corrected data because the algorithms work properly only if the number of errors remains below a certain level. If an operating system corrects memory errors automatically, in principle it is never possible to be certain that the correction was made properly.

For example, suppose that automatic error correction is applied to the balance in an electronic purse. The system operator can never be sure what will happen to the credited amount in the event of an error. The balance may be corrected properly, but there is a certain probability that it will be too high or too low after correction. In this regard, it should also be borne in mind that smart cards are inexpensive mass-produced articles that can simply be replaced if they malfunction.

If a problem occurs with the data contents, the solution is usually determined by a higher-level system that allows human intervention. For example, the first time an error occurs in a smart card purse, the cardholder's balance will doubtless be restored manually. However, if errors recur repeatedly the system operator will be much less forthcoming with respect to the cardholder because there is a possibility of attempted manipulation of the EEPROM or flash memory with fraudulent intent. This cannot be dealt with by error correction codes in the smart card, but instead must be handled by the actions of a system administrator.

In case of smart cards with flash memory capacities in the megabyte range,<sup>3</sup> a somewhat different approach is taken to error correction. Such cards have a flash translation layer<sup>4</sup> (FTL) between the memory hardware and the software functions that access the memory. The flash translation layer isolates memory blocks recognized as being defective and denies all access to them. This usually does not involve error correction, but only detection of memory errors.

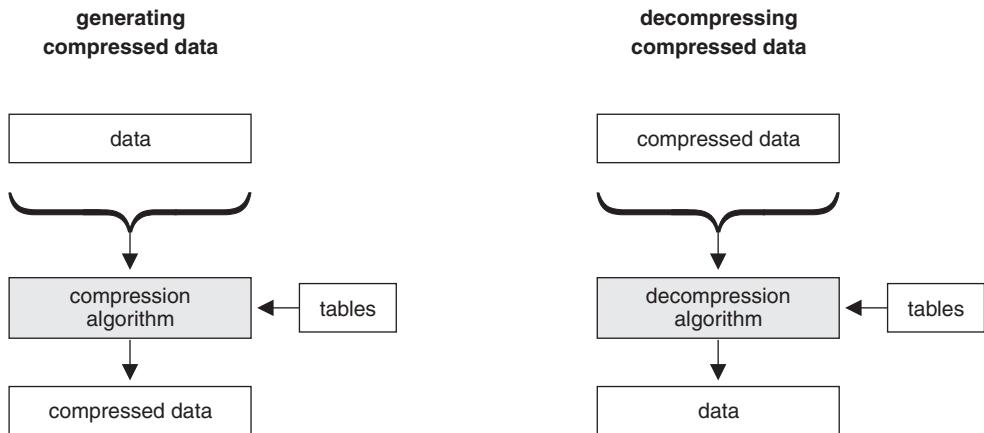
## 6.6 DATA COMPRESSION

As is well known, the amount of memory available in a smart card is severely limited. The basic processes of data compression and decompression are shown in Figure 6.17. This repeatedly leads to the desire to improve this situation by using data compression.

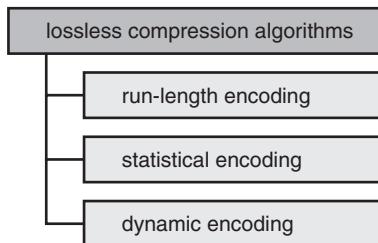
However, there are certain obstacles that must be overcome first. The algorithm must not take up too much program memory space, and above all it must require very little RAM. In addition, an acceptable compression speed should be achieved, although the throughput is not especially important if the data volume to be compressed is only a few hundred bytes. The document 'Lossless Compression Algorithms for Smart Cards: A Progress Report' by J.-F. Dhem *et al.* [Dhem 96] provides an overview that is still current.

<sup>3</sup> See also Section 2.3.5, 'Multi-megabyte cards', on page 24

<sup>4</sup> See also Section 13.6.3, 'Flash memory management', on page 461



**Figure 6.17** The basic processes of data compression and decompression



**Figure 6.18** Various lossless data compression methods

Only lossless compression methods (see Figure 6.18) can be used with smart cards, since the extracted data must always match the original data exactly. The methods that are used with smart cards are run-length encoding and variable-length encoding.

With run-length encoding, a contiguous string of repeated data items is replaced by the combination of a repeat count and the repeated data item (such as a character). With variable-length encoding, the frequency of occurrence of items with fixed lengths (such as one byte) is analyzed and the most frequently occurring items are replaced by codes with shorter lengths (Huffman algorithm). Less frequently occurring items are encoded using longer codes.

With static variable-length encoding, the replacements are made using a previously defined table. The dynamic version of variable-length encoding first analyzes the frequency distribution of the items in the original data and then constructs a replacement table based on the results of this analysis. A third variant is adaptive variable-length encoding, in which the replacement table is continuously updated during the compression process to achieve optimum compression.

Due to the complexity of their algorithms and their large RAM requirements, dynamic and adaptive variable-length encoding can only be used in exceptional cases with smart cards. Run-length encoding and static variable-length encoding are thus the only practical choices for implementation in smart cards. The algorithm for run-length encoding does not need much program code, but it has the drawback that it can only be used with repetitive data. Image data,

for example, is particularly suitable because images often contain large areas with the same value. Keys for symmetric cryptographic algorithms are entirely unsuitable for compression with this algorithm because they have the characteristics of random numbers.

Static variable-length encoding is the second compression method used with smart cards. It is quite suitable for data objects such as telephone directory files because the structure of the stored data is known and the replacement table can be permanently built into the algorithm. Telephone numbers contain only the numerals 0 through 9 and a few special characters, such as ‘#’ and ‘\*’. If the names are restricted to capital letters, the replacement table only has to accommodate the 26 characters of the alphabet. Furthermore, certain letters occur significantly less often in names than others, which also affects the encoding. With telephone directories, a memory space reduction of 30 % relative to the uncompressed data can certainly be achieved, although the memory occupied by the compression algorithm must also be taken into account.

In many cases it is possible to presort the data in order to improve the compression rate, which enables the compression algorithm to achieve a higher packing density. For example, a compression level of around 60 % can be achieved if the Huffman algorithm with presorting using the Burrow–Wheeler algorithm is used to compress 350 KB of ARM 7 code. The associated program code size is approximately 4 KB, but the algorithms also need around 10 KB of RAM for execution.

Certain things must be borne in mind with regard to data compression in smart cards. Ideally, data compression should be performed in the operating system in a manner that is fully transparent to the outside world, so that uncompressed data can be read and written in the usual way using standard commands. In addition, compression can only be applied to certain types of data. The results of attempting to compress program code and keys are usually unsatisfactory. This must be taken into account in the application design, as otherwise compression can actually cause the required memory space to increase instead of yielding the expected reduction.

For all these reasons, data compression has been used only sparingly in smart cards up to now. Compression algorithms are sometimes used in certain applications, such as telephone directories in SIM cards. With general-purpose operating systems and applications whose data structures are not known in advance, data compression usually does not yield satisfactory results.

# 7

# Security Foundations

Nearly half of this chapter is dedicated to the cryptographic methods used in the smart card environment. Until a few years ago, the subject of cryptography was surrounded by a veil of secrecy and ignorance. However, this situation has changed dramatically in recent years, and there is now extensive literature on this subject. Here we provide only the basic information necessary for understanding cryptographic algorithms and protocols. For more detailed information, we refer you to well-known books on this subject, such as those by Bruce Schneier [Schneier 96] and Alfred Menezes [Menezes 97]. A further rich source of information on cryptography is the World Wide Web, where you can find the sites of several research institutes (e.g. [Fraunhofer SIT]), standards organizations (e.g. [ETSI, IEC, ISO]), agencies (e.g. [BSI, NSA]), companies (e.g. [Certicom, Counterpane, RSA]), associations (e.g. [CCC, Teletrust]), and individuals with an interest in the subject (e.g. [Gutmann]). The Wikipedia [WikiEN] has also developed into a rather extensive and reliable source of information on many aspects of cryptography.

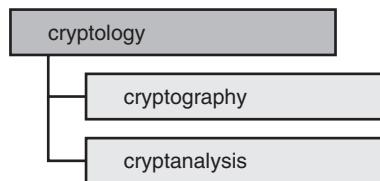
## 7.1 CRYPTOLOGY

In addition to serving as data storage media, smart cards are used as authorization media and encryption modules. Cryptography thus became one of the key aspects of smart cards at an early stage in their development. The methods and algorithms of this discipline are now firmly established components of smart card technology.

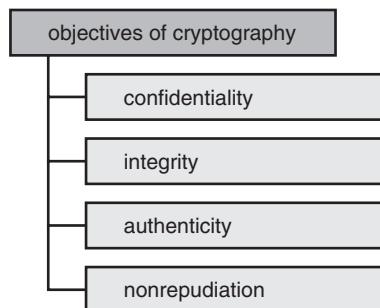
Cryptology can be divided into two fields: cryptography and cryptanalysis, as illustrated in Figure 7.1 on the following page. Cryptography is the study of methods for encrypting and decrypting data, while cryptanalysis focuses on attempts to break existing cryptographic systems.

In the smart card realm, the practical use of existing cryptographic methods and algorithms is the principal task and primary application of cryptography. Consequently, here we concentrate more on the practical aspects of cryptography than on the theoretical aspects. However, we do not entirely neglect issues related to using the algorithms or the basic aspects of the theoretical foundations.

As illustrated in Figure 7.2, the four objectives of cryptography are maintaining the secrecy or confidentiality of messages, ensuring the integrity and authenticity of messages, and



**Figure 7.1** The two principal fields of cryptology: cryptography and cryptanalysis



**Figure 7.2** The four separate objectives of cryptography

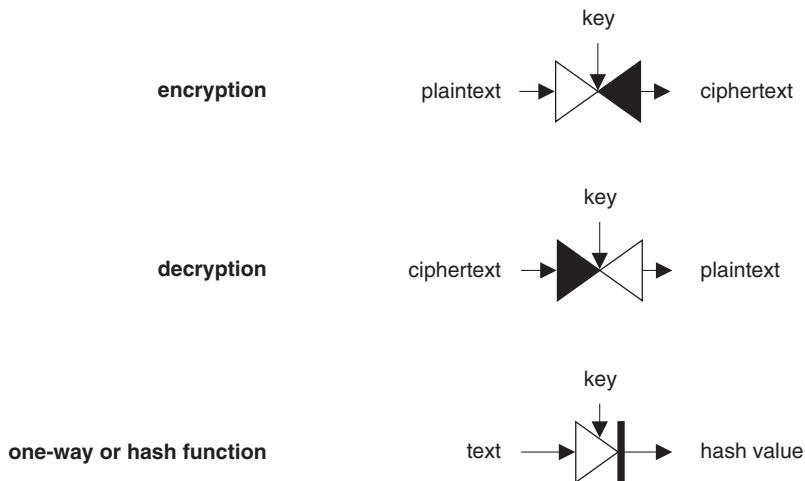
ensuring the binding force (nonrepudiation) of messages. These objectives are mutually independent, and they place different demands on the system concerned. Confidentiality means that only the intended recipient of a message can decrypt its contents. Authenticity means that the recipient can verify that the received message has not been altered during transmission. Nonrepudiation means that the sender can verify that a certain recipient has received a particular message, which means that the message has binding force.

The notation used in this book for cryptographic algorithms is described under ‘Symbols and Notation’ on page xxv and illustrated in Figures 7.3 and 7.4 on the facing page. The terms and principles described below form the basis for cryptology and are a prerequisite for understanding the algorithms described here.

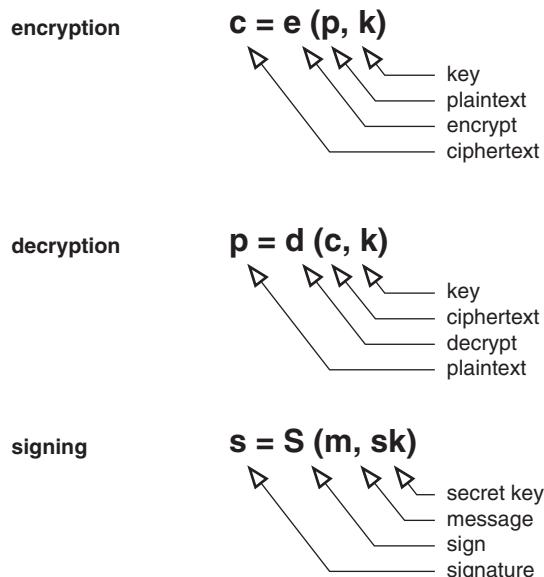
In simplified terms, encryption technology deals with three types of data. The first is unencrypted data, which is called plaintext. The opposite of this is encrypted data, which is called ciphertext. In addition, one or more keys are needed for encryption and decryption. These three types of data are processed by an encryption algorithm. The cryptographic algorithms currently used in smart cards (see Figure 7.5) are all block-oriented, which means that the plaintext and ciphertext can only be processed in fixed-length packets, such as eight bytes with the AES algorithm.

Modern cryptographic algorithms are generally based on Kerckhoff’s principle. This principle, which is named after Auguste Kerckhoff (1835–1903), says that the entire security of an algorithm should be based only on the confidentiality of the key, not on the confidentiality of the actual algorithm. The consequence of this generally known but often disregarded principle is that many algorithms used in the civil sector have been published, and some of them have been standardized.

The opposite of Kerckhoff’s principle is the principle of security by obscurity. With this principle, the security of a system is based on the idea that a would-be attacker does not

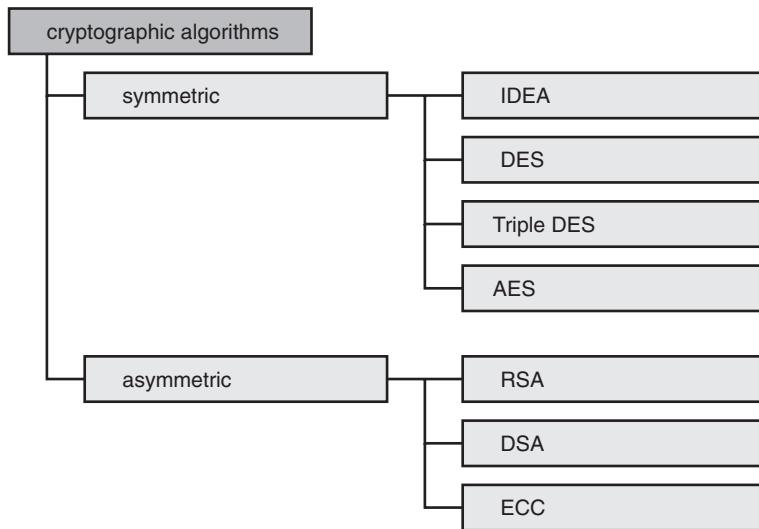


**Figure 7.3** The symbols used in this book for cryptographic algorithms



**Figure 7.4** The notation used in this book for cryptographic algorithms. The entire notation used in this book is described under ‘Symbols and Notation’ on page xxv

know how the system works. This principle is very old, and it is still frequently used even today. However, cryptographic systems (and other systems) should not be developed solely on the basis of this principle. Up to now, every system based on this principle alone has been broken, usually in a very short time. In our information society, it is generally not possible to keep the technical details of a system secret for a long time, which is precisely what this principle requires.



**Figure 7.5** Classification of cryptographic algorithms used in the smart card environment

Of course, the consequences of incidental interception of messages can certainly be limited by using obscurity. This principle is thus very often used in combination with Kerckhoff's principle. In many large systems, it is also included as a supplementary security level. As the security of modern, published cryptographic algorithms is primarily based on the limiting processing power of current computers, obscuring the method that is used increases the degree of protection against attacks.

Protection strategies based solely on the assumption that potential attackers do not have access to sufficient computing power may be quickly overtaken by the rapid pace of technological progress. Statements such as 'it would take a thousand years to break this cryptographic system' are unreliable because they are usually based on currently available computing power and algorithms. They cannot take future developments into account because such developments are generally unknown. The computing power of processors doubles approximately every eighteen months, with the result that the computing power of individual processors has increased by a factor of approximately 25 000 over the last 25 years.

The list of the top 500 supercomputers [Top500], which is updated several times a year, provides an overview of the computing power of the world's most powerful computers. In the spring of 2008, the top value with a single processor was 478.2 teraflops/s, while the collective capacity of the 500 fastest computers was around 7 petaflops/s. The upward trend has been essentially stable for many years, and it shows that the processing power of this sort of computer doubles every year, which is well above the rate given by Moore's Law.

In recent years, increased networking of computers has created another option for mounting serious attacks on keys and cryptographic systems. For instance, a request to help break a DSS key, if posted on the Internet, would reach millions of users as a result of the snowball effect. If only one out every thousand Internet users<sup>1</sup> agreed to participate in such a project, the

<sup>1</sup> According to [ISC], the estimated number of Internet users was approximately 550 million in the spring of 2008

potential attacker would have access to a massively parallel computer composed of 550 000 individual computers.

Cryptographic algorithms are classified into two types: symmetric and asymmetric. This classification is based on the key that is used. Symmetric algorithms use the same key for encryption and decryption, while asymmetric algorithms (which were first postulated in 1976 by Whitfield Diffie and Martin E. Hellman) use different keys for encryption and decryption.

A term that often arises in connection with cryptographic algorithms is the size of the key space. This refers to the number of possible keys that can be used with a particular cryptographic algorithm. A large key space is one of the essential features of a secure cryptographic algorithm.

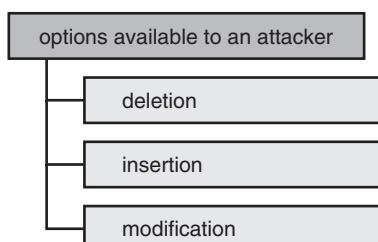
An implementation requirement smart card cryptographic algorithms that first became prominent in the mid-1990s is freedom from noise. In this context, it means that the processing time of the algorithm is independent of the key, the plaintext, and the ciphertext. If this requirement is not met, it may be possible to discover the secret key in a relatively short time, which would compromise the entire cryptographic system.

In cryptology, there is a strong distinction between the theoretical and practical security of a system or an algorithm. A system is theoretically secure if the system cannot be breached, even if the attacker has unlimited time and tools available. For example, this means that even if an attacker would need hundreds of years and several supercomputers to break a system, the system could not be considered theoretically secure. A system that cannot be broken if the attacker has only limited time and tools available is regarded as practically secure.

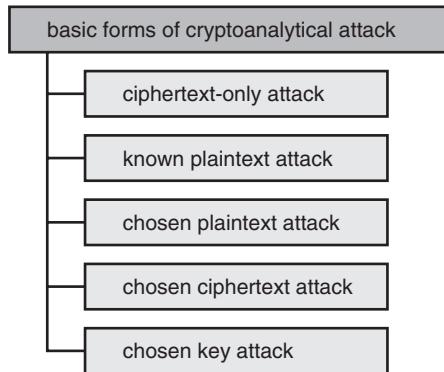
A cryptographic system can ensure the confidentiality and/or authenticity of messages. If the system is compromised, this confidentiality and/or authenticity is no longer ensured with respect to the attacker. If an attacker can discover the secret key of an encryption algorithm, he can decrypt data that has been protected by being encrypted, learn the content of this data, and modify it as desired.

Several basic approaches are available to attackers (see Figure 7.6), and various types of attack can be used to break the key of a cryptographic algorithm (see Figure 7.7). In a ciphertext-only attack, the attacker knows only the ciphertext and attempts to use this information to discover the key or the plaintext. A more promising approach is the known-plaintext attack, in which the attacker has access to several plaintext–ciphertext pairs associated with a secret key. Even more promising are the chosen-plaintext and chosen-ciphertext attacks, which are possible if the attacker is able to generate his own plaintext–ciphertext pairs. With these forms of attack, the secret key can be discovered experimentally.

Discovering a key by trial and error (brute-force attack) is naturally the least sophisticated form of attack. With this method, an attempt is made to discover the correct key by employing a large amount of computing power to test all possible keys with a known plaintext–ciphertext



**Figure 7.6** Various manipulation options available to attackers



**Figure 7.7** Basic types of cryptanalytical attack

pair. Obviously, computing power in the supercomputer range is normally required for this method. Statistically seen, on average half of the possible keys must be tested before the right one is found. Naturally, a large key space considerably increases the difficulty of this form of attack.

### 7.1.1 Symmetric cryptographic algorithms

As illustrated in Figure 7.8, symmetric cryptographic algorithms are based on the principle of performing encryption and decryption using the same secret key – hence the designation ‘symmetric’.

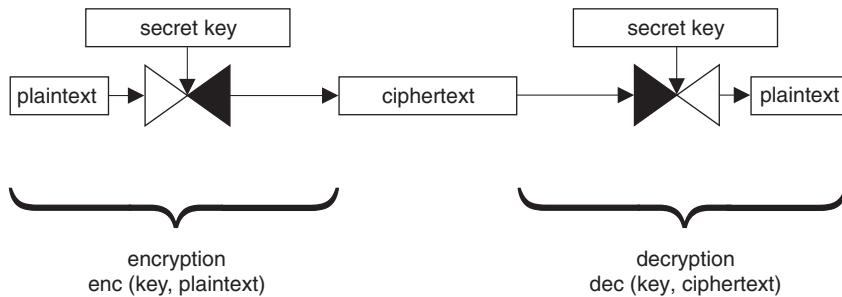
#### 7.1.1.1 DES algorithm

The best-known and most widely used symmetric cryptographic algorithm is the Data Encryption Algorithm (DEA). It was developed by IBM in collaboration with the US National Bureau of Standards (NBS) and published in 1977 as the FIPS 46 standard. The standard that describes the DEA is generally called the Data Encryption Standard (DES). Consequently, the Data Encryption Algorithm is commonly (but not entirely correctly) called the DES algorithm. Figure 7.9 shows the operating principle of the DES algorithm.

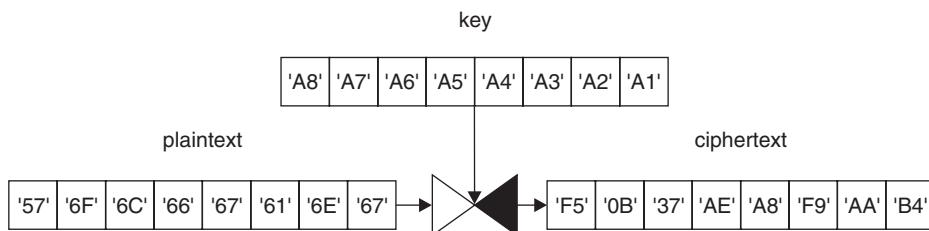
As this algorithm was fashioned in accordance with Kerckhoff’s principle, it could be published without impairing its security. However, even today not all of the development criteria have been published, which repeatedly leads to speculation on potential methods of attack and the possible presence of trap doors. However, up to now all attempts break the algorithm on this basis have failed.

Two important principles for good encryption algorithms were incorporated in the design of the DES algorithm: confusion and diffusion, as originally proposed by C. Shannon. The confusion principle states that the statistics of the ciphertext should affect the statistics of the plaintext in a manner that is so complex that an attacker can derive no advantage from them. The diffusion principle states that each bit of the plaintext and each bit of the key should affect as many bits of the ciphertext as possible.

DES is a symmetric block encryption algorithm that does not expand the ciphertext, which means that the plaintext and ciphertext blocks have the same size. The block size is 64 bits (8 bytes), which is also the key size, although only 56 of these bits are used as the actual key.



**Figure 7.8** Basic operating principle of a symmetric cryptographic algorithm for data encryption and decryption



**Figure 7.9** Operating principle of the DES algorithm for data encryption

The key includes eight parity bits, which reduce the available key space. The 64 bits of the key are numbered consecutively from left (msb) to right (lsb). Bits 8, 16, 24, ..., 64 are parity bits. The parity is always odd. Due to the eight parity bits, the key space is  $2^{56}$ , which means that there are approximately  $7.2 \times 10^{16}$  possible keys. At first glance, a key space with 72 057 594 037 927 936 possible keys may appear very large, but the limited size of its key space is actually the main weakness of the DES algorithm.<sup>2</sup> Given the processing capacity of modern computers and the worldwide networking of computers, a key space of this size is considered too small for a secure cryptographic algorithm. If the key space is too small, it is easy to test all possible keys if a suitable plaintext–ciphertext pair is available.

If a plaintext–ciphertext pair is obtained by tapping the communication between a terminal and a smart card, a brute force attack can be mounted by encrypting the plaintext using all possible keys. The correct key can be determined by comparing all of the resulting ciphertexts with the previously obtained ciphertext. This process can be performed very easily using multiple processors in parallel, with each processor testing only a small portion of the key space. In 1993, Michael Wiener published a design for a million-dollar computer that could test all the DES keys of a given plaintext–ciphertext pair in seven hours [Wiener 93]. In 1998, a similar concept called ‘DES Cracker’ was implemented by the Electronic Frontier Foundation (EFF) at a cost of approximately US\$ 250 000. The DES Cracker can test approximately 88.8 billion keys per second, and in the first public test it took 56 hours to discover an

<sup>2</sup> The following analogy may help convey the size of this large number. The mass of the earth is approximately  $5.974 \times 10^{27}$  g. Based on this value, the number of electrons, protons and neutrons that make up the earth can be taken to be around  $10^{52}$ . If each elementary particle could store one bit, at most  $10^{52}$  bits could be stored in a memory with the mass of the earth

**Table 7.1** Typical DES computation times with an 8-byte block

Implementation	Time and throughput
Smart card with 3.5-MHz clock, software implementation	17.0 ms (3.8 kbit/s)
Smart card with 3.5-MHz clock and DES processing unit	112 µs (571 kbit/s)
Smart card with 3.5-MHz clock and triple-DES processing unit	130 µs (492 kbit/s)
Smart card with 4.9-MHz clock, software implementation	12.0 ms (5.3 kbit/s)
Smart card with 4.9-MHz clock and DES processing unit	80 µs (800 kbit/s)
Smart card with 4.9-MHz clock and triple-DES processing unit	93 µs (688 kbit/s)
PC (Pentium, 200 MHz)	4 µs (16 Mbit/s)
PC (Pentium, 2 GHz)	800 ns (80 Mbit/s)
DES hardware component	64 ns (1 Gbit/s)

unknown key [EFF 98]. The main problem with using DES is that its key space has become too small over time. Consequently, modern applications use at least triple DES with three keys or the AES algorithm.

A full description of the exact implementation of the DES algorithm is far beyond the scope of this book, and it is not necessary for understanding the subject. If you are interested in detailed information, you can consult FIPS Publication 46, Carl Meyer [Meyer 82], or Bruce Schneier [Schneier 96]. However, one significant detail should be mentioned here. DES was designed as an encryption algorithm that can easily be implemented in hardware, and many smart card microcontrollers now have a DES hardware unit. However, if DES must be implemented in software in a smart card, it will occupy around 1 KB of assembly language code, even in a highly optimized form. The size of a DPA-resistant form<sup>3</sup> is approximately 2 KB. Its processing speed is consequently rather low.

Typical computation times for encryption and decryption using a smart card, with corresponding times using PC or a hardware component for comparison, are shown in Table 7.1. These numbers may vary depending on the implementation, and they represent only the pure processing time for DES encryption or decryption of an eight-byte block, assuming that all registers are already loaded.

Keys for the DES algorithm can be generated by a random number generator that produces an eight-byte random number, which is then checked against the four weak and twelve semi-weak keys. If the generated value does not match any of these easily broken keys, the parity bits are computed and the result is a DES key.

### 7.1.1.2 AES algorithm

By the late 1990s, the amount of computing power available due to technological progress and international networking of computers was sufficient to allow even ambitious private individuals with organizational ability to mount successful brute-force attacks on the DES algorithm. As a result, the viability of DES became so questionable that the relevant national authorities increasingly devoted their attention to specifying a new symmetric cryptographic

<sup>3</sup> See also Section 16.5.1, ‘Attacks on the hardware’, on page 684

algorithm as the official successor to DES. In light of the never fully extinguished doubts about the integrity of DES, arising from the fact that not all of the design criteria were made available for public inspection, in 1997 the US National Institute of Standards and Technology (NIST) requested the international cryptographic community to submit suggestions for a successor to DES, accompanied by all associated documentation, to the NIST for a sort of competitive evaluation. In 1998, the NIST announced that there were 15 candidates for the Advanced Encryption Standard (AES) and published all the submitted documents for review (with regard to information technology and cryptanalytic aspects) by the US National Security Agency (NSA), research institutes, experts, and interested parties. After detailed review of these algorithms and broad public discussion of the results, in 2000 the NIST announced the selected successor to the DES algorithm. It was developed by the Belgian cryptologists Joan Daemen and Vincent Rijmen, and it was already known as the Rijndael algorithm before the competition was announced.

AES is a symmetric block encryption algorithm with a block length of 128 bits (16 bytes) that can be used with three different key sizes: 128 bits (16 bytes), 192 bits (24 bytes), and 256 bits (32 bytes). It is thus called AES-128, AES-192 or AES-256, depending on the key size. AES is suitable for implementation in hardware, and it can also be implemented readily in software running on low-performance 8-bit processors or high-performance 16-bit and 32-bit processors. Furthermore, it is internationally license-free, and according to the official statement of the NIST its useful life is more than 20 years. AES is standardized by FIPS 197, which is available free of charge on the Internet [NIST].

The size of the key space of AES with a 128-bit key is  $2^{128}$  ( $\approx 3.4 \times 10^{38}$ ), which is a factor of  $4.7 \times 10^{21}$  larger than the key space of DES with a 56-bit key. This large key space with a key size of only 128 bits enormously increases the difficulty of brute-force attacks. A software implementation of AES in a DPA-resistant form in a smart card requires approximately 4 KB of program code, including tables. Table 7.2 lists some typical processing times for the AES algorithm with a 128-bit key.

### 7.1.1.3 IDEA algorithm

There are many other symmetric cryptographic algorithms besides DES. Here we describe only the International Data Encryption Algorithm (IDEA) as a representative example. It was developed by Xuejia Lai and James L. Massey and originally published in 1990 as the ‘Proposed Encryption Standard’ (PES). An improved version was published in 1991. For a short

**Table 7.2** Typical AES computation times with a 128-bit key and a 16-byte block

Implementation	Time and throughput
Smart card, 16-bit CPU, 4.9-MHz clock, software implementation; encryption using a 128-bit key	20 ms (6.4 kbit/s)
Smart card, 16-bit CPU, 4.9-MHz clock, software implementation; decryption using a 128-bit key	25 ms (5.1 kbit/s)
Smart card with 3.5-MHz clock, software implementation	12.3 ms (5.2 kbit/s)
Smart card with 4.9-MHz clock, software implementation	8.8 ms (7.3 kbit/s)
PC (Pentium 4, 3.5 GHz)	160 ns (400 Mbit/s)
AES hardware component	1.9 ns (34 Gbit/s)

**Table 7.3** Typical IDEA computation times with an 8-byte block

Implementation	Time and throughput
Smart card with 3.5-MHz clock, software implementation	12.3 ms (5.2 kbit/s)
Smart card with 4.9-MHz clock, software implementation	8.8 ms (7.2 kbit/s)
PC (80386, 33 MHz)	70 µs (914 kbit/s)
PC (Pentium Pro, 180 MHz)	4 µs (16 Mbit/s)
IDEA hardware component	370 ns (173 Mbit/s)

time, the improved version was called the ‘Improved Proposed Encryption Standard’ (IPES), but now it is commonly known as IDEA. The development criteria and internal structure of this algorithm have been fully published, so Kerckhoff’s principle is satisfied. However, IDEA is a patented algorithm, which was formerly also the case with the RSA algorithm.

Like DES, IDEA is a block-oriented cryptographic algorithm that uses eight-byte plaintext and ciphertext blocks. In contrast to the DES algorithm, the key size is 16 bytes ( $2 \times 8$  bytes), which provides a significantly larger key space of  $2^{128}$  ( $\approx 3.4 \times 10^{38}$ ). Table 7.3 lists some typical processing times for the IDEA algorithm with an 8-byte block size.

IDEA is structurally compatible with DES except for its larger key size. It is also compatible with triple-DES systems, which use two 56-bit keys, so it can be used in place of triple DES without any impact on key size or the size of the input and output data blocks. Of course, compatibility in this regard does not mean that DES-encrypted data can be decrypted using IDEA.

There are only a few smart card implementations of IDEA. The amount of memory space needed for the program code is around 1000 bytes. Typical computation times for encryption and decryption are somewhat less than with DES.

#### 7.1.1.4 COMP128 algorithms

The originally confidential COMP128 algorithm was broken after details of its structure became known in 1998. COMP128 works with nine rounds and has a hash function in its kernel that generates a 128-bit hash value from a 256-bit input value. After the successful attack in 1998, the original algorithm was designated COMP128-1 to distinguish it from its successors. The COMP128-2 and COMP128-3 algorithms eliminate the weaknesses of the COMP128-1 algorithm and are its successors. Another successor is the Milenage algorithm, also sometimes called COMP128-4, which is based on AES.

#### 7.1.1.5 Milenage algorithm

The public Milenage algorithm is a symmetric cryptographic algorithm for the USIM cryptographic functions f1 to f5. Its kernel uses AES and is standardized in TS 35.206. The Milenage algorithm is the official example algorithm of the 3GPP project. However, it is also used by many network operators in real-time operation.

#### 7.1.1.6 Operating modes of block encryption algorithms

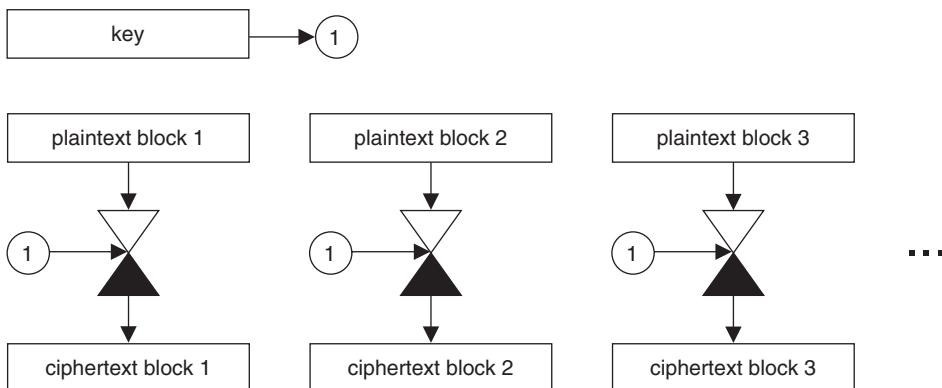
Every block encryption algorithm, including DES, can be used in four different operating modes, which are standardized in ISO 8372. Two of these modes, CFB and OFB, are especially

suitable for text strings with no block structure. The other two modes, ECB and CBC, are based on eight-byte blocks. These two block modes are used extensively in smart card applications.

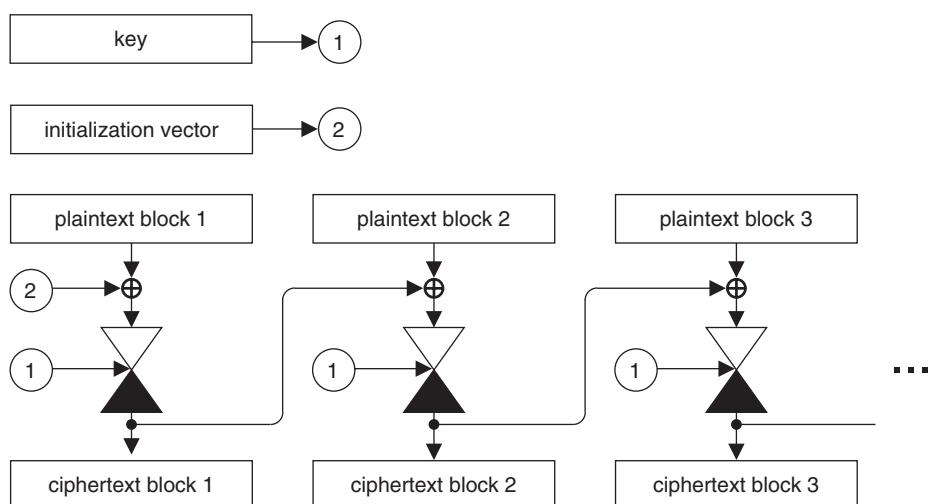
The basic operating mode of the DES algorithm is called ‘electronic code book’ (ECB). In this mode, eight-byte plaintext blocks are encrypted separately using the same key, as illustrated in Figure 7.10. This is the fundamental form of the DES algorithm without extensions.

The second block mode is called cipher block chaining (CBC). In this mode, the blocks of a multiblock data string are chained during encryption by using XOR operations as illustrated in Figure 7.11, with the result that each block is dependent on the preceding block. This enables reliable detection of attempts to exchange, add or delete encrypted blocks, which is not possible with ECB mode.

With suitably structured plaintext blocks (with affixed send sequence counters or send sequence counters in their initialization vectors), this chaining can also map identical plaintext



**Figure 7.10** Data encryption with a block encryption algorithm operating in ECB mode. Decryption is performed in a similar manner



**Figure 7.11** Data encryption with a block encryption algorithm operating in CBC mode. Decryption is performed in a similar manner

blocks into nonidentical ciphertext blocks. This makes cryptanalysis of intercepted data much more difficult, since codebook analysis (for example) cannot be used.

The first plaintext block is XORed with an initialization vector (IV) and then encrypted using the DES algorithm. The result is the ciphertext, which is in turn XORed with the next plaintext block. All subsequent blocks are processed in the same way.

As a rule, the initialization vector is preset to null. However, in some systems a session-specific random number is written to the initialization vector as a substitute for a temporary key. Of course, this number must be known for subsequent decryption.

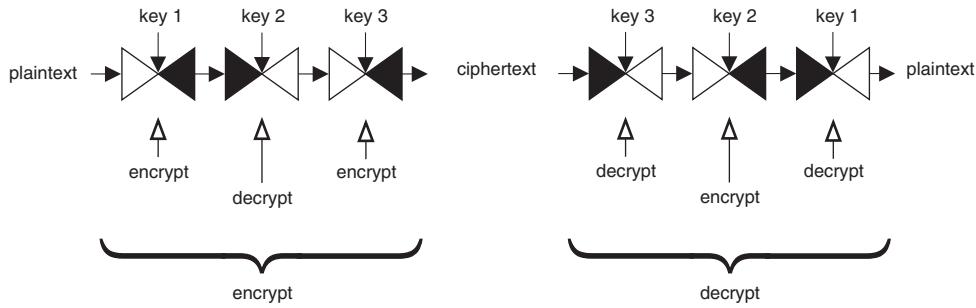
#### 7.1.1.7 *Multiple encryption*

Along with the four operating modes of block encryption algorithms, there is another technique that is used for increased security. In practice, it is used only with the DES algorithm due to the small key space of this algorithm. However, in principle it can be used with any block encryption algorithm that is not a group. If an encryption algorithm has the group property, double encryption with two different keys does not increase the cryptographic security because the same result could be obtained by single encryption with a third key. This means that double encryption using an algorithm with the group property does not enlarge the key space, since an attacker would only have to discover the third key in order to obtain the result previously obtained by double encryption with two different keys.

As DES does not have the group property, in theory double encryption using two different keys can be performed with no risk that the same result could be obtained by single encryption using a third key. However, in 1981 Ralph C. Merkle and Martin E. Hellman published an attack method called ‘meet-in-the-middle’ [Merkle 81], which is very effective against every form of double encryption using a block encryption algorithm. It requires prior knowledge of one or more plaintext–ciphertext pairs. The operating principle of this attack is based on encrypting the known plaintext using every possible version of one key and decrypting the known result (the ciphertext) using every possible version of another key. The set of results from the first process is then compared with the set of results from the second process. If a match can be found, there is a certain probability that the two keys have been discovered. The degree of confidence that the correct keys have been found can be increased by performing the same process with additional known plaintext–ciphertext pairs. As can be seen, this form of attack does not require significantly more effort than a conventional attack that involves searching the entire key space. Consequently, cascaded double encryption is not used with the DES algorithm.

The method that is used instead is called triple DES. As shown in Figure 7.12, it consists of a sequence of three CBC-mode DES operations with alternating encryption and decryption. Blocks that have been encrypted in this manner are decrypted by reversing the order of the operations (decryption, encryption, and then decryption). If all three keys are the same, the result of alternating encryption and decryption with this method is the same as the result of single encryption. This is the reason for not using a sequence of three encryption operations.

If the three DES operations with the three keys are performed directly on each plaintext block in sequence, the process is called DES in inner CBC mode. If instead the plaintext is first fully encrypted using the first key and the result is further processed in the described manner, the process is called DES in outer CBC mode. The outer CBC mode is more resistant to attack and is therefore generally recommended [Schneier 96].



**Figure 7.12** Basic data encryption process using triple DES in outer CBC mode. Key 1 is usually the same as key 3, so the effective key size is 112 bits ( $2 \times 56$  bits). Less commonly, three independent keys are used to yield a key size of 168 bits ( $3 \times 56$  bits)

**Table 7.4** Input and output parameters of symmetric cryptographic algorithms used in smart cards

Name	Plaintext size	Ciphertext size	Key size
DES	8 bytes	8 bytes	56 bits
Triple DES with two keys	8 bytes	8 bytes	112 bits ( $2 \times 56$ bits)
Triple DES with three keys	8 bytes	8 bytes	168 bits ( $3 \times 56$ bits)
IDEA	8 bytes	8 bytes	128 bits
AES	16 bytes	16 bytes	128 bits (16 bytes) 192 bits (24 bytes) 256 bits (32 bytes)

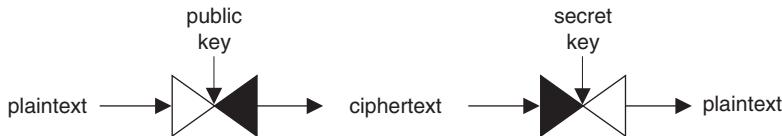
The triple-DES algorithm is known by several other names, including TDES, DES-3, 3-DES, and 2-DES. These names express the fact that three 56-bit keys are used, although the first and third keys are often the same. Consequently, the key size must always be stated in connection with triple DES in order to specify the algorithm unambiguously. This also applies to the frequently used designations 2TDES and 3TDES.

The triple-DES method is significantly more secure than sequential double encryption using two different keys because the meet-in-the-middle attack is not effective against this method. It requires three 56-bit keys instead of only one, which yields a key size of  $2 \times 56$  bits or  $3 \times 56$  bits. This method is thus data-compatible with the normal DES algorithm and incurs no additional cost other than the double or triple key size. This compatibility is one of the main reasons for the widespread use of triple DES in smart cards.

The main characteristics of symmetric cryptographic algorithms used in smart cards are summarized in Table 7.4.

## 7.1.2 Asymmetric cryptographic algorithms

In 1976, Whitfield Diffie and Martin E. Hellman described the possibility of developing an encryption algorithm based on two different keys [Diffie 76]. One of these keys was to be public, the other private (confidential). This would allow anyone possessing the public key to encrypt messages, while only persons possessing the private key could decrypt them.



**Figure 7.13** Encryption and decryption using an asymmetric cryptographic algorithm

This would eliminate the problems associated with exchanging and distributing confidential symmetric keys, and for the first time it would enable certain other processes, such as generating digital signatures that can be verified by everyone. Figure 7.13 depicts the basic processes of encryption and decryption with an asymmetric algorithm.

#### 7.1.2.1 RSA algorithm

Two years later, Ronald L. Rivest, Adi Shamir and Leonard Adleman [Rivest 78] proposed an algorithm that satisfied the criteria stipulated by Diffie and Hellman. This algorithm, which is called RSA after the initials of its inventors, is the best known and most versatile asymmetric encryption algorithm presently in use. Its very simple operating principle is based on the arithmetic of large integers. The two keys are generated from two large prime numbers.<sup>4</sup>

Encryption and decryption using the RSA algorithm can be expressed mathematically as follows, where  $x$  is the plaintext,  $y$  is the ciphertext,  $e$  is the public key,  $d$  is the private key,  $n$  is the public modulus, and  $p$  and  $q$  are secret prime numbers:

$$\begin{aligned} \text{encryption: } & y = x^e \bmod n \\ \text{decryption: } & x = y^d \bmod n \\ \text{where } & n = p \cdot q \end{aligned}$$

The plaintext block must be padded to the appropriate block size before encryption. The block size of the RSA algorithm can vary depending on the key size. Encryption is performed by exponentiation of the plaintext followed by a modulo operation. The result of this process is the ciphertext, which can only be decrypted if the private key is known, using a process similar to the encryption process.

The security of the algorithm is based on the difficulty of factoring large numbers. It is quite easy to compute the public modulus from the two prime numbers by multiplication, but it is very difficult to decompose the modulus into its two prime factors because there are no efficient algorithms for this process.

The RAM capacity of smart cards is not sufficient for performing exponential operations on large numbers during encryption or decryption, since the numbers become very large before the

<sup>4</sup> Whitfield Diffie and Martin E. Hellman invented the principle of public-key algorithms, while Ronald L. Rivest, Adi Shamir and Leonard Adleman developed the first implementation. However, public-key algorithms had already been discovered several years earlier in the field of military cryptography. In the period 1969–1973, James Ellis, Clifford Cocks and Malcom Williamson, working at British Government Communication Headquarters (GCHQ), developed the principle of public-key algorithms and the asymmetric encryption algorithm now known as the RSA algorithm, several years before they were invented by civilians. Unfortunately, these cryptologists were obliged to maintain confidentiality and were not allowed to publish anything about their work until after 1997 [Levy 99].

**Table 7.5** Typical public keys for the RSA algorithm

Public key (decimal and binary)	Remarks
$2 = 10$	The only even prime number; used with the Rabin algorithm
$3 = 2^1 + 1 = 11$	The smallest odd prime number
$17 = 2^4 + 1 = 1\ 0001$	—
$65\ 537 = 2^{16} + 1 = 1\ 0000\ 0000\ 0000\ 0001$	The fourth Fermat number

modulo operation. Consequently, modular exponentiation is used to prevent the intermediate results of the calculations from exceeding the value of the modulus. For example, if the value of  $x^2 \bmod n$  must be calculated, the expression  $(x \cdot x) \bmod n$  is not evaluated directly because the size of the intermediate result  $(x \cdot x)$  before reduction by the modulo operation would be too large. Instead,  $((x \bmod n) \cdot (x \bmod n)) \bmod n$  is calculated, which yields the same mathematical result. The advantage of this is that it requires significantly fewer calculations and less memory because the intermediate results are immediately reduced in size.

Another technique that dramatically accelerates the RSA algorithm is to use the Chinese remainder theorem for the calculations.<sup>5</sup> However, a prerequisite for using the Chinese remainder theorem is that both secret prime numbers  $p$  and  $q$  are known, which means it can only be used for decryption (e.g. for signing).

To increase the difficulty of attacks, the private key should be as long as possible. The public and private keys may have different lengths, and in fact this is usually true because the time required to verify a digital signature can be reduced considerably by making the public key as short as possible. The fourth Fermat number is often used as a public key. This prime number has the value of 65 537 ( $2^{16} + 1$ ), and due to its small size it suitable for quick verification of digital signatures. The numbers 3 and 17 are likewise used. Table 7.5 lists some typical public keys used with the RSA algorithm.

If an attacker manages to factor the public modulus into its two prime components, he can replicate the entire key calculation. Factoring is very easy with a small value such as 33, but there are presently no fast algorithms for factoring large numbers. If both prime factors can be found, the system is broken because the private key is known.<sup>6</sup>

Consequently, a requirement for RSA keys is that they are sufficiently long. A key size of 1024 bits (128 bytes) is currently considered the lower limit. However, 2048-bit keys (256 bytes) are also used. In the coming years, the first 4 096-bit (512-byte) keys will come into use. The computational effort needed for encryption and decryption is proportional to the key size. This proportionality is not linear, but instead approximately exponential. As can be seen from Table 7.6 on the following page, even with a 512-bit key the number of possible prime numbers is so large that collisions between two different key pairs will never occur.

Smart card microcontrollers, some of which still have eight-bit CPUs, usually cannot perform RSA computations within a few minutes. However, there are now microcontrollers

<sup>5</sup> See [Simmons 92] and [Schneier 96] for more information on these two techniques

<sup>6</sup> In the summer of 1994, a 426-bit key was broken in eight months using approximately 1600 computers networked via the Internet. The combined computing effort was 5000 MIPS-years. For comparison, a 100-MHz Pentium processor has a capacity of 50 MIPS. In 1999, a 512-bit key was factored in seven months using 292 networked computers

**Table 7.6** Typical RSA key sizes and characteristic values. Here NS/PN is the ratio of the number of nonprime numbers to the number of prime numbers. The reciprocal of this ratio is the probability that a randomly selected number in the number space is a prime number. This is very important with regard to the time required to generate an RSA key

Key size	Maximum number of digits	Size of the number space (NS)	Number of prime numbers in the number space (PN)	NS/PN
8 bits = 1 byte	3	256	54	≈ 4.7
40 bits = 5 bytes	13	≈ $1.1 \times 10^{12}$	≈ $3.9 \times 10^{10}$	≈ 28
512 bits = 64 bytes	155	≈ $1.3 \times 10^{154}$	≈ $3.8 \times 10^{151}$	≈ 342
768 bits = 96 bytes	232	≈ $1.6 \times 10^{231}$	≈ $2.9 \times 10^{228}$	≈ 552
1024 bits = 128 bytes	309	≈ $1.8 \times 10^{308}$	≈ $2.5 \times 10^{305}$	≈ 720
2048 bits = 256 bytes	617	≈ $3.2 \times 10^{616}$	≈ $2.3 \times 10^{613}$	≈ 1 391
4096 bits = 512 bytes	1 234	≈ $1.0 \times 10^{1233}$	≈ $2.5 \times 10^{12290}$	≈ 4 000

with numerical coprocessors specifically designed to perform fast exponentiation. With such coprocessors, it is possible to perform RSA computations in an acceptable length of time with reasonable software complexity. The code size of a hardware-assisted 1024-bit RSA algorithm in a smart card is around 1 kilobyte in assembly language. Typical processing times for the RSA algorithm are shown in Table 7.7.

One of the strengths of the RSA algorithm is that it is not restricted to a particular key size, unlike algorithms such as DES. If increased security is needed, longer keys can be used without modifying the algorithm. The RSA algorithm is thus scalable. However, the computation time and memory space requirements must be kept in mind, since a key size of 1024 bits is presently regarded as secure. With currently available factoring algorithms, a reasonable rule of thumb is that increasing the key size by 15 bits doubles the computational effort of factoring.<sup>7</sup>

Although the RSA algorithm is very secure, it is rarely used for data encryption due to its long computation time. It is primarily used for digital signatures, where the advantages of an asymmetric algorithm can be fully exploited. The greatest drawback of the RSA algorithm for use in smart cards is the amount of memory required for the key. The complexity of the key generation process also causes problems in certain cases.

### 7.1.2.2 Generating RSA keys

Keys for the RSA algorithm are generated using a simple procedure, which is illustrated here by a simple example.

1. First, select two prime numbers  $p$  and  $q$ :  $p = 3, q = 11$
2. Next, calculate the public modulus:  $n = p \cdot q = 33$
3. Calculate a temporary variable  $z$  for use in key generation:  $z = (p - 1) \cdot (q - 1)$

<sup>7</sup> The largest known prime number in the spring of 2008 was discovered in September 2006. It has 9808 358 digits and a value of  $2^{32\,582\,657} - 1$

**Table 7.7** Example computation times for RSA encryption and decryption as a function of key size. Some of these values may vary considerably because they are strongly dependent on the microcomputer used, the bit structure of the key, and use of the Chinese remainder algorithm (which can only be used for signing)

Implementation	Mode	512 bits	768 bits	1024 bits	2048 bits
Smart card without NPU, 8-bit CPU, 3.5-MHz clock	signing	20 min	—	—	—
Smart card without NPU, 8-bit CPU, 3.5-MHz clock, with Chinese remainder theorem	signing	6 min	—	—	—
Smart card, with NPU and 3.5-MHz clock	signing	308 ms	910 ms	2000 ms	—
Smart card with NPU and 3.5-MHz clock, with Chinese remainder theorem	signing	84 ms	259 ms	560 ms	—
Smart card with NPU and 4.9-MHz clock	signing	220 ms	650 ms	1400 ms	65 ms
Smart card with NPU and 3.5-MHz clock	verifying	1.04 s	—	—	—
Smart card with NPU and 4.9-MHz clock, with Chinese remainder theorem	signing	60 ms	185 ms	400 ms	—
Smart card with NPU and PLL, with Chinese remainder theorem	signing	—	—	—	40 ms
Smart card with NPU and PLL	verifying	—	—	—	840 ms
PC (Pentium, 200 MHz clock)	signing	12 ms	46 ms	60 ms	390 ms
PC (Pentium, 200 MHz clock)	verifying	2 ms	4 ms	6 ms	20 ms

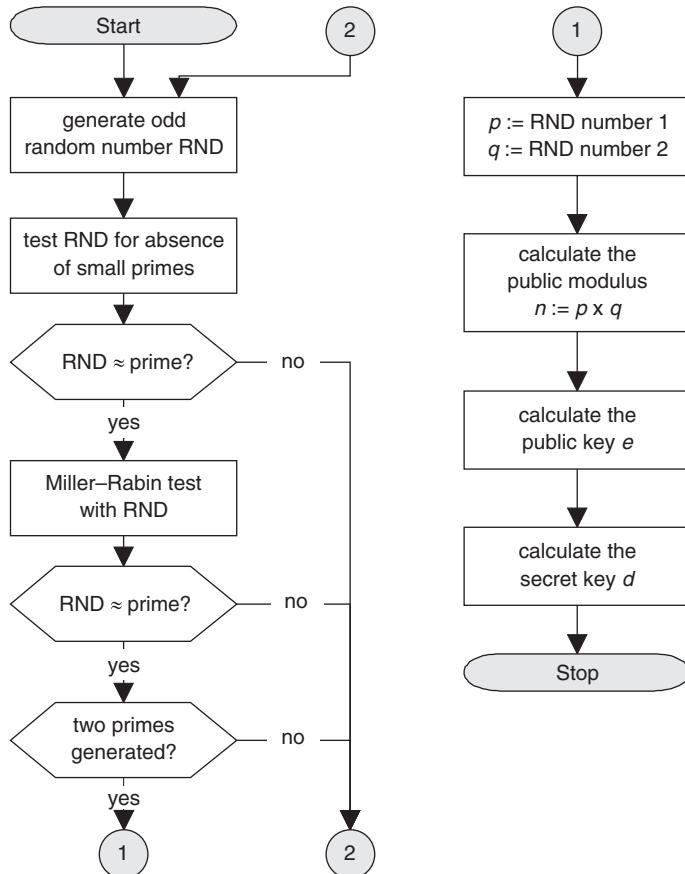
4. Calculate a public key  $e$  that satisfies the conditions  $e < z$  and  $\gcd(z, e) = 1$  (the greatest common denominator of  $z$  and  $e$  is 1). As there are several numbers that satisfy these conditions, select one of them:  $e = 7$
5. Calculate a private key  $d$  that satisfies the condition  $(d \cdot e) \bmod z = 1$ :  $d = 3$

This completes the computation of the keys. The public and private keys can now be tested for encryption and decryption using the RSA algorithm, as illustrated in the following numeric example:

1. Use the number 4 as the plaintext  $x$  ( $x < n$ ):  $x = 4$
2. Encryption:  $y = 4^7 \bmod 33 = 16$
3. The result of the calculation is the ciphertext  $y$ :  $y = 16$
4. Decryption:  $x = 16^3 \bmod 33 = 4$

As expected, the result of decrypting the ciphertext is the original plaintext.

In practice, the key generation process is somewhat more complex because it is rather difficult to check whether large numbers are prime. The well-known sieve of Eratosthenes cannot be used for this purpose because it requires knowledge of all prime numbers smaller than



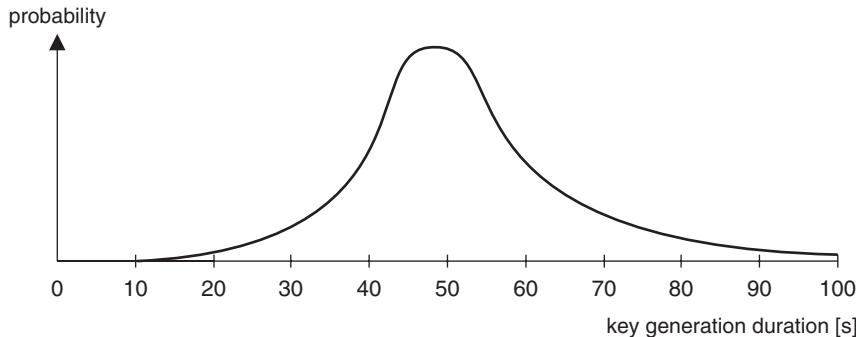
**Figure 7.14** Basic procedure for generating RSA keys for use in smart cards

the one being checked. This is practically impossible with numbers in the range of 1024 bits. Consequently, probabilistic tests are used to determine the likelihood that the selected number is a prime number. The Miller–Rabin test and the Solovay–Strassen test are typical examples of such tests.<sup>8</sup> To avoid having to use these time-consuming tests more than necessary, randomly generated candidate numbers are first tested to see if they have any small prime factors. If the randomly generated number can be exactly divided by a small prime number, such as 2, 3, 5 or 7, it obviously cannot be a prime number. Once it has been determined that the number to be tested does not have any small prime factors, a prime number test such as Miller–Rabin can be used. The operating principle of this test is illustrated in Figure 7.14 and described in detail in the appendix of the IEEE 1363 standard.<sup>9</sup>

The algorithms used to generate RSA keys have a special feature, which is that the time required to generate a key pair (a public key together with a private key) has a statistical distribution, as illustrated in Figure 7.15. This means that it is only possible to say that there

<sup>8</sup> The procedure and the algorithm are described by Alfred Menezes [Menezes 97]

<sup>9</sup> Many tips and considerations regarding the generation of prime numbers can be found in an article by Robert Silverman [Silverman 97]



**Figure 7.15** Typical time distribution of a probabilistic algorithm for generating key pairs for the RSA algorithm. The vertical axis shows the probability that a given length of time will be required to generate a 1024-bit key in a smart card. In this example, the average key generation time is 50 s. The area under the curve has a probability of 1

**Table 7.8** Typical times required to generate public–private key pairs for the asymmetric RSA cryptographic algorithm. For comparison, a PC with a 2-GHz CPU is approximately 7 times faster than the stated values

Key pair generation	Average time (s)	95% upper limit (s)	99% upper limit (s)
Smart card, 1024 bits	1.4	4.5	9.7
Smart card, 1536 bits	2.1	9.4	20.7
Smart card, 2048 bits	2.7	12.7	27.6

is a certain probability that key generation will take a given amount of time. A definitive statement such as ‘... will take  $x$  seconds’ is not possible, due to the need to perform the nondeterministic prime number test on the random numbers. It is common practice to state three times: the average key generation time, the time required to generate 95% of all key pairs, and the time required to generate 99% of all key pairs. Some typical processing times for generating key pairs are listed in Table 7.8.

### 7.1.2.3 DSS algorithm

In mid-1991, the US National Institute of Standards and Technology (NIST) published a draft version of a cryptographic algorithm for signing messages. This algorithm, which is now specified by the FIPS 186 standard, is called the Digital Signature Algorithm (DSA), and the standard that defines it is called the Digital Signature Standard (DSS). DSA and RSA are the two most widely used algorithms for generating digital signatures. The DSA algorithm is a modification of the El Gamal algorithm.

The standardization of this algorithm arose from the desire for an algorithm that could be used to generate signatures but not to encrypt data. For this reason, the DSA algorithm is more complicated than the RSA algorithm. However, a way to encrypt data using this algorithm was discovered fairly quickly [Simmons 93].

Unlike the RSA algorithm, the security of the DSS algorithm is not based on the difficulty of factoring large numbers, but instead on the discrete logarithm problem. The expression  $y = ax \bmod p$  can be computed quickly, even with large numbers. However, the inverse

process (calculating  $x$  for given values of  $y$ ,  $p$  and  $t$ ) requires a very large amount of computational effort.

With all signature algorithms, the message to be signed must first be reduced to a predefined length by using a hash algorithm. The NIST therefore published a suitable algorithm for use with the DSS algorithm. It is called SHA-1 (secure hash algorithm).<sup>10</sup> This algorithm is derived from the MD5 hash algorithm and generates a 160-bit hash value from a message of any desired length.

In practice, the RSA algorithm is used on a much broader scale than the DSS algorithm, which up to now has seen only very limited use. The original objective of defining a signature algorithm that cannot be used for encryption, which was the reason for standardizing the DSS algorithm, has failed to materialize. The complexity of this algorithm also discourages its widespread use. Nonetheless, for many official bodies the fact that the standard exists and political pressure to generate signatures using the DSS and SHA act to encourage its use.

#### **7.1.2.4 Elliptic curves as asymmetric cryptographic algorithms**

In addition to the two well-known asymmetric cryptographic algorithms (RSA and DSA), there is a third type of cryptography that is used for digital signatures and key exchange in the smart card environment. It is based on elliptic curves (ECs).

In 1985, Victor Miller and Neal Koblitz independently proposed the use of elliptic curves for constructing asymmetric cryptographic algorithms. The properties of elliptic curves are well suited to this purpose, and practical cryptographic systems based on these proposals were developed in the subsequent years. They are generally called elliptic curve crypto-systems (ECCs).

Elliptic curves are continuous planar curves that satisfy the equation  $y^2 = x^3 + ax + b$  in a finite three-dimensional space. No point on the curve is allowed to be a singularity, which for example means that  $4a^2 + 27b^2 \neq 0$ . The finite bodies  $GF(p)$ ,  $GF(2^n)$  and  $GF(p^n)$  are used in cryptography, where  $p$  is a prime number and  $n$  is a positive integer greater than 1.

The mathematics of cryptographic systems based on elliptic curves are relatively difficult. For this reason, we refer readers to a book on this subject by Alfred Menezes [Menezes 93]. The comprehensive IEEE 1363 public key cryptography standard and the ISO/IEC 15946 family of standards for elliptic curves also provide good overviews of elliptic curves and other asymmetric cryptographic algorithms.

The main advantages of asymmetric cryptographic systems based on elliptic curves are that they require much less processing power than systems such as RSA and that the same degree of cryptographic strength can be attained with significantly shorter keys. For example, roughly the same amount of computation is required to break an ECC algorithm with a 160-bit key as an RSA algorithm with a 1024-bit key. Similarly, an ECC algorithm with a 256-bit key corresponds to an RSA algorithm with a 2048-bit key, while an ECC algorithm with a 320-bit key roughly corresponds to an RSA algorithm with a 5120-bit key. This cryptographic strength and the relatively small size of the keys are precisely the reasons why ECC systems are used in the smart card environment.

The numerical processing units of current smart card microcontrollers generally support ECC, which means that relatively high computation speeds are available. As with the RSA

<sup>10</sup> See also Section 7.2, ‘Hash Functions’, on page 156

**Table 7.9** Example computation times for cryptographic algorithms based on elliptic curves in  $GF(p)$ . The remarkably good times for smart cards without coprocessors are achieved by using a lookup table to accelerate certain time-intensive computations (table size approximately 10 KB)

Implementation	Generate a 160-bit signature	Verify a 160-bit signature
Smart card with 3.5-MHz clock and 8-bit processor	1 s	4 s
Smart card with 3.5-MHz clock and numeric coprocessor	150 ms	450 ms
PC (Pentium III, 500 MHz)	10 ms	20 ms

**Table 7.10** Key parameters of asymmetric cryptographic algorithms used in smart cards

Name	Key size
RSA	1 024 bits (128 bytes)
	2 048 bits (256 bytes)
	4 096 bits (512 bytes)
DSS	1 024 bits
	2 048 bits
ECDSA	160 bits
	224 bits
	256 bits
	384 bits
	521 bits

algorithm, the key size is an important characteristic of these asymmetric cryptographic algorithms.

Interestingly enough, cryptographic systems based on elliptic curves require so little processing power that they can even be implemented in microcontrollers lacking coprocessors. Some typical times for generating and verifying signatures are listed in Table 7.9. Using a lookup table with a size of approximately 10 KB, an eight-bit microcontroller without a coprocessor, operating at 3.5 MHz, needs around 1 s to generate a 160-bit ECC key pair. This can be reduced to 200 ns by using a coprocessor.

Although asymmetric cryptographic algorithms based on elliptic curves are now well known, they are still used only to a limited extent. One reason for this is that they require some system-level modifications in systems designed for use with the well-known RSA algorithm. Another probable but poorly founded reason for their limited use up to now is that they are mathematically considerably more complex, and thus more difficult to understand, than the RSA algorithm. Nevertheless, elliptic curve cryptosystems have the substantial advantage of providing the highest degree of security per bit (in terms of key size) of all currently available asymmetric methods. The key parameters of asymmetric cryptographic algorithms used in smart cards are listed in Table 7.10.

### 7.1.3 Padding

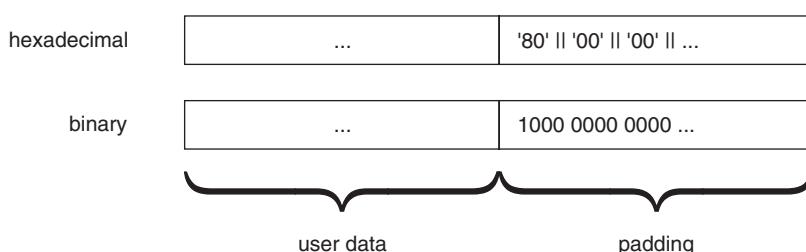
In the smart card realm, symmetric cryptographic algorithms are used primarily in the two block modes: ECB and CBC. As the data supplied to the card does not always fit exactly into an integral number of blocks, it is sometimes necessary to extend the data to fill a block. This process of extending data to correspond to an integral multiple of the block size is called padding.

The party that receives an encrypted padded data block is faced with a problem after the data has been decrypted, since it does not know where the actual data ends and the padding begins. One solution to this problem would be to state the message length at the start of the message, but this would alter the structure of the message, which is generally undesirable. It would also create unnecessary complexity with data that does not always have to be encrypted, since padding is not needed if no encryption is performed, so the length data is not always needed.

It is thus necessary to use a different technique to identify the padding bytes. The algorithm defined in the ISO/IEC 9797 standard is described here as a typical example, although a variety of other techniques are available. As illustrated in Figure 7.16, the most significant bit (msb) of the first padding byte after the user data is set to 1. This byte thus has the hexadecimal value ‘80’. If additional padding bytes are needed, they have the value ‘00’. The message recipient searches backward from the end of the message for a nonzero bit or the value ‘80’. If it is found, the recipient knows that this byte and all subsequent bytes are padding and not part of the message.

In this regard, it is important for the recipient to know whether messages are always padded or padded only if necessary. If padding only occurs when the length of the data to be encrypted is not an integer multiple of the block length, the recipient must take this into account. Consequently, there is often an implicit agreement that padding always takes place, which of course has the disadvantage that occasionally an unnecessary block of padding data must be encrypted, transferred and decrypted.

In some application scenarios, only the value ‘00’ is used for padding. This is because this value is normally used for padding in MAC computations, and using only one padding algorithm yields savings in program code size. Of course, in this case the application must know the exact structure of the data to allow it to distinguish between user data and padding. The MAC is usually appended to the message as shown in Figure 7.17.



**Figure 7.16** Padding according to ISO/IEC 9797 Method 2, in hexadecimal and binary notation



**Figure 7.17** The usual arrangement of a message and a message authentication code (MAC)

**Table 7.11** Typical padding formats using in the smart card realm. The data to be padded is designated here as ‘data’

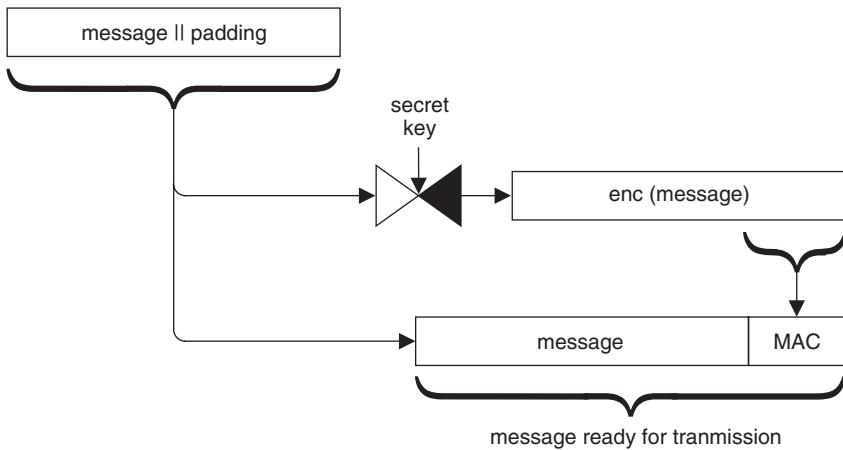
Padding format	Description
ISO/IEC 9797	<p>These padding formats are used for encryption and MAC generation.</p> <p>Method 1: the data is padded with ‘00’ (formal representation: data <math>\parallel n \times \text{‘00’}</math>)</p> <p>Method 2: ‘80’ is appended to the data to be padded, followed by ‘00’ padding bytes (formal representation: data <math>\parallel \text{‘80’} \parallel n \times \text{‘80’}</math>)</p>
ISO/IEC 9796-2	<p>These padding formats are used for digital signatures. The data to be padded is prefixed by a bit sequence starting with 11 and ending with 1, with the number of 0 characters necessary for padding in between, and the code ‘BC’ is appended to the data. In addition, a random number can be included in the padding sequence to individualize the data to be padded.</p> <p>Formal representation with bytewise padding: ‘60’ <math>\parallel n \times \text{‘00’} \parallel \text{‘01’} \parallel \text{data} \parallel \text{‘BC’}</math></p> <p>Formal representation with bytewise padding and individualized data: ‘60’ <math>\parallel n \times \text{‘00’} \parallel \text{‘01’} \parallel \text{RND} \parallel \text{data} \parallel \text{‘BC’}</math></p>
PKCS #1	<p>The Type 1 version of these padding formats is used for digital signatures, while the Type 2 version is used for encryption and MAC generation. The data to be padded is prefixed by a tag and a fixed value or random number with the necessary padding length.</p> <p>Formal representation with Type 1: ‘00’ <math>\parallel \text{‘01’} \parallel n \times \text{‘FF’} \parallel \text{‘00’} \parallel \text{data}</math></p> <p>Formal representation with Type 2: ‘00’ <math>\parallel \text{‘02’} \parallel n \times \text{RND} \parallel \text{‘00’} \parallel \text{data}</math></p>
PKCS #5	For DES and triple DES
ISO 14888	Only for encryption

The various padding methods used with smart cards and their characteristics are summarized in Table 7.11.

#### 7.1.4 Message authentication code and cryptographic checksum

The authenticity of messages is far more important than their confidentiality. The designation ‘authentic’ means that the message has not been altered or manipulated, and is thus genuine. For this purpose, a message authentication code (MAC) is computed and appended to the message before it is sent. The recipient of the message can then compute the MAC of the message on its own and compare it with the received MAC. If they match, the message has not been altered during the transfer.

As illustrated in Figure 7.18, a cryptographic algorithm is used with a secret key to generate the MAC. This key must be known to both of the communicating parties. In principle, a MAC is a sort of error detection code (EDC) that can only be verified if the corresponding secret key is known. For this reason, the term ‘cryptographic checksum’ (CCS) (among other terms) is also used, although from a technical perspective a CCS is equivalent to a MAC. Generally speaking, the difference between the two terms is that ‘MAC’ is used with data transmission and ‘CCS’ is used in all other application areas. The term ‘signature’ is often used in place of ‘MAC’. However, MACs differ from digital signatures in that digital signatures are generated using asymmetric cryptographic algorithms.



**Figure 7.18** Example of a MAC computation process

In principle, any symmetric cryptographic algorithm can be used to compute a MAC. However, triple DES or AES is often used in practice.

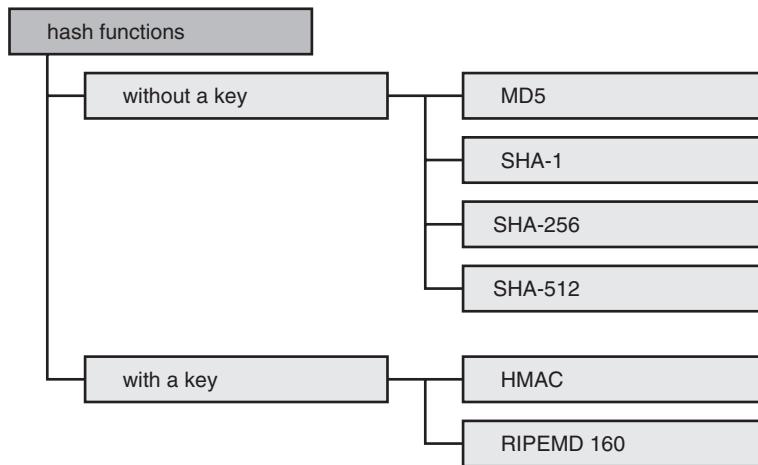
If message encryption is performed in CBC mode, each block is linked to the previous block. This means that the final block depends on all previous blocks. This final block, or part of it, forms the MAC of the message. However, the original message is left in plaintext form, which means that it is transmitted unencrypted.

There are a few important conditions that apply to MAC generation using the DES algorithm. If the length of the message is not an exact multiple of eight bytes, it must always be extended appropriately, which falls under the heading of padding.<sup>11</sup> In most cases only '00' is used for padding (in accordance with ANSI X.99, 'Message Authentication'). This is allowable in this case because there must be prior agreement on the length of the MAC and its location in the message. The MAC consists of the left-most (most significant) four bytes of the final block generated by CBC-mode encryption. The padding bytes are not transmitted with the message. This limits the transmitted data volume to the protected user data and the appended MAC.

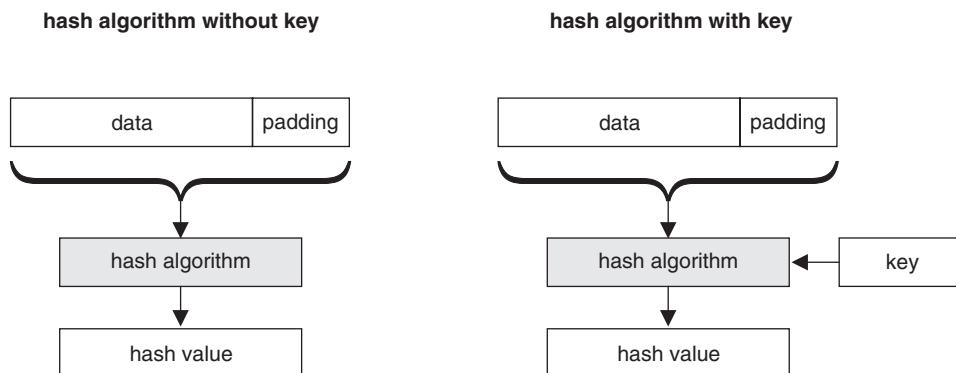
## 7.2 HASH FUNCTIONS

Even powerful computers require a great deal of time to compute digital signatures. In addition, large documents would need a large number of signatures because there is a limit to the size of the input document for signature generation. A special technique is used to resolve this problem. The document is first compressed to a much shorter fixed length, and then the digital signature of the compressed data is computed. Here it does not matter whether the compression is reversible, since the signature can always be reproduced from the original document. The functions used for this type of computation are called one-way hash functions. There are two basic classes of hash functions, as illustrated in Figures 7.19 and 7.20: with a key or without a key.

<sup>11</sup> See also Section 7.1.3, 'Padding', on page 154



**Figure 7.19** A selection of the principal hash functions used in the smart card environment



**Figure 7.20** The operating principle of hash algorithms. The input data must be padded to an integer multiple of the block size of the hash function as necessary. The basic operation of a keyless hash function is shown on the left, while the basic operation of a keyed hash function is shown on the right

In general terms, a one-way hash function is a function that computes a fixed-length value from a variable-length document, such that this value represents the original content of the document in compressed form and cannot be used to derive the original content of the document. In the smart card domain, these functions are used to compute input values for digital signatures and to generate pseudorandom numbers.<sup>12</sup> If the size of the document is not an integral multiple of the block size of the hash function, the document must be padded appropriately.

A good hash function is characterized by several properties. First, the result should have a fixed length so it can be used readily by signature algorithms. As it is usually necessary to process large volumes of data, the hash function must have high throughput. It must also be

<sup>12</sup> See also Section 7.3, ‘Random Numbers’, on page 159

easy to compute the hash value, but it should be difficult, or better yet impossible, to derive the original document from a known hash value. Finally, the hash function must be collision-resistant. This means that it should not be easy to find another document that yields the same hash value as a given document, even though it unquestionably possible for other documents to have the same hash value. This is unavoidable because all possible messages, ranging in length from zero to infinity, must be represented by a set of hash values having the same fixed length. Consequently, collisions can always occur. This is why the term ‘collision-resistant’ is used instead of ‘collision-free’.

If a collision occurs, this means that there are two different documents with the same hash value and thus the same digital signature. This has the fatal consequence of making the signature worthless, since it would be possible to alter the document without anyone being able to detect the fact. This is precisely the objective of one of the two typical forms of attack on hash functions, which consists of systematically searching for a second document that has the same hash value as the original document. If the content of this document is meaningful, the digital signature based on the hash value is discredited. Since the two documents are interchangeable, the signature is worthless. After all, it makes an enormous difference whether a house purchase contract is for €10 000 or €750 000.

The second form of attack on a hash value is somewhat subtler. With this approach, two documents with identical hash values but different contents are prepared in advance. This is not particularly difficult, considering all the special characters and extensions available in the font that is used. The result is that a single digital signature is valid for both documents, and it is impossible to prove which document was originally signed.

Finding two documents with the same hash value is not as difficult as it might seem. This can be illustrated by the birthday paradox, which is well known in the field of statistics. This paradox revolves around two questions. The first question is: how many people must be in a room for the probability to be greater than 50 % that one of them has the same birthday as the person asking the question. The answer can be easily found, since it is only necessary to compare the birthday of the questioner with the birthdays of all the other people in the room. The answer is that there must be at least 183 ( $365 \div 2$ ) people in the room.

The second question reveals the paradox, or better put, the surprising conclusion of this birthday comparison. This question is: how many people must be present in a room for the probability to be greater than 50 % that two people in the room have the same birthday? The answer is only 23 people. This comes from the fact that although only 23 people are present, 253 different pairs (for comparing birthdays) can be formed from this group. The probability that two people have the same birthday is based on these pairs.

Precisely this paradox is utilized in attacking a hash function. It is much easier to create two documents that have the same hash value than to modify an existing document so that it yields a particular hash value. The consequence of this form of attack is that the results of hash functions must be large enough to successfully foil both forms of attack. Most hash functions thus produce values with a length of at least 160 bits, which at present is generally regarded as adequate for protection against the two forms of attack just described.

There are many published hash functions, and many of them are also specified in standards. However, existing hash functions are subject to repeated revision resulting from the discovery of successful forms of attack. The hash functions commonly used in the smart card domain are listed in Table 7.12 and described briefly below. Unfortunately, descriptions of their internal operation is beyond the scope of this book.

The ISO/IEC 10118-2 standard specifies a hash function based on an  $n$ -bit block encryption algorithm (such as AES). With the described algorithm, the length of the hash value can be  $n$  bits

**Table 7.12** Summary of hash functions commonly used in the smart card domain

Algorithm	Hash value length (bits)
ISO/IEC 10118-2	128, 160, 256, 384, 512
MD5	128
RIPEMD-160	160
SHA-1	160
SHA-256	256
SHA-384	384
SHA-512	512
HMAC-MD5	128
HMAC-SHA-1	160

or  $2n$  bits. The MD5 hash function, which is an upgraded version of the MD4 (message digest 4) function developed by Ronald L. Rivest and no longer regarded as secure, was published in 1990/1991. MD5 is still used, although it is no longer considered secure due to the discovery of certain collisions. Both of these hash functions are based on an independent algorithm, and they both generate a 128-bit hash value. In early 1992, the NIST published a hash function for DSS that is known as SHA. It was reworked after certain weaknesses were discovered, and the result has been known since mid-1995 as SHA-1. It is also standardized in FIPS 180-1.

As hash functions that produce hash values shorter than 160 bits are no longer regarded as adequately secure, the NIST has published the SHA-2 family of hash functions. These four algorithms are called SHA 224, SHA 256, SHA 384 and SHA 512, and each of them generates a hash value with a length indicated by its name. No critical forms of attack have been found for any of them as yet, so they are presently still regarded as adequately secure.

There are also hash functions whose calculated hash values depend on a key as well as the input data. They are called keyed hash algorithms. These functions are basically used to compute MACs.<sup>13</sup> A typical example is HMAC, which dates from 1996 and is specified in RFC 2104. This algorithm utilizes a hash function and links it with a secret key. The two common implementations of HMAC are HMAC-SHA-1 and HMAC-MD5, which utilize the hash function designated by the second part of their name as the base function. The advantage of these algorithms is that they allow data integrity and data authenticity to be assured with a single checksum.

As data transmission to and from smart cards is generally slow, hash functions are usually executed in the terminal or in a computer connected to the terminal. This drawback is offset by the fact that it makes the hash functions interchangeable.

## 7.3 RANDOM NUMBERS

Random numbers are repeatedly needed in connection with cryptographic algorithms. In the smart card domain, they are typically used in authentication processes to ensure the uniqueness of a session, as padding for data encryption, and as initial values for send sequence counters. The length of the random numbers needed for these functions usually lies in the range of two

<sup>13</sup> See also Section 7.1.4, ‘Message authentication code and cryptographic checksum’, on page 155

to eight bytes. The maximum value is a natural consequence of the block size of the DES or AES algorithm.

The security of all of these methods is based on random numbers that cannot be predicted or influenced from outside. The ideal solution would be a hardware-based random number generator in the smart card microcontroller. However, it would have to be fully immune to external influences such as temperature, supply voltage and radiation, as otherwise it would be possible to compromise algorithms whose security is based on the randomness of the random numbers. Current random number generators in smart card microcontrollers are usually based on linear feedback shift registers (LFSRs) driven by voltage-controlled oscillators.

Even with state-of-the-art technology, it is difficult to fabricate a good random number generator in the silicon of the microcontroller die that is immune to external influences (a ‘true random number generator’, or TRNG). Consequently, designers often use software to refine the generated random numbers. With this approach, the result of the random number generator is used as the input to a pseudorandom number generator whose random numbers fulfill the necessary quality criteria, so it can effectively be regarded as a true random number generator.

Pseudorandom number generators (PRNGs) use a strictly deterministic algorithm to produce random numbers. These numbers are predictable if the algorithm and its input values are known, although the randomness of their sequences is often quite good.

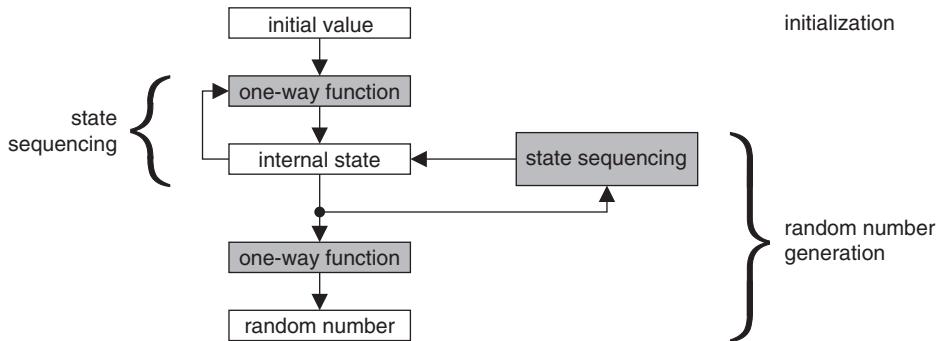
It is also very important to ensure that the all smart cards of a production batch generate different sequences of random numbers, so that the random numbers produced by one card cannot be deduced from those produced by another card from the same batch. This is achieved during completion of the smart card operating system by storing a random number to be used as the seed number (initial value) of the random number generator.

### 7.3.1 Generating random numbers

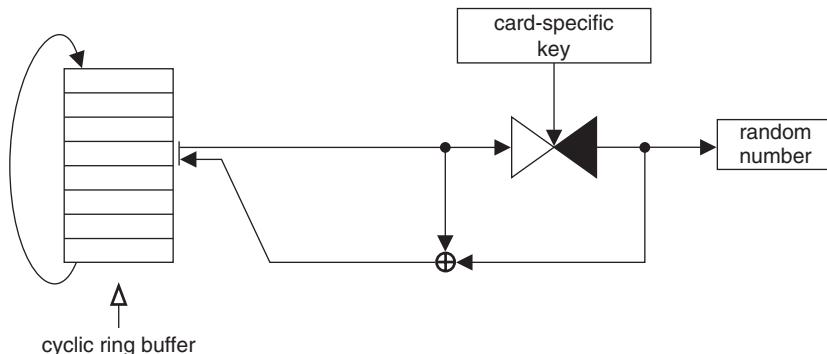
There are many different ways to generate random numbers using software. However, the number of options that can be used with smart cards is severely limited due to the small memory capacity of smart cards and the need to minimize the processing time for generating random numbers. Figure 7.21 depicts the basic operating principle of a strong pseudorandom number generator.

It goes without saying that the quality of the random numbers must not be impaired if a session is interrupted by a reset or by pulling the card out of the terminal. In addition, the generator must be constructed such that the sequence of random numbers is not the same in every session. This may sound trivial, but it requires at least a write access to the nonvolatile memory (EEPROM or flash) to store a new random number generator seed value for the next session. RAM is not suitable for this purpose because it needs power to retain its contents. A potential attack scenario would be to keep generating random numbers until the NVM cells that hold the seed number are worn out. In theory, this would cause the same sequence of random numbers to occur in every session, which would make them predictable and thus give the attacker an advantage. This type of attack can easily be averted by fashioning the relevant part of the NVM as a ring buffer and blocking all further operations after a write error occurs.

Another very important criterion for software random number generators is that they must never run in an endless loop, as otherwise the time required to repeat the cycle of random numbers would be reduced dramatically. In this case it would be easy to predict the numbers, and the system would be compromised.



**Figure 7.21** Basic operating principle of a strong pseudorandom number generator with the following stages: initialization, state sequencing, and random number generation. The initial value (seed) should ideally be a random number obtained from a physical random number generator



**Figure 7.22** Example architecture of a pseudorandom number generator for smart card operating systems that utilizes a symmetric cryptographic algorithm. This generator is primarily designed to minimize the number of NVM write accesses

Almost every smart card operating system has an encryption algorithm for authentication, which can form a good starting point for building a random number generator. This is based on the idea that a good encryption algorithm churns the plaintext as thoroughly as possible so it cannot be deduced from the ciphertext without knowledge of the key. A principle known as the avalanche criterion says that on average, changing one input bit should cause half of the output bits to change. This property can be put to very good use for generating random numbers. The exact structure of the generator depends on the specific implementation.

Figure 7.22 illustrates a possible implementation. This generator uses a symmetric block encryption algorithm (such as AES) with a block length of eight bytes and the output fed back to the input.

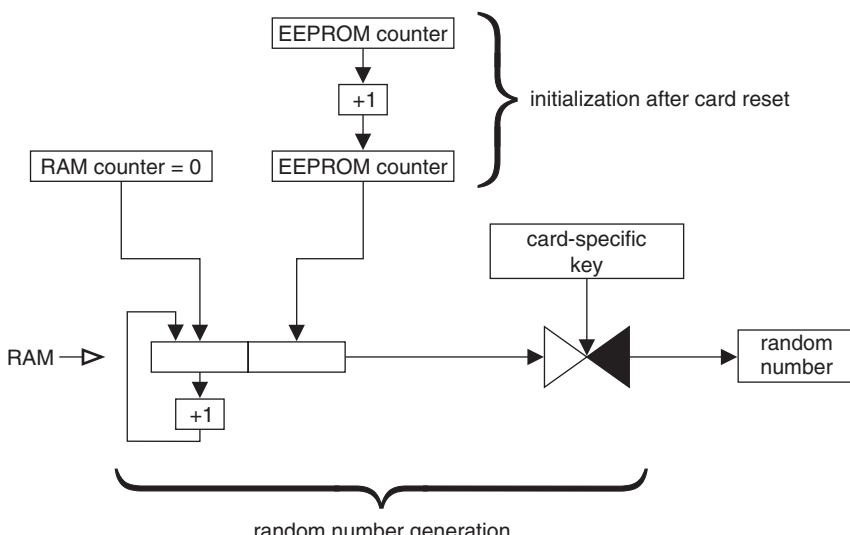
The generator works essentially as follows. The value of a ring buffer element is encrypted by the symmetric cryptographic algorithm with a card-specific key. The resulting ciphertext is the eight-byte random number. It is XORed with the previous plaintext to produce a new entry for the ring buffer in nonvolatile memory. The generator then advances to the next entry in the ring buffer.

A card-specific key for the AES algorithm is stored in each card during completion, and at the same time random numbers that serve as seed numbers are stored in the ring buffer, which for example could have twelve eight-byte entries. This ensures that each card produces a unique series of random numbers, and the ring buffer structure in this example increases the lifetime of the generator by a factor of twelve. If the guaranteed lifetime of the EEPROM is 100 000 write cycles per page, this generator can produce at least 1200 000 eight-byte random numbers.

Erasing and writing eight bytes of EEPROM memory takes about 14 ms ( $2 \times 2 \times 3.5$  ms), and executing the DES algorithm takes about 17 ms at 3.5 MHz if it is implemented in software. The processing time required for other activities is negligible. The smart card thus needs around 31 ms to generate a random number. If the AES computation is performed in hardware (with a typical performance of 0.1 ms/block), it would take only 14.4 ms to generate a random number using this method.

Figure 7.23 shows another example of a pseudorandom number generator. It is initialized each time the card is reset, which is the only time a write access to the EEPROM occurs. Only RAM accesses are used for the subsequent generation of random numbers, which makes this generator relatively fast. However, the disadvantage of this is that the generator uses a few bytes of RAM for the duration of the session. The statistical quality of this pseudorandom number generator is not very good, but it is adequate for normal smart card authentication processes. The primary concern with such processes is to avoid generating random numbers with short repeat cycles, since this would allow authentication to be compromised by replaying messages from previous sessions.

The FIPS 140-2 standard recommends that security modules perform statistical tests to check their built-in random number generators after every reset. Only after these tests have



**Figure 7.23** Example architecture of a pseudorandom number generator for smart card operating systems. This generator is faster than the one shown in Figure 7.22 on the preceding page because only one EEPROM write cycle is necessary for each session. The quality of the random numbers it produces is adequate for normal smart card authentication processes using the challenge-response method

been successfully completed should the random number generator be released for further use. Most standard smart card operating systems currently available do not include this capability because it is assumed that the deterministic nature of the pseudorandom number generators will prevent any significant changes in the statistics of the generated random numbers.

There are vast numbers of proposals, standards and designs for pseudorandom number generators. Some well-known examples are the generators defined in the X9.17 and FIPS 186 standards, the proposals in Internet RFC 1750, and the arrangements described by Bruce Schneier [Schneier 96], Peter Gutmann [Gutmann 98a], and Benjamin Jun [Jun 99]. The guiding principle with random number generators should always be to keep their design as simple and comprehensible as possible. Only then is it possible to assess their characteristics and thus determine their quality.

### 7.3.2 Testing random numbers

After a random number generator has been implemented, it is generally necessary to test the quality of the numbers it produces. Fundamentally, the frequency of ones and zeros in the generated random numbers should be nearly the same. However, it is not enough to simply print out a few numbers and compare them. Random numbers must be tested mathematically using standard statistical methods. It goes without saying that a large number of eight-bit random numbers are needed for such tests. In order to obtain reasonably reliable results, it is necessary to generate and analyze 10 000 to 100 000 random numbers. The only way to test these numbers is to use computer-aided test programs. A number of suitable tests are listed in Table 7.13.

In order to evaluate the quality of random numbers, it is also necessary to assess the distribution of the generated numbers. If it is very uneven, with certain values more common than others, exactly these regions can be used for prediction purposes. In this connection, Bernoulli's theorem should be satisfied as closely as possible. This theorem states that in processes with random outcomes, the likelihood of the occurrence of a particular outcome depends only on the probability of occurrence of the outcome, independent of what comes before it. For example, the likelihood that a 4 will appear when a die is thrown is always 1/6, independent of whatever number appeared on the previous throw. This is called event independence.

The period of the random numbers, which means the number of random numbers generated before the series repeats itself, is also very important. It should naturally be as long as possible, and in any case longer than the lifetime of the random number generator. In this way, the possibility of attacking the system by recording all random numbers generated during a complete period can be excluded easily and reliably.

There are many statistical tests for examining the randomness of events, but in practice we can limit ourselves to a few simple tests whose results are easily interpreted. There are also many publications on the subject of testing random numbers [Knuth 97, Menezes 97], as well as corresponding standards [FIPS 141-2, RFC 1750]. The open-source Diehard test suite from George Marsaglia [Diehard], which includes a fairly large number of individual tests, is also popular for testing the quality of random numbers. Two documents from the BSI have also established a position in the smart card world as references for testing random number generators. They are AIS 20, ‘Functionality classes and evaluation methodology for deterministic random number generators’, and AIS 31, ‘Functionality classes and evaluation methodology for physical random number generators’.

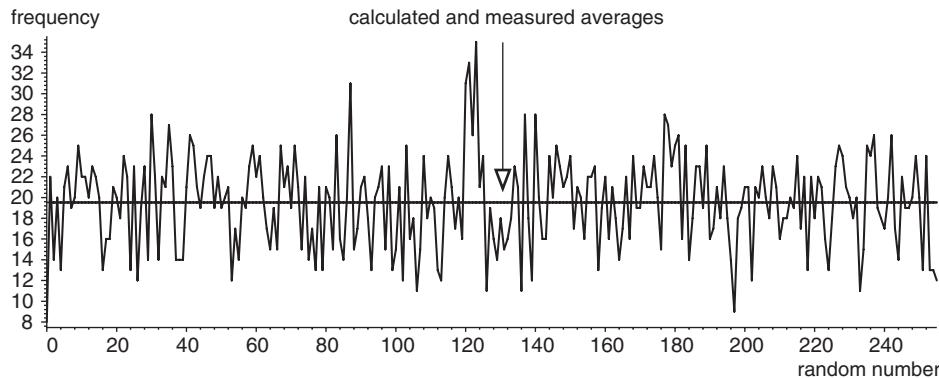
**Table 7.13** Summary of customary statistical tests for random numbers

Test and reference	Remarks
AIS 20	A collection of statistical tests for random number generators
Coupon collector test [Knuth 97]	$\chi^2$ distribution of nonpattern sequences in a series of random numbers
Frequency test [Knuth 97, Menezes 97]	Counts the number of ones in a series of random numbers
Gap test [Knuth 97]	Determines nonpattern sequences in a series of random numbers
FIPS 140-2 long run test	Determines whether a sequence of ones or zeros with a length of 34 bits occurs in a series of random numbers with a length of 20 000 bits
FIPS 140-2 monobit test	Counts the number of ones in a series of random numbers with a length of 20 000 bits
Poker test [Knuth 97]	$\chi^2$ distribution of patterns occurring in a series of random numbers
Poker test [Menezes 97]	$\chi^2$ distribution of nonpattern sequences in a series of random numbers
FIPS 140-2 poker test	Counts four-bit patterns in a series of random numbers with a length of 20 000 bits
FIPS 140-2 run tests	Determine the maximum length of a sequence of all ones or all zeros in a series of random numbers with a length of 20 000 bits
Serial test [Knuth 97]	Determines patters appearing in a series of random numbers
Serial test [Menezes 97]	Counts four-bit patterns in a series of random numbers with a length of 20 000 bits
Spectral test [Knuth 97]	Determines the distribution of successor values of random numbers

One test that is easy to perform and interpret is to count the number of times each byte value occurs in a large set of random numbers. If the results are displayed graphically, they give a good indication of the distribution of the numbers. The possible values are plotted on the Y axis and the frequency of occurrence is plotted on the X axis, as illustrated in Figure 7.24 on the next page.

If this plot is used to examine eight-byte random numbers, the values plotted on the horizontal axis can still only be single-byte or at most two-byte numbers, since the number of samples needed for a statistical analysis would otherwise become extremely large. A good guideline is that every random number should occur approximately five to ten times for each value in order to obtain reasonably reliable results. In this way, it is possible to quickly see whether the random numbers that have been generated exploit the full scope of possible byte values. If certain values are much more common than others, this offers an attacker a possible starting point.

Unfortunately, this test only indicates the statistical distribution of the random numbers; it does not say anything about their order of occurrence. For example, it would be conceivable for a random number generator to simply output cycles of numbers sequentially from 0 to



**Figure 7.24** Statistical distribution of a series of 5000 single-byte random numbers. This is also called the spectral distribution over one byte. The numbers were generated by a typical smart card pseudorandom number generator. Based on purely mathematical considerations, each of the possible values (0–255) should occur 19.5 times

255. This would yield an outstandingly uniform distribution, but the numbers would be fully predictable. Other tests must be used to assess this quality criterion.

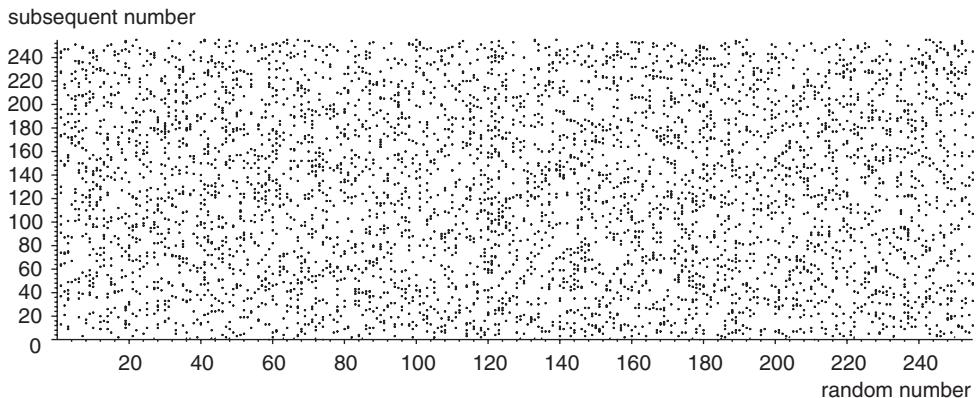
Another practical test that yields a simple and quick estimate of the quality of a series of random numbers is to compress the series using a file-compression program. According to Shannon, the degree of compression that is possible is inversely proportional to the randomness of the set of generated numbers.

A significantly more reliable test is the very well known  $\chi^2$  test. Like the previously described graphic test, it is used to check the uniformity of the statistical distribution, but it is significantly more exact because it uses a mathematical method [Bronstein 97]. If the random numbers are assumed to be evenly distributed, their mean value and standard deviation can be calculated. The deviation from the normal distribution can then be determined from the  $\chi^2$  distribution. From this, the distribution of the random numbers can be assessed in numerical terms.

However, this test cannot be used to assess the sequence of the generated random numbers. Other statistical tests [Knuth 97] can be used for this purpose, such as the serial test, which determines the intervals between patterns appearing in the sequence of random numbers. Similarly, the gap test determines the intervals that are free of patterns. The poker test should also be used to determine the  $\chi^2$  distribution of pattern sequences, and the coupon collector test should be used to determine the  $\chi^2$  distribution of nonpattern sequences.

The spectral test, which determines the relationship between each random number and its subsequent number, also has a certain relevance [Knuth 97]. In the two-dimensional version of this test, random numbers and their direct successors are plotted in an X-Y coordinate system as illustrated in Figure 7.25 on the following page. The three-dimensional version of this test includes the successor of the successor number and requires a third axis (the Z axis). Spectral tests with  $n$  dimensions can be performed in a similar manner, but for understandable reasons they cannot be represented in graphic form.

At minimum, all of the tests described above must be performed and evaluated in order to obtain a secure and reliable assessment of a random number generator. Additional analyses



**Figure 7.25** Graphic representation of the distribution of the successor values of 5000 single-byte random numbers, which is a form of spectral test. From the regularity of the pattern, you can see at a glance that the successor values have a nearly uniform distribution. The numbers were produced by a typical smart card pseudorandom number generator

and tests can be used to confirm the results of these tests. Only in this way is it possible to make a reasonably accurate assessment of the quality of random numbers.

Considering the purposes for which random numbers are used in smart card applications, the effort required to construct a random number generator with superior characteristics is usually not justified. For example, if the random numbers used for authentication could be predicted, the effect on security would be relatively minor because no attack is possible without the knowledge of the secret key used for the encryption of the random number.

However, a significantly more critical situation would arise if the random number generator could be manipulated, for example such that it always produced the same sequence of random numbers. In this situation, replay attacks would be not only possible but also successful. They are also possible if the period of the random numbers is very short. The primary properties required of the random number generator must be carefully considered in each case, since they impact the design of the random number generator. Although very complex methods may enable the generation of random numbers with very high quality, they usually result in increased memory usage, and memory is a scarce resource in smart cards.

## 7.4 AUTHENTICATION

Authentication is the process of proving the identity of an entity. In an authentication process, a specific method is used to determine whether an entity is actually what it claims to be. Authentication can be based on possession, knowledge, or a biometric feature. The terms ‘two-factor authentication’ and ‘three-factor-authentication’ are also used in this connection. Two-factor authentication combines two of the three different forms of authentication (authentication by possession, by knowledge, or by a biometric feature). This means that two different forms of authentication must be performed with positive results in order to achieve successful authentication. With three-factor authentication, all three forms of authentication are combined, which means that all three forms must be performed with positive results in order to achieve successful authentication.

This can be illustrated by an example. Three-factor authentication of a person can be achieved if the person presents a passport (possession), knows a secret (knowledge), and passes a fingerprint test (biometric feature). All three forms of authentication must yield positive results in order for the person to be authenticated.

Incidentally, authentication is the process of determining whether an entity or a message is genuine and unaltered, while identification is a process that leads to the recognition of an entity. Identification can be performed by means of authentication.

Determining whether a particular action is allowed to be performed, which means whether someone has been granted the authority to do something, is called authorization. For example, when a credit card transaction is authorized by the credit card issuer, the card data is checked to verify that the data is correct, the purchase amount is less than the allowed limit, and other conditions are satisfied. The intended payment is allowed if all checks are positive. Authorization can be achieved by the authentication of the entity concerned, such as a smart card. In simplified terms, authorization amounts to granting someone permission to perform a particular action.

The purpose of authentication is to verify the identity and genuineness of a communication party. Translated into the world of smart cards, this means that the card or the terminal determines whether its opposite party is a genuine terminal or a genuine smart card, respectively.

Authentication can be further classified into static authentication and dynamic authentication. With static authentication, the same (static) data is always used for authentication (for example, supplying a password). By contrast, dynamic authentication is intended to provide protection against attacks based on replaying data recorded during previous sessions, since different data is used for each authentication.

The challenge-response method is used almost exclusively for dynamic authentication in smart card systems. In this method, one of the communicating parties asks the other party a randomly generated question (the challenge), and the other party uses an algorithm to compute a response and then returns it to the first party. Naturally, the algorithm should preferably consist of encryption using a secret key that represents the shared secret of the two communicating parties. This is significantly more secure than a static method, such as supplying a password, because an attacker can very easily replicate static authentication by intercepting and replaying the data. By contrast, with challenge-response authentication the shared secret used for authentication does not have to be transferred over the interface and the data that is exchanged differs from one session to the next.

Another fundamental distinction is made between unilateral and mutual authentication. Successful unilateral authentication establishes the authenticity of one of the two communicating parties. Successful mutual authentication establishes the authenticity of both of the communicating parties.

The authentication methods used with smart cards, which are based on cryptographic algorithms, can be further classified into symmetric and asymmetric methods. The methods presently used with smart cards are almost exclusively symmetric. Asymmetric methods, such as methods based on the RSA algorithm or similar algorithms, are not used to any significant extent with smart cards due to their slow execution speeds. However, it is conceivable that this will change in the future. In any case, the operating principles of asymmetric methods are the same as those of symmetric methods.

There are several standards that deal with the authentication of devices. The ISO/IEC 9798 standard is the most prominent. Part 2 of this standard describes symmetric methods, while Part 3 describes asymmetric methods. Fundamentally, the five parts of the ISO/IEC 9798

standard form an outstanding compilation of commonly used authentication methods, including symmetric, asymmetric, MAC-based, and zero-knowledge-based methods.

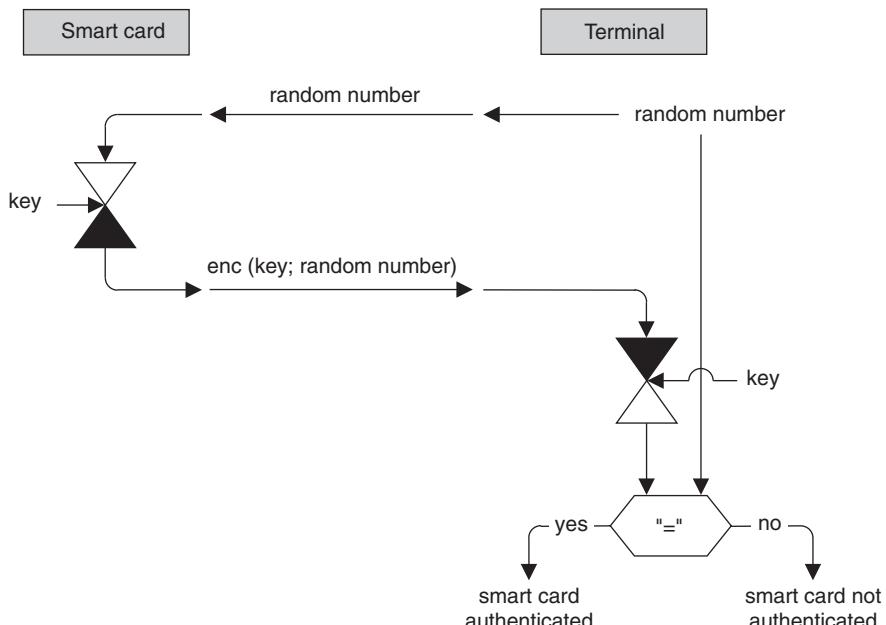
### 7.4.1 Unilateral symmetric authentication

Unilateral authentication assures one party of the trustworthiness of the other party. This is only possible if the parties have a shared secret, in which case authentication verifies their knowledge of this secret. The secret is the key for an encryption algorithm, and the security of the authentication method depends entirely on this key. If the key becomes known, an attacker can be authenticated just as readily as a genuine party.

Figure 7.26 shows the operating principle of unilateral authentication with a symmetric cryptographic algorithm. For the sake of clarity, here it is assumed that a smart card is authenticated by a terminal. This means that the terminal determines whether the smart card is trustworthy.

The terminal first generates a random number and sends it to the smart card. This is the challenge. The smart card encrypts the random number it receives, using a key known to both the card and the terminal. The security of the method depends on this key, since only the possessor of the secret key can generate the correct response to the challenge from the terminal.

The card then returns the result of the encryption to the terminal. This is the response to the challenge. The terminal uses the secret key to decrypt the encrypted random number it received and compares the result with the random number it originally sent to the smart card.



**Figure 7.26** Operating principle of unilateral authentication with a symmetric cryptographic algorithm. This example shows the authentication of a smart card by a terminal as implemented using the ISO/IEC 7816-4 INTERNAL AUTHENTICATE command

If the two numbers match, the terminal knows that the smart card knows the secret, and it concludes that the card is authentic.

This procedure cannot be attacked by replaying a challenge or response intercepted during an earlier session, since a different random number is generated for each session. The only form of attack with a reasonably good chance of success would be a systematic search for the secret key. As the challenge and the response are simply a plaintext–ciphertext pair, the secret key could be discovered using a brute-force attack.

If all cards for a particular application have the same key and this key becomes known, the entire system is discredited. In order to prevent this, in most cases only card-specific keys are used in practice. This means that each card has its own key, which the terminal can derive from a nonsecret card feature. This individual feature may be the serial number that is written to the chip during fabrication, or some other number that is specific to each individual card. In order to compute the card-specific key, the terminal requests the chip number from the smart card. This number is specific to the card and unique within the system, so there is no other card in the system with the same number.

The card-specific secret key depends on the card number and the master key, which is known only to the terminal. In practice, part of the card number is encrypted using the master key to produce the card-specific authentication key. The triple DES or AES algorithm can be used for encryption.

Here it must be borne in mind that if the master key (which is known only to the terminal) is compromised, the entire system is compromised because all card-specific authentication keys can be computed using the master key. The master key must therefore be stored securely in the terminal (in a security module, for example), and if possible it should be possible to actively erase the key if an attack is detected.

After the terminal has computed the necessary authentication key for the card, the usual challenge–response process occurs. The smart card receives a random number, encrypts it using its card-specific key, and returns the result to the terminal. The terminal reverses the sequence of operations performed by the smart card and compares the two results. If they match, the terminal and the smart card have a shared secret, namely the secret card-specific key, and the smart card has been authenticated by the terminal.

### 7.4.2 Mutual symmetric authentication

The operating principle of mutual authentication is based on dual unilateral authentication. Two successive unilateral authentications (one for each of the communicating parties) could also be used to achieve the same result as mutual authentication. However, the communication effort must be kept as small as possible to minimize the transaction time, so there is a method that merges the two unilateral authentications. This also provides better security than two successive authentications because it is much more difficult for an attacker to intervene in the communication process. Sequence diagram 11.8 on page 378 shows the corresponding command sequence.

In order to compute the card-specific authentication key from the card number, the terminal first needs the card number. After the terminal receives this number, it computes the card's specific authentication key. It then requests a random number from the card, and at the same time it generates a random number itself. The terminal concatenates the two random numbers and encrypts the concatenated number using the authentication key. It then sends the resulting ciphertext to the card.

The card decrypts the received block and checks whether the random number it received from the terminal matches the number it previously sent to the terminal. If it does, the smart card knows that the terminal possesses the secret key. This authenticates the terminal with respect to the smart card. Next, the smart card swaps the two random numbers, encrypts the resulting number using the secret key, and sends the result back to the terminal. The purpose of swapping the random numbers is to make the challenge and the response different.

The terminal decrypts the received block and compares the random number received from the smart card with the number it previously sent to the card. If they match, the smart card has been authenticated with respect to the terminal. This completes the mutual authentication, and the terminal and the smart card both know that the opposite party is trustworthy.

To minimize the communication time, the smart card can also return a random number together with its card number. This is particularly attractive when mutual authentication takes place between a smart card and a background system. In this case, the smart card is accessed directly by the background system, with the terminal being transparent. The data transmission rate is sometimes relatively low in such situations, so the communication process must be simplified as much as possible.

### 7.4.3 Static asymmetric authentication

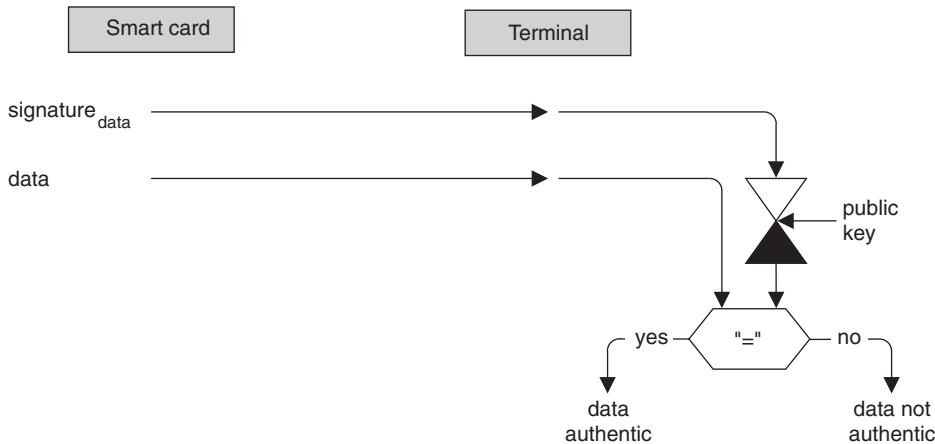
Some smart card microcontrollers do not have a numerical processing unit (NPU) that can be used to execute the RSA algorithm. This is primarily because a NPU takes up additional space on the chip, which increases its price.

However, a supplementary asymmetric authentication offers increased protection because it requires an attacker to break two cryptographic algorithms instead of only one, so this capability is often desirable. Static authentication of the smart card by the terminal can be used as an expedient to avoid the problem posed by the absence of a suitable NPU in the smart card. This only requires a verification in the terminal, and a supplementary processing unit in the terminal does not significantly increase its total cost. This approach is thus much more economical than using special smart card microcontrollers. In addition, it is much faster than dynamic asymmetric authentication because it requires only one asymmetric encryption instead of two.

The price that must be paid for this compromise is reduced security of the authentication process. With a static method, there is naturally no protection against replay. This is why it is used only for supplementary verification of the authenticity of the card, which has already been checked using a dynamic symmetric method.

The method essentially works as follows. During personalization, card-specific data is loaded into each smart card. This can for example be a card number and the name and address of the cardholder. A digital signature of this data, which cannot be changed during the lifetime of the card, is computed during personalization using a secret key. This key is used globally in the system. When the card is used in a terminal, the terminal reads the signature and the signed data from a file in the card. The terminal has the public key that is valid for all smart cards in the system, and it can use this key to encrypt the signature read from the card and compare the result with the data read from the card. If they match, the card has been authenticated by the terminal. This process is illustrated in Figure 7.27 on the next page.

In addition to lacking protection against replay, the method described above has the drawback that global keys are used for signature generation and verification. Although the key in



**Figure 7.27** Operating principle of static unilateral asymmetric authentication of a smart card by a terminal using a global key

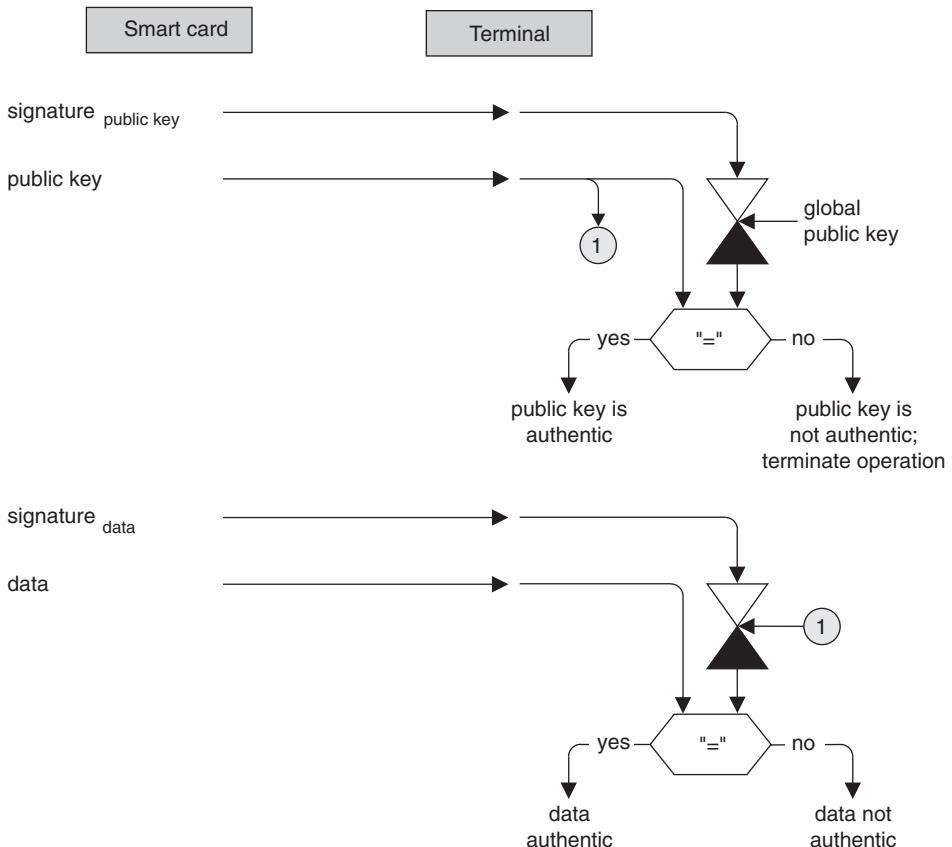
the terminal is public and thus does not require protection, as a matter of principle keys that are the same for all cards should not be used in relatively large systems. If such a key is discovered as the result of an attack, or if it becomes known for any other reason, authentication is rendered worthless in the entire system. This means that card-specific key pairs must be used for static authentication.

However, this creates a problem with the memory capacity of the terminals, since each terminal must hold all available public keys for signature verification. Even in a medium-sized system, such as a system with one million smart cards, this would require each terminal to have 256 MB of memory for key storage if 2 048-bit RSA keys are used. This would increase the cost of the terminals and the associated key management effort to a level that would not be acceptable to system operators.

If symmetric methods are used, it is fairly easy to derive the card-specific keys from a master key.<sup>14</sup> This is not possible with asymmetric methods, due to the key derivation process. Consequently, a different approach is taken when card-specific keys are necessary. The public key for signature verification is stored in the card along with the signature. With the previously mentioned example, the amount of memory needed to store the public keys is still 256 MB, but it is now distributed over one million cards in 256-byte chunks. The terminal reads the public key from a file in the smart card and can then use it to verify the signature, as illustrated in Figure 7.28 on the following page. This avoids the need to store all the public keys of the system in every terminal.

However, an attacker could now generate a key pair and use these keys to sign the data in a counterfeit card. The terminal would read the public key and conclude that the card was genuine, so refinement of the method described above is necessary. This consists of using a secret global key to sign the combination of the public key and the card-specific key stored in each card and storing this signature in the card. The terminal now proceeds as follows. It first reads the public and card-specific keys from the card and uses the public global key to check

<sup>14</sup> See also Section 7.7.1, 'Derived keys', on page 181



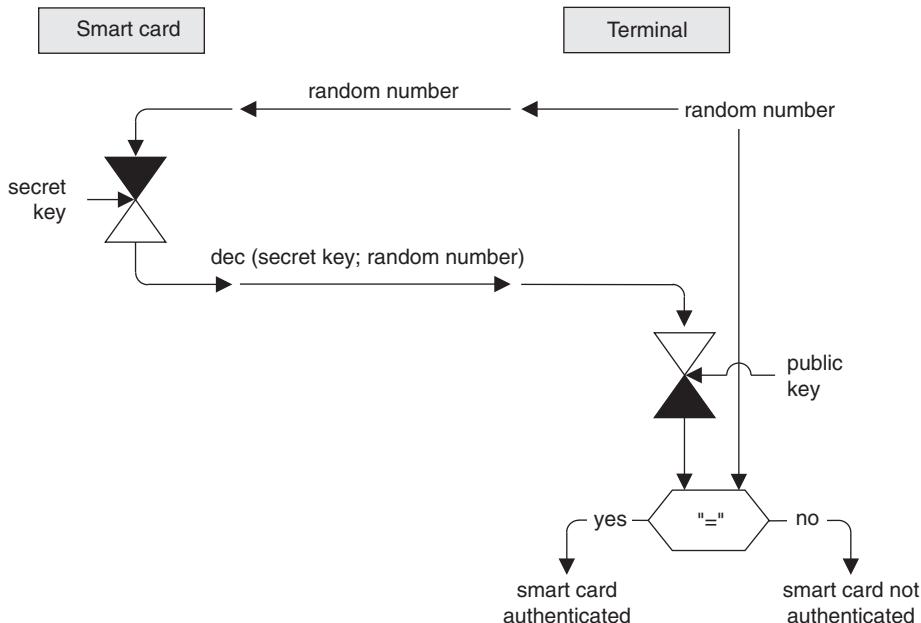
**Figure 7.28** Operating principle of static unilateral asymmetric authentication of a smart card by a terminal using a card-specific key

the authenticity of the card-specific key. If it is authentic, the terminal then reads the actual data and verifies it using the public key stored in the smart card.

These two methods are already used in some systems, and they will certainly be used increasingly in the coming years. However, as soon as a numerical processing unit for asymmetric cryptographic algorithms can be integrated in smart card microcontrollers without significantly increasing their cost, these methods will become much less important. Their main drawback is the lack of protection against replaying data from earlier sessions. Although this can be remedied to certain extent by special techniques, such as using the signed data with the subsequent symmetric cryptographic algorithm, it is still not possible to match the security level of dynamic authentication methods.

#### 7.4.4 Dynamic asymmetric authentication

The previously described static asymmetric methods have certain disadvantages. These can be eliminated by using dynamic authentication, which provides protection against replaying



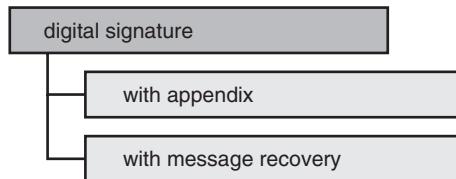
**Figure 7.29** Operating principle of dynamic unilateral asymmetric authentication of a smart card by a terminal

previously intercepted data. The usual method is to use a random number as the input to a cryptographic algorithm. Of course, this requires the smart card to have a processing unit that can execute the asymmetric cryptographic algorithm.

Figure 7.29 illustrates unilateral authentication using a global public key. If card-specific authentication keys are necessary, methods for storing and authenticating card-specific public keys as described in Section 7.4.3, ‘Static asymmetric authentication’, on page 170 are also necessary.

As with symmetric authentication, the terminal generates a random number and sends it to the smart card. The smart card uses the private key to decrypt the random number<sup>15</sup> and returns the result to the terminal. The terminal holds the global public key, and it uses this key to encrypt the random number received from the card. If the result of this operation is the same as the random number previously sent to the smart card, the card has been authenticated by the terminal.

<sup>15</sup> Decryption is used in signature generation operations due to the convention that with an asymmetric cryptographic algorithm, the private key is always used for decryption and the public key is used for encryption. This convention goes back to the origins of the RSA algorithm. One of the ideas at that time was that agents operating in hostile territory could use the RSA algorithm to encrypt confidential information. They would only need to know the RSA algorithm and the public key in order to send their messages to headquarters in encrypted form. The messages could then be decrypted by headquarters staff in friendly territory using the private key. The main advantage of this arrangement is the ease of key distribution, since there is no inherent need to use security measures when conveying the keys to the agents. Even if a key becomes known, nobody can use it to decrypt the encrypted messages, since this requires the private key. The convention of using the public key for encryption and the private key for decryption, which still applies, arose from this early application scenario for the RSA algorithm, with its strong military tint.



**Figure 7.30** The two basic forms of digital signatures

The basic features of mutual authentication of the smart card and the terminal are similar to the previously described method for unilateral authentication. However, it takes a relatively long time due to the large amount of data that must be exchanged and the time-consuming asymmetric encryption algorithm. Consequently, it is presently used very rarely.

## 7.5 DIGITAL SIGNATURES

Digital signatures, which are often called electronic signatures, are used to establish the authenticity of electronically transmitted messages or electronic documents. Verification of the signature can be used to determine whether a message or document has been altered. As shown in Figure 7.30, there are two basic forms of digital signature: with appendix or with message recovery.

A signature has the property that it can be correctly produced by only one individual, but it can be verified by any recipient of a message – or at least, by any recipient who has previously seen the signature or has a copy available for comparison. This is also the essential property of a digital signature. Only one smart card can sign a document, but all cards can check whether the signature is genuine. Due to required property, asymmetric cryptographic methods provide an ideal basis for digital signatures.

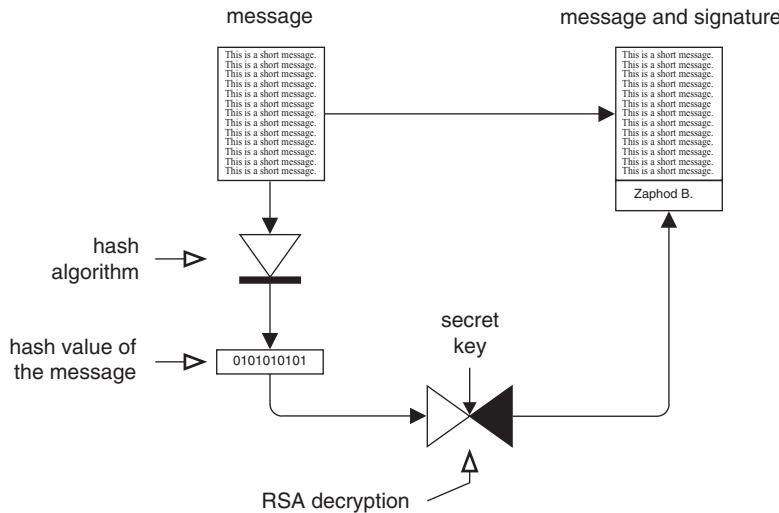
The message or document to be signed is usually at least several thousand bytes long. In order to keep the computation time for generating the cryptographic checksum within acceptable limits, the checksum is not computed from the entire data volume. Instead, a hash value is first computed from the data. In simplified terms, hash functions<sup>16</sup> are one-way data compression functions. This compression is not reversible, which means that the original data cannot be recovered from the compressed data. Hash values can be computed very quickly, which makes hash functions an ideal aid for generating digital signatures.

The term ‘digital signature’ is normally used only in connection with asymmetric cryptographic algorithms, since the combination of public and private keys makes these algorithms very suitable for use with digital signatures. Nonetheless, signatures based on symmetric cryptographic methods are often used in practice. However, with such signatures it is only possible to verify the authenticity of a document if the secret key used to generate the signature is known. Such a signature is thus actually not a signature in the true sense of the word, but it is often referred to as such in practice. The term ‘digital’ is omitted in such cases to indicate the type of algorithm that is used.

In terms of information technology, there are two ways to attach a signature to a message. The first is a form of cryptographic checksum of a given data string, similar to a message authentication code (MAC),<sup>17</sup> with the signature appended to the actual message. This is

<sup>16</sup> See also Section 7.2, ‘Hash Functions’, on page 156

<sup>17</sup> See also Section 7.2, ‘Hash Functions’, on page 156



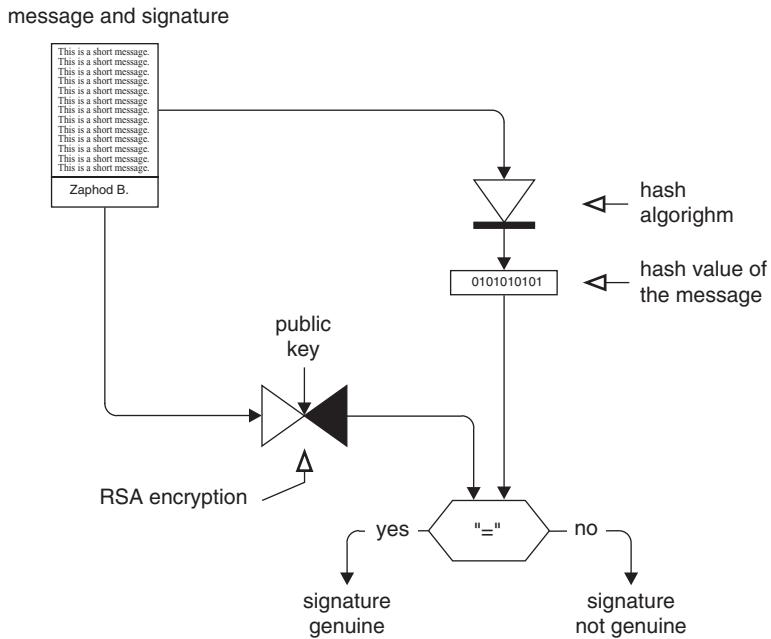
**Figure 7.31** Signing a message with the RSA algorithm by appending the generated signature to the message (digital signature with appendix)

called ‘digital signature with appendix’. It has the advantage that the message can be read completely without requiring prior verification of the signature. However, it has the drawback that the size of the message is increased by the length of the signature, which can be a problem in some cases. This drawback can be avoided by using the second technique for attaching a digital signature to a message, which is called ‘digital signature with message recovery’. With this technique, the hash value of the actual message is first appended to the message, and then the input block of the digital signature algorithm is formed starting from the end of the resulting data string. The length of the digitally signed message is thus increased only by the length of the hash value, although the message cannot be read completely until the digital signature has been verified.

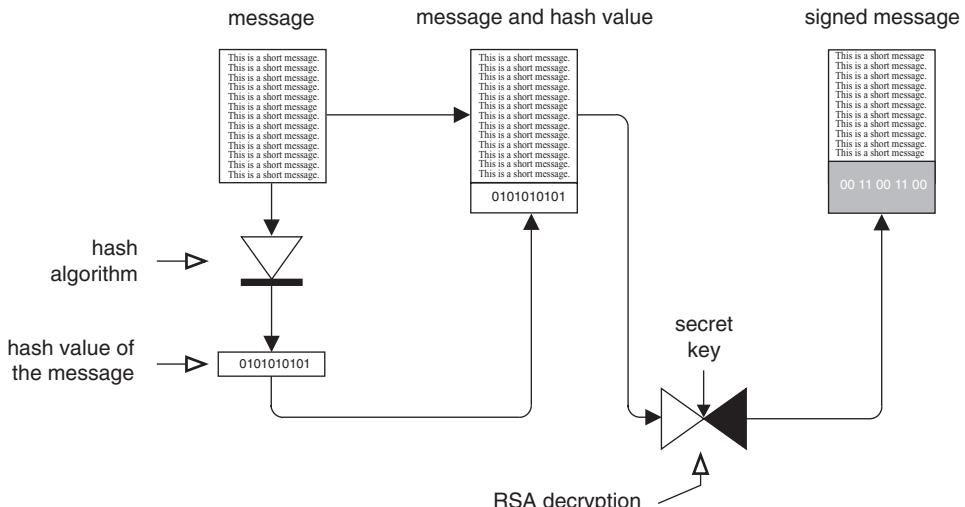
The process of generating a digital signature with appendix can be depicted quite easily, as shown in Figure 7.31. First, a hash algorithm is used to compute a hash value from the message, which may for example be a file produced by any desired word processing program. This hash value is then decrypted using an asymmetric cryptographic algorithm, such as RSA in the example shown in the figure. The result of this computation is the signature, which is appended to the message.

The signed message can now be sent via a nonsecure path to the recipient. The recipient proceeds as illustrated in Figure 7.32 on the next page. The recipient separates the signature from the message and then compresses the message using the same hash algorithm as was used by the sender. The digital signature is encrypted using the public key of the RSA algorithm, and the result is compared with the computed hash value. If these values match, the message has not been altered while underway; otherwise, either the message or the signature has been altered during transmission. This means that authenticity is no longer assured and it cannot be assumed that the content of the message is unaltered.

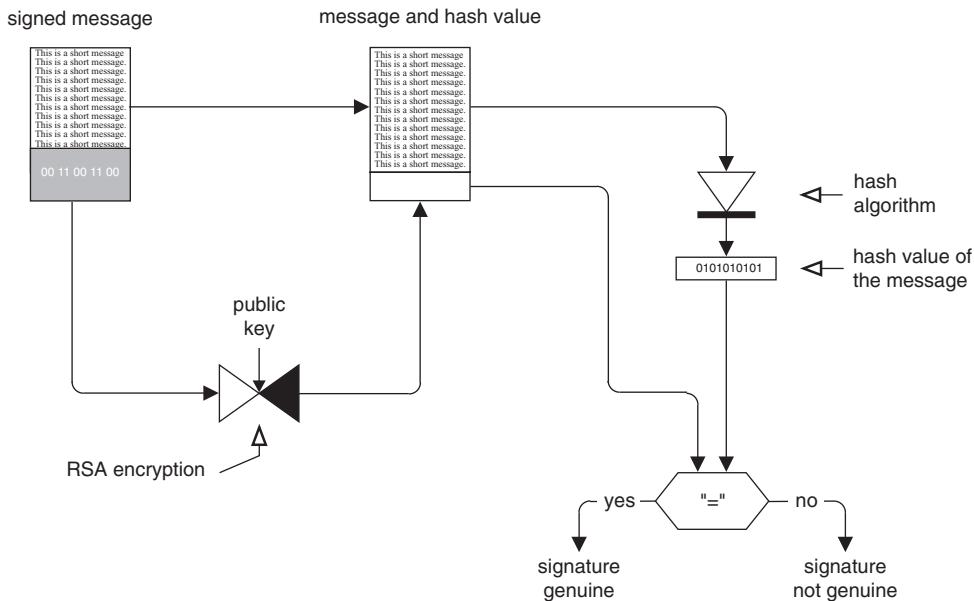
The procedures for generating a digital signature message recovery and verifying this type of digital signature are somewhat more complicated, as illustrated in Figure 7.33 on the following page and Figure 7.34 on page 177.



**Figure 7.32** Verifying the signature of a message that has been signed using the RSA algorithm with the signature appended to the message (digital signature with appendix)



**Figure 7.33** Signing a message with the RSA algorithm by using the message and a hash value computed from the message to compute the digital signature (digital signature with message recovery)



**Figure 7.34** Verifying a message that has been signed using the RSA algorithm with part of the message being replaced by the signature (digital signature with message recovery)

The task of the smart card in this scenario is very simple. It stores at minimum the private RSA key, and it decrypts the hash value computed from the message, which means it generates the signature. Everything else, such as generating the hash value or subsequently checking the signature, can in principle be performed equally well by a PC.

However, the ideal solution would be for the smart card to receive the message via its interface, compute the hash value, and then send the signed message back to the terminal. The smart card could also verify the signature. Although this scenario does not provide more security than the scenario in which the smart card only computes the signature, it is significantly more application-friendly, since the hash algorithms and RSA key can be changed by simply using a different smart card without any need to modify any programs or data in the PC.

In these two examples, the keys used to generate and verify digital signatures are global, which means they are the same for every smart card in a particular system. If it is necessary to alter this situation for security reasons so that each card has its own key for digital signatures, methods such as those described in Section 7.7, ‘Key Management’, on page 180 must be used.

The RSA and EC algorithms are not the only algorithms that can be used to generate digital signatures. There is also a cryptographic algorithm called the Digital Signature Algorithm (DSA) that was specifically developed for this purpose. It was first proposed by the US National Institute of Standards and Technology (NIST) in 1991, and it can be used for both signature generation and signature verification. Unlike the RSA algorithm, DSA was designed to prevent it from being used for data encryption or decryption, although it has since been

shown that it can be used for this purpose. In terms of exports and international use, this design limitation gives DSA an advantage over the RSA algorithm and other algorithms based on elliptic curves, which are subject to strong export restrictions.

## 7.6 CERTIFICATES

A not inconsiderable problem quickly arises in connection with the use of digital signatures: anyone who wants to check the digital signature of a message needs the corresponding public key. However, the public key cannot simply be sent without any protection, as otherwise the recipient cannot verify the authenticity of the key. The public key must therefore be signed by a trustworthy entity so that its authenticity can be verified. This entity is called a certification authority (CA). The combination of a public key that has been signed by a certification authority, the accompanying digital signature, and certain additional parameters is called a certificate, and the process of generating and verifying signatures with the aid of certificates is illustrated in Figure 7.35.

There is also another entity involved in this process, which is called a trust center (TC). A trust center generates and manages certificates and associated blacklists, and it can optionally generate keys for digital signature cards. As a rule, a trust center also maintains a public directory of certificates, so that anyone who wants to check a signed message can request the corresponding signed public key from the center, for example via the Internet.

A certificate contains not only the signed public key, but also a large number of additional parameters and options, since it must be possible to verify the public key of a certificate without using any other information. This means that among other things, the algorithms used to generate the hash value and the signature must be unambiguously specified. In theory, anyone who signs documents could specify their own certificate structure. However, this would make certificates noninterchangeable, which would generally negate their significance because interchangeability is one of the essential properties of certificates.

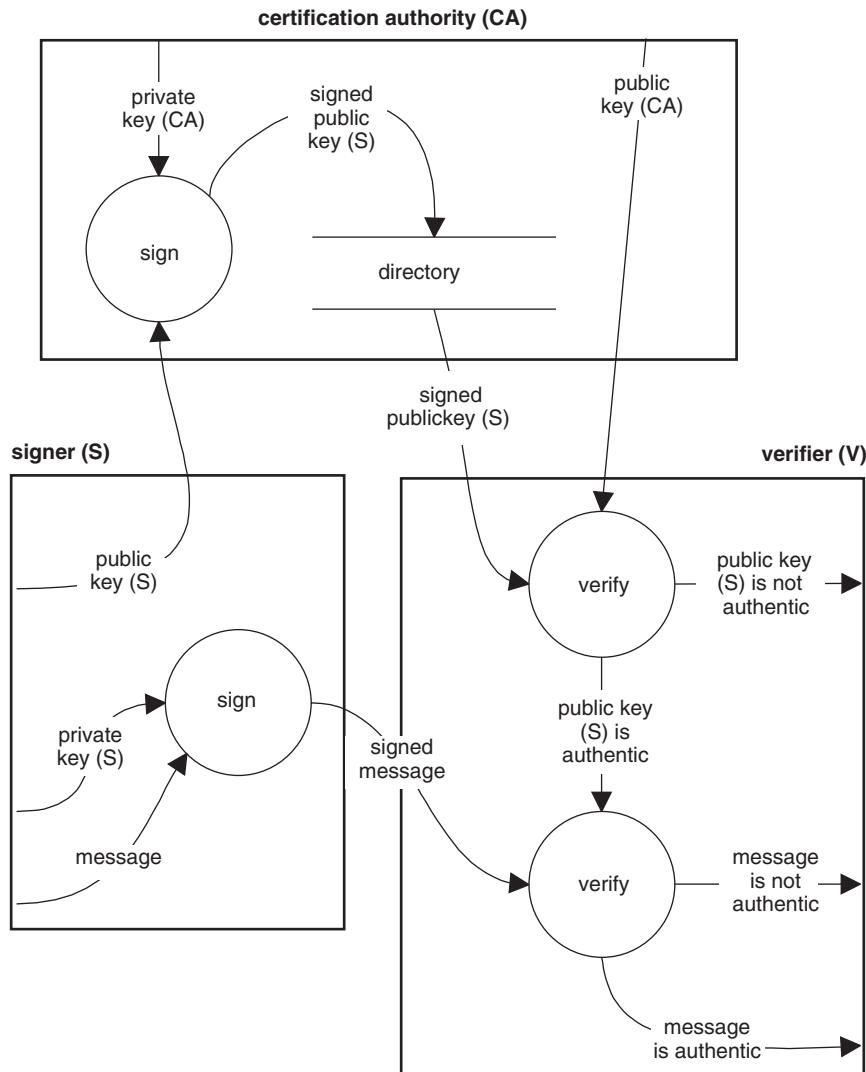
To make this cooperation possible, standards that specify the structure of certificates have been generated. The best known of the relevant standards is X.509, which specifies the structure and coding of certificates. It has also joined the ranks of ISO/IEC standards as ISO/IEC 9594-8.

The wide-ranging X.509 standard is a framework in which the structure of certificates is defined in detail and in unambiguous terms. It forms the basis for many digital signature applications. Some examples that can be mentioned here are the Internet Secure Socket Layer (SSL) mechanism and applications such as Privacy Enhanced Mail (PEM), Secure Multipurpose Internet Mail Extensions (S/MIME), and Secure Electronic Transaction (SET).

ASN.1 is consistently used in X.509 to describe certificates, and the widely known TLV coding scheme according to the Distinguished Encoding Rules (DER) is used for the actual coding.<sup>18</sup> Some data objects that may be included in a certificate are described succinctly in Table 7.14 on page 180. A brief introduction to and overview of X.509 certificates can be found in a paper by Peter Gutmann [Gutmann 98b].

Many optional data fields for a wide variety of applications are defined for X.509 certificates. For example, it is easily possible to include several public keys in a single certificate and to have

<sup>18</sup> See also Section 6.1, ‘Data Structures’, on page 109



**Figure 7.35** Data flow diagram of the basic processes for generating and checking a transmitted message using a certificate. The certificate is generated by a certification authority and contains the public key of the signer and the signature of the certification authority

them signed by different certification authorities. This can result in certificates containing several kilobytes of data, which is fairly large if a smart card is supposed to be used to check the certificate. However, this mechanism can also be used to generate items such as mutual certificates and tree-shaped certificate hierarchies. A typical X.509 certificate in a smart card usually has a size of approximately 1 KB.

**Table 7.14** Typical structure of an X.509 certificate

Data object and X.509 designation	Explanation
Version	Identifies the version of X.509 that defines the data objects of the certificate. This is usually version 3.
Serial number	The serial number of the certificate. This must be assigned by the issuer of the certificate to ensure that it is unique.
Signature algorithm identifier	Identifies the cryptographic algorithm used for the digital signature.
Issuer name	The name of the issuer of the certificate. In analogy to X.500, the designation of this name is unique in the entire world.
Validity period	The period during which the certificate is valid.
Subject name	The name of the entity whose public key should be recognized as authentic based on this certificate. In analogy to X.500, the designation of this name is unique in the entire world.
Public key	The public key of the subject entity, which the certificate is intended to identify as authentic.
Signature	The digital signature of the certificate data.

## 7.7 KEY MANAGEMENT

Key management in large systems involves many different tasks, as shown in Figure 7.36.<sup>19</sup> The life cycle of keys begins with key generation. The steps used in the key generation process must always be secure against replication by potential attackers. For this reason, physical random number generators that are by nature nondeterministic are often used to generate the initial data for key generation algorithms.

After being generated, the keys must be stored and distributed according to the system architecture. Keys are stored in hardware security modules (HSMs) that are suitably hardened against attacks, or in smart cards during the personalization process.<sup>20</sup>

Key management also includes updating and renewing specific keys. The purpose of these activities is to avoid having keys remain in use too long, since attackers can certainly take advantage of situations in which the same keys are used for a long time.

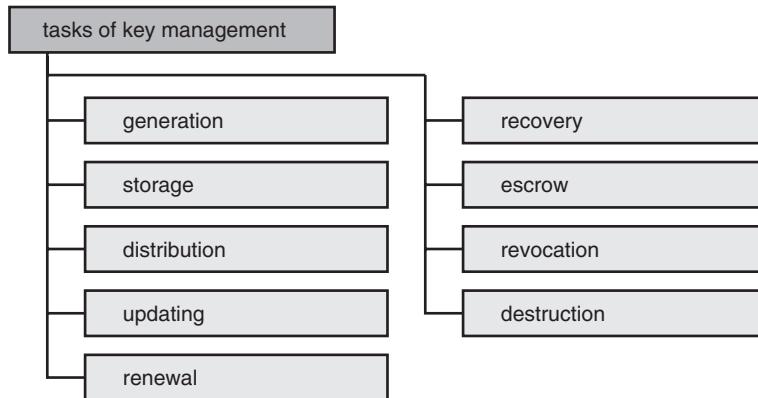
Recovery of lost keys, such as keys in defective smart cards, is also one of the usual tasks of key management. A related function is key escrow, which enables government organizations to access the secret keys of external parties if necessary.

Key revocation if keys become compromised and key destruction at the end of the life cycle of keys are the final two basic tasks of key management. These tasks are handled by the background system in most smart card systems. Consequently, the following description of key management concentrates on the tasks directly related to smart cards.

The paramount objective of all management principles for handling keys used with cryptographic algorithms is to minimize the impact on the overall system and the smart card application in the event that one or more secret keys become known. If it could be guaranteed

<sup>19</sup> See [Horster 99]

<sup>20</sup> See also Section 14.8.2, ‘Personalization (individualization)’, on page 619



**Figure 7.36** The basic functions of key management

that the keys would always remain confidential, a single secret key for each smart card would be sufficient. However, nobody can provide this guarantee.

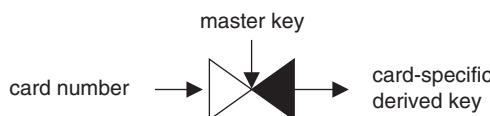
The number of keys increases dramatically as a result of the principles described here for enhancing the security of keys used with cryptographic algorithms. If all the principles and methods described here are implemented in a smart card, the keys may occupy more than half of the memory available for user data.

However, the choice of methods and principles that are used depends on the application, and it is not always necessary to use all of them. For example, there is no imperative need to support multiple key generations if the card is valid for only a short time, since the extra management effort and memory space would not be justified.

### 7.7.1 Derived keys

Unlike terminals, smart cards can be taken anywhere by anyone and analyzed at any desired effort and expense, so they are naturally exposed to the most severe attacks. If no master keys are present in the cards, the consequences of a successful attempt to read the card contents can be minimized. Consequently, only keys that have been derived from a master key are present in smart cards.

Derived keys are generated using cryptographic algorithms. The inputs are a card-specific feature and a master key. The commonly used algorithms are triple DES and AES. For the sake of simplicity, the specific feature is usually the card number. This number, which is generated when the card is produced, is unique in the entire system and can be used throughout the system to identify the card. Each derived key is thus unique. Figure 7.37 shows a method that can be used to generate derived keys.



**Figure 7.37** Deriving a card-specific symmetric key from the card number and a master key

## 7.7.2 Key diversification

In order to minimize the impact of a key being compromised, a separate key is often used for each cryptographic function. For example, different keys can be used for signatures, secure data transmission, authentication, and data encryption. This requires a separate master key for each type of key, so that the individual keys can be derived.

## 7.7.3 Key versions

It is usually not sufficient to use only one key generation for the lifetime of the smart card. For example, suppose the value of a master key could be determined as the result of a successful attack. This would mean that all application suppliers would have to shut down their systems and the card issuer would have to replace all the smart cards in the system. The resulting loss would be enormous. Consequently, all modern systems support multiple key generations.

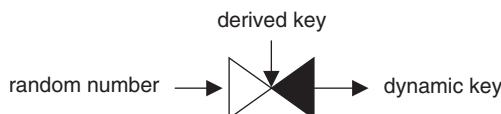
In the worst case, switching to a new key generation may be compelled by a successful attack on a key, but switching to a new generation can also occur routinely at fixed or variable intervals. The result of switching to a new generation is that all the keys in the system are replaced by new ones, without any need for card recall. All that is needed to supply new secret keys to a card is a secure data exchange with a terminal, since the master keys are located in the terminals and the higher-level components of the system.

## 7.7.4 Dynamic keys

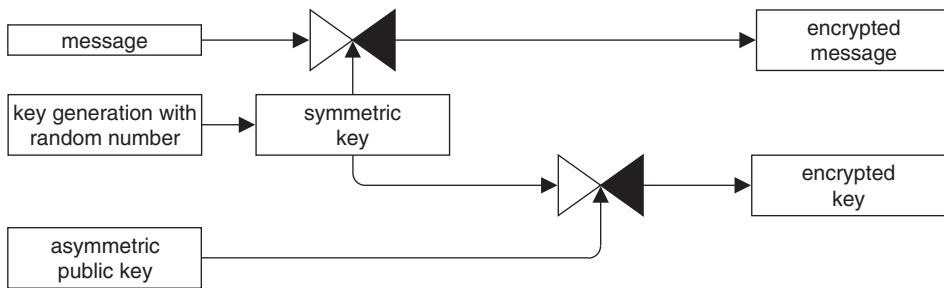
In many applications, and in particular for secure data transmission, it is common practice to use dynamic keys. Such keys are also called session keys or temporary keys. To generate a dynamic key, one of the two communicating parties first generates a random number or some other value specific to an individual session and sends it to the other party. The rest of the process varies, depending on whether the cryptographic algorithms that are used are exclusively symmetric or a combination of symmetric and asymmetric.

### 7.7.4.1 Generation with a symmetric cryptographic algorithm

In methods that use only symmetric cryptographic algorithms, the random number generated by the first party is sent as plaintext to the other party. The smart card and the terminal encrypt this number using a derived key. The result, as illustrated in Figure 7.38, is a key that is valid only for one session.



**Figure 7.38** Using a random number and a derived symmetric key to generate a dynamic key



**Figure 7.39** Example key exchange method using a combination of symmetric and asymmetric cryptographic algorithms. An encrypted dynamic symmetric key is first generated and then exchanged between two parties using an asymmetric cryptographic algorithm. Before this happens, a key pair for the asymmetric cryptographic algorithm is generated separately in a process not shown here

The main advantage of dynamic keys is that a different key is used in each session, which makes attacks significantly more difficult. However, dynamic keys must be used with caution for signature generation, since the same key will be needed for signature verification, and this key can only be generated with the same random number used to generate the signature. This means that if a dynamic key is used to generate a signature, the random number used to generate the key must be saved for use in verification, which means it must be stored. The ANSI X 9.17 standard specifies a somewhat more elaborate method for generating derived keys and dynamic keys. It is widely used in payment systems. This method requires two inputs: a time-independent and session-independent parameter  $T_i$  and a key  $\text{Key}_{gen}$  that is reserved for generating new keys. The resulting initial key  $\text{Key}_i$  can be used to compute as many new keys as desired. This key generation method has the additional advantage that it is a one-way process, which means that the computations cannot be reversed.

$$\text{Key}_{i+1} = e(\text{Key}_{gen}, e(\text{Key}_{gen}, (T_i \text{ XOR } \text{Key}_i)))$$

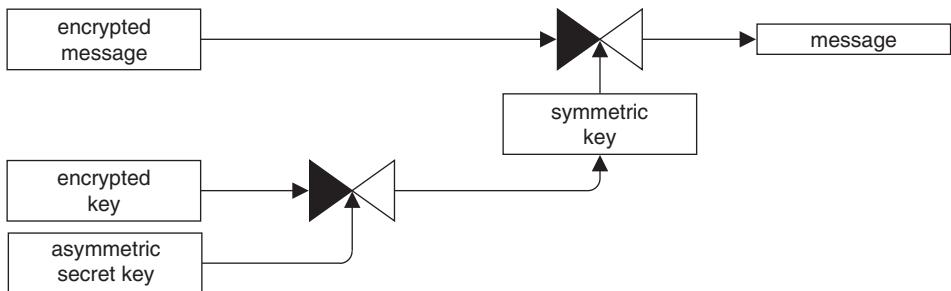
#### 7.7.4.2 Generation with an asymmetric cryptographic algorithm

Figures 7.39 and 7.40 on the following page illustrate a method for generating and subsequently exchanging a symmetric dynamic key for message encryption. An asymmetric cryptographic algorithm, such as RSA or DSS, is used for key exchange.

A similar method is used in PGP, for example, which employs the IDEA and RSA cryptographic algorithms. The basic advantage of this hybrid method is that the bulk of the data can be encrypted using a symmetric cryptographic algorithm, which has significantly higher throughput than an asymmetric algorithm.

#### 7.7.5 Key data

The mechanism used for external access to the keys stored in a smart card should be as simple as possible. In addition, the smart card operating system must ensure that the keys can only be



**Figure 7.40** Example key exchange process using a combination of symmetric and asymmetric cryptographic algorithms. A previously encrypted dynamic symmetric key (see Figure 7.39 on the preceding page) is recovered using an asymmetric cryptographic algorithm. Before this happens, a key pair for the asymmetric cryptographic algorithm is generated separately in a process not shown here

**Table 7.15** Typical key data stored in smart cards

Data object	Remarks
Key number	Key reference number; unique within the key file
Version number	Version number of the key, which may affect key derivation
Intended use	Indicates the cryptographic algorithms and methods for which the key is intended to be used
Blocked	Can be used to temporarily or permanently block use of the key
Retry counter	Counts unsuccessful attempts to use the key in a cryptographic method
Maximum retry count	The key is blocked if the retry count reaches this value
Key size	The size of the key
Key	The actual key

used for their intended purpose. For example, it must prevent the use of an authentication key for encrypting data. In addition to the intended use, the key number must be known in order to access a key. This number is the actual reference to the key. The version number is also needed in order to access a specific key. The typical key data stored in smart cards is listed in Table 7.15.

Some smart card operating systems increment the value of a retry counter assigned to a key on each unsuccessful attempt to use the key for some purpose, such as authentication. This provides fairly reliable protection against trial-and-error attacks on the key, although this form of attack is not a serious threat due to the long processing time in the smart card. If the retry counter reaches its maximum value, the key is blocked and cannot be used any more. The retry counter is reset to zero when an attempt to use the key is successful. This mechanism must always be used with considerable caution because an incorrect master key in a terminal could easily lead to massive card failures. As a rule, the retry counter can only be reset by a special terminal, and the identity of the cardholder must first be verified.

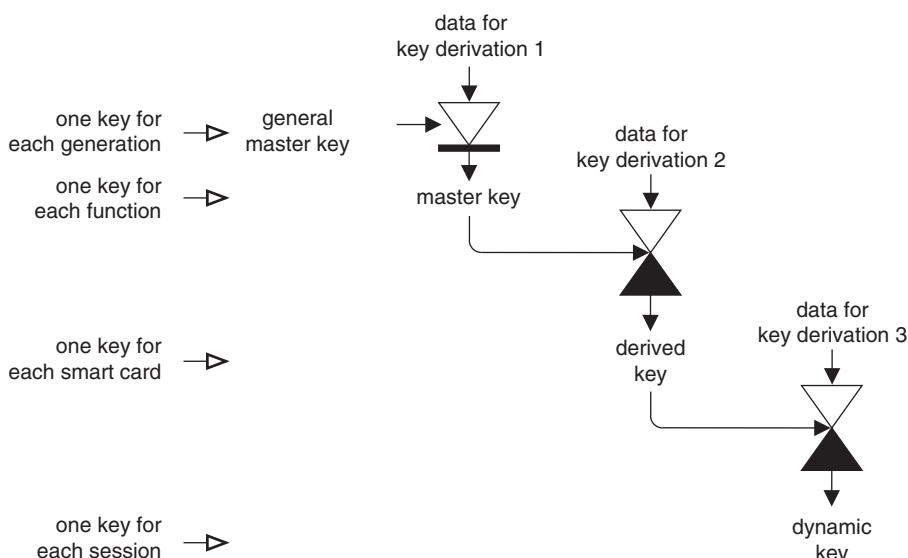
Some systems prohibit further use of expired key versions. This is accomplished by including a blocking parameter in the key data, which is activated when a new key with the same key number is accessed.

### 7.7.6 Key management example

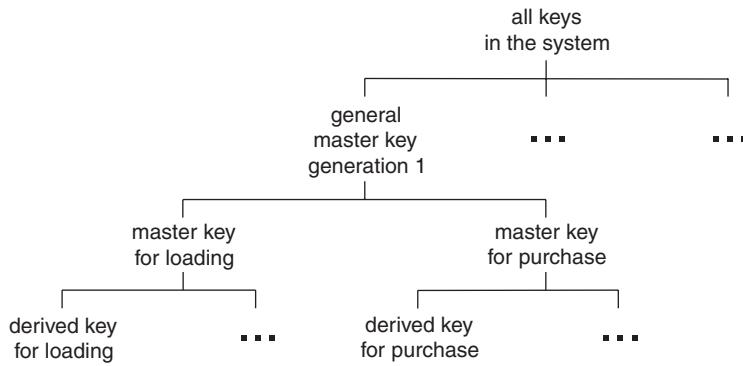
The following description of key management in a system based on smart cards is intended to illustrate the previously described principles by means of a general example. Large real systems often have layered structures that are significantly more complex than this example. Many small systems use a secret global key for all cards and do not have any form of key hierarchy. The system described here lies somewhere in between systems with very simple structures and large systems, which makes it a useful example.

In this example, the keys can be used with symmetric cryptographic algorithms for loading an electronic purse and making payments with the purse. These keys are evidently important in the system, since they are relatively well protected by the described key hierarchy. The individual key derivation operations are not explained in detail here, but in any case the AES or triple DES algorithm could be used for this purpose. The various keys can certainly have different lengths, although this is not described in detail here. For security reasons, the keys at the top of the hierarchy are usually derived using more powerful cryptographic algorithms than the lower-level keys.

Figure 7.41 shows an example of a key hierarchy in a system based on smart cards with symmetric cryptographic algorithms. The key at the very top of the hierarchy is called the general master key. There is only one such key for an entire generation of keys. A generation may remain valid for a year, for example, and be replaced in the following year by a new generation, which means switching to a new generation of the general master key. This key is the most sensitive key of the system with regard to security. If it becomes known, all the keys of its generation can be computed, and the system is insecure for one key generation. The general master key can be generated from a random number. It is also conceivable to construct the general master key from the results of dice thrown by several independent



**Figure 7.41** An example of a key hierarchy in a system based on smart cards with symmetric cryptographic algorithms



**Figure 7.42** Examples of keys in an electronic purse system with two functions: loading and purchasing. Only the stored keys are shown here; the keys that are generated dynamically for individual sessions are omitted for the sake of simplicity

persons, each of whom consequently knows only part of the key. The general master key should never be completely known by any single person, and generation of this key must never be reproducible.

Individual master keys for various functions are derived independently from the general master key. These keys may be used for functions such as loading or paying with an electronic purse. In this example a one-way function, such as a modified AES algorithm, is used to derive the individual master keys for the various functions. This makes it impossible to determine the general master key from a master key by reversing the process. Otherwise the general master key could be computed if, despite all security measures, a master key becomes known and the derivation data is also known. A one-way function is used here because it is assumed that in this imaginary purse system the master keys are located in the security modules of the terminals. In terms of system security, they are much more vulnerable to attack than the general master key, which never leaves the background system.

The derived keys form the next level of the key hierarchy. These keys are located in the smart cards. Each card contains sets of derived keys separated according to their intended use and generation number. When the card is used with a terminal, the terminal can use the key derivation data to compute the appropriate derived key for the transaction concerned. Naturally, the terminal first reads the derivation data from the card. Once a derived key is available, the next step is to compute a session-specific dynamic key, which is valid only for the duration of one session. In typical smart card applications, the session duration ranges from a few hundred milliseconds to a few seconds. The dynamic key is no longer used after the end of the session.

This example may appear elaborate and complex at first glance, but it is relatively simple compared with real systems. The objective of this example is to show exactly how all the keys in a system can be generated. It also implicitly shows what measures must be taken if a key becomes known. If the general master key becomes known, the system must switch to a new generation in order to allow continued operation without concerns about security risks. If a master key becomes known, only the keys derived from this key need to be blocked or replaced by a new generation. If a derived key becomes known, only the card in question needs to be

blocked; any other changes to the key system would surely be inappropriate. Of course, all of these measures presume that the reason why one or more keys have become known can be determined, so that it can be prevented in the future.

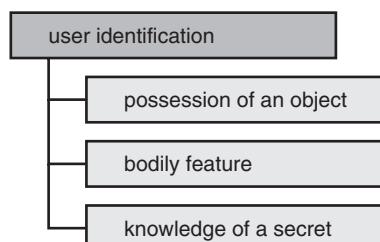
With this key hierarchy, it is clear that a large number of keys must be generated and stored in the smart cards. Of course, it is always possible to assign a single key to several functions in order to save memory space. It is also quite conceivable to use a different key hierarchy, which naturally depends strongly on the system for which the key management system is developed.

One of the main advantages of smart cards compared with other data storage media, such as magnetic-stripe cards and diskettes, is that they can provide secure, confidential data storage. An essential requirement for this is chip hardware that is designed and optimized for this purpose, along with suitable cryptographic algorithms for protecting confidential data. However, security depends on more than just special microcontroller hardware and the algorithms implemented in the operating system software. The security of the smart card application and the design principles used in the system development process are also of fundamental importance. This chapter provides a synopsis of the essential principles, methods and strategies for generating secure smart cards and secure smart card applications.

## 7.8 IDENTIFICATION OF PERSONS

Since ancient times, a variety of techniques have been used for the unambiguous identification of persons. As shown in Figure 7.43, there are three techniques for identifying a person: possession of an object, bodily features, or knowledge of a secret. The simplest form of identification is an identity card bearing a photograph or a signature written in the presence of the examiner. The photograph on an identity card can be compared with the actual person, with the result being an assessment of the genuineness of the person's identity.

In the information technology context, this comparison is not so easy because it must be performed by a computer instead of another person. Despite their success in performing mindless tasks, computers still have tremendous difficulty in performing intelligent activities. Consequently, entering a password via a keypad has generally become the preferred identification method. The effort needed for the comparison is minimal, since essentially all the computer has to do is to compare the entered password with a stored reference value and make a simple yes/no decision. Password comparison effectively amounts to making a decision regarding the genuineness of the identity of the person being checked.



**Figure 7.43** Methods for identifying a person

### 7.8.1 Knowledge-based identification

There are basically only three different ways to identify a person. If a password is used, what is tested is whether the person knows a particular secret. If the answer is yes, the conclusion is that the person is who he or she claims to be. The second option is to test whether a person possesses a particular object. The third possibility is to test specific, unique biometric features of the person.

Methods that rely on knowing a secret or possessing a particular item have a significant drawback, which is that the person to be identified must either remember something or carry something on his or her person. Depending on the situation, the fact that the secret or object can be passed to another person can be considered an advantage or a disadvantage. In any case, it is not possible to unambiguously ascertain that the person holding the secret or the object is truly its legitimate owner, rather than someone else who may have illegitimately acquired the secret or object that is tested.

The third identification method eliminates the problem of transferability, since it is based on using specific features of the human body for identification purposes. Of course, measuring these features is difficult in most cases, since for obvious reasons biometric features that can be easily measured, such as height or weight, cannot be used.

It is easier to understand these three possible identification methods if you consider the following example. Suppose you have to meet an unknown person at the train station. As soon as you see a possible candidate, you have the problem of deciding whether he is really the person you are looking for. However, if the unknown person shows up at the right place and the right time, this actually amounts to an implicit test of a secret, since you can at least hope that the place and time of your meeting are not generally known. An explicit test of a secret would occur if the unknown person were to utter a password that is known only to you and him. Alternatively, he could identify himself by means of an item that he possesses, for example by holding a newspaper printed on a specific day under his arm. Of course, the most secure method would be to check the person for a specific bodily feature.

This train station example clearly shows that identifying an unknown person can be regarded as a classic problem that occurs in everyday life as well as in spy novels, rather than just being limited to computers and smart cards.

It has now become common practice to enter PIN codes in many types of automated equipment and computers. The resulting sharp increase in the number of PIN codes used for various purposes makes it very difficult for ordinary people to keep track of all of their PINs. After all, who can remember twenty or more different PINs? The security and reputation of a system are naturally not improved if every user jots down the PIN code on the card, since the incidence of fraud will increase by leaps and bounds. For these reasons, a desire to use other identification methods in place of PIN codes has arisen in recent years. Biometric features that allow a particular person to be identified unambiguously by a machine are ideal for this purpose.

### 7.8.2 Testing a secret number

The most commonly used method of user identification is entering a secret number, which is generally called a PIN code (PIN is the abbreviation of ‘personal identification number’), or in technical terms ‘cardholder verification’ (CHV).

A PIN code is usually a four-digit number, commonly composed of the numerals ‘0’ to ‘9’. The reason for using a purely numeric code is simply that card terminals generally only have numeric keypads. However, in principle any desired sequence of alphanumeric characters could be used. The PIN code is entered using a terminal keypad or a computer keyboard, and then sent to the smart card. The smart card compares the supplied value with an internally stored reference value and reports the result to the terminal.

Particularly in payment applications, PIN entry is regarded as relevant to security, so requirements relating the nature of the keypad are frequently found in this application area. Special keypads that satisfy these requirements are often called PIN pads. In Germany, for example, there is a requirement (from the ZKA) that PIN codes for Eurocheque cards can only be entered using a keypad having special mechanical and cryptographic protection. These PIN pads have all the features of a security module, such as case-opening sensors and foils to protect against drilling, and they encrypt the PIN directly as it is entered. This provides reliable protection against tampering with a keypad in order to intercept PIN codes while they are being entered.

A distinction can be made between static and modifiable PINs. A static PIN cannot be changed by the user, so it must be memorized by the user. If it becomes known, the user should destroy the card and obtain a new one with a different static PIN. A modifiable PIN can be altered as desired by the user or changed to a number that the user finds easy to remember. There is a danger in this, since the numbers that many people find easy to remember are ones such as ‘1234’, ‘0711’, and ‘0815’. Smart cards do not check for such trivial numbers, since there is not enough memory available to store the necessary table. However, it would be perfectly conceivable for terminals to prohibit changing the PIN code to a number of this sort. For security reasons, the current PIN code must always be entered before it can be changed, as otherwise attackers could replace every existing PIN with their own.

The situation is different with personal unblocking keys (PUKs), which are also called ‘super PINs’. They usually have more digits than a normal PIN (a typical value is six), and they are used to reset the retry counter of a PIN to zero if it has reached its maximum value. A new PIN code is also stored in the card when the PUK is used, since resetting the retry counter is of little use if the user has forgotten the PIN code, which is most often the reason why the retry counter reaches its maximum value.

There are also applications that use transport PINs. In this case, the smart card is personalized with a random PIN and the cardholder receives the PIN code by letter. The cardholder replaces the PIN entered during card personalization with a PIN of his or her choice before actually using the card. In a similar method called the null PIN method, the card is preloaded with a trivial PIN such as ‘0000’ and the smart card again forces the user to change the PIN before using the card. Both methods prevent the subsequent use of PIN codes that may have been spied out during card personalization or handling.

According to a recommendation of the ISO 9564-1 standard, a PIN should consist of four to twelve alphanumeric characters in order to minimize the probability of discovering the PIN code by pure trial and error. However, actual practice is often somewhat different. Entering nonnumeric characters is technically impossible in many locations because the keypad has only numeric characters.

The number of characters in a PIN depends not only on the desired level of security, but also to a large extent on the memory abilities of average card users. Over the course of many years, people have grown accustomed to using four-digit PINs, so changing to PINs with six or more digits would be very difficult. In practice, the anticipated improvement in security

provided by using six-digit or eight-digit PINs could also prove to be purely theoretical. Many people find it difficult to remember numbers of this length, especially if they do not use them very often, and consequently write them down on the card or a slip of paper kept near the card. In such cases, the level of security with a long PIN is significantly lower than with a short PIN.

The otherwise reasonable insistence on periodically changing PIN codes suffers from a similar drawback. It may work in high-security applications having only a few users, but it has fatal consequences for the acceptance of mass-market applications, which must employ the simplest possible methods in order to accommodate people with poor memories.

In this connection, there is also another very important issue. In many cases, entering and verifying a PIN does more than just identify the user and show that the user has rightful possession of the card; it also forms a declaration of intent. By entering the PIN code, the user effectively agrees to a particular transaction. A good example is entering a PIN code at a cash dispenser. In addition to identifying the card user by means of his or her knowledge of the secret PIN, it represents a declaration that the user agrees to have a certain amount of cash paid out from his or her account. This is very important with regard to certain biometric features, some of which can be tested without the explicit permission of the person concerned and do not necessarily represent a declaration of intent.

### 7.8.3 The probability of guessing a PIN

The simplest form of attack on a PIN, aside from watching it being entered, is simply guessing. The probability of success depends on the length of the PIN, the characters that can be used to compose the PIN, and how many guesses are allowed. The probability of correctly guessing a four-digit PIN in three tries is 0.03 %, which is not especially high. Two basic formulas for guessing PINs and similar passwords are presented here. They can be used in practice to estimate the risk associated with using a particular password.

$$x = m^n \quad (7.1)$$

$$p = \frac{i}{m^n} \quad (7.2)$$

Table 7.17 lists the characteristics of various types of PINs or passwords.

There is yet a fourth factor relating to guessing a PIN, which for a long time has been inexcusably neglected. This is the uniformity of the distribution of PIN codes in an application. It is much easier to guess a PIN if you know that certain PIN codes are more common than others. The significance of this important auxiliary condition became evident almost overnight.

**Table 7.16** Definitions and descriptions of the variables in Formulas (7.1) and (7.2)

Variable	Example	Description
$i$	3	number of guesses
$m$	10	number of possible characters per position
$n$	4	number of positions
$p$	0.0003 (0.03 %)	probability of guessing the password
$x$	10 000	number of possible passwords

**Table 7.17** PIN codes and passwords with various lengths and codings, and the number of possible combinations

Type of PIN or password	Range of values and coding of the PIN or password	Number of possible PINs or passwords
1-digit PIN	$\text{PIN} \in 0 \dots 9$	10
1-character password	$\text{password} \in 0 \dots 9, 'A' \dots 'Z'$	36
1-character password	$\text{password} \in 0 \dots 9, 'a' \dots 'z', 'A' \dots 'Z'$	62
1-character password	$\text{password} \in 0 \dots 9, 'a' \dots 'z', 'A' \dots 'Z', 20 \text{ arbitrary special characters}$	82
4-digit PIN, no leading zero	$\text{PIN} \in 1000 \dots 9999$	$9.00 \times 10^3$
4-digit PIN	$\text{PIN} \in 0000 \dots 9999$	$1.00 \times 10^4$
4-character password	$\text{password} \in 0 \dots 9, 'A' \dots 'Z'$	$1.68 \times 10^6$
4-character password	$\text{password} \in 0 \dots 9, 'a' \dots 'z', 'A' \dots 'Z'$	$1.48 \times 10^7$
4-character password	$\text{password} \in 0 \dots 9, 'a' \dots 'z', 'A' \dots 'Z', 20 \text{ arbitrary special characters}$	$4.52 \times 10^7$
5-digit PIN, no leading zero	$\text{PIN} \in 10\,000 \dots 99\,999$	$8.90 \times 10^4$
5-digit PIN	$\text{PIN} \in 00\,000 \dots 99\,999$	$1.00 \times 10^5$
6-digit PIN, no leading zero	$\text{PIN} \in 100\,000 \dots 999\,999$	$8.99 \times 10^5$
6-digit PIN	$\text{PIN} \in 000\,000 \dots 999\,999$	$1.00 \times 10^6$
6-character password	$\text{password} \in 0 \dots 9, 'A' \dots 'Z'$	$2.18 \times 10^9$
6-character password	$\text{password} \in 0 \dots 9, 'a' \dots 'z', 'A' \dots 'Z'$	$5.68 \times 10^{10}$
6-character password	$\text{password} \in 0 \dots 9, 'a' \dots 'z', 'A' \dots 'Z', 20 \text{ arbitrary special characters}$	$3.04 \times 10^{11}$

in 1977 in connection with German Eurocheque cards. Although the details of the method used to compute a PIN code from the data stored on the magnetic stripe of a Eurocheque card are still secret, a few general steps of the method became known. From this information, it could be concluded that the generated PIN codes are not uniformly distributed; the algorithm generates the numerals 0 to 5 significantly more often than the numerals 6 to 9. It also became known that the PIN computation algorithm suppresses leading zeros when generating PIN codes. With this nonuniform distribution, only 150 attempts are necessary to guess a four-digit PIN code if three incorrect guesses are allowed each time, instead of 3333 with a uniform distribution [Knapp 97]. With 10.5 % of the cards, the distribution is so poor that only 72 attempts are necessary if the properties of the PIN generation method are taken into account [Schindler 97]. As a result, an improved PIN code generation algorithm is used with new Eurocheque cards, and the DES algorithm originally used has been replaced by the triple-DES algorithm.

#### 7.8.4 Generating PIN codes

In order to generate a PIN code for a smart card, it is only necessary to have a random number generator and an algorithm that converts a random number into an ASCII-coded PIN of the required length. A table of known trivial sequences can be used to detect and discard trivial PIN codes. Finally, the PIN code must be stored in the smart card, after which the VERIFY

command can be used as necessary to compare it with PIN codes sent to the card from the terminal.

A somewhat more complicated process for generating PIN codes is required in a system that uses magnetic-stripe cards, since it must be possible for a cash dispenser operating offline to check an entered PIN code using the data on the magnetic stripe. This does not apply directly to smart cards, but all debit cards (such as Eurocheque cards) currently in use have magnetic stripes for compatibility reasons, even if they also contain microcontrollers. If hybrid cards with chips and magnetic stripes are used, the PIN generation algorithm must be deterministic, which means that it must always produce the same result with a given set of input values. A random number generator cannot do this.

A method is thus needed that can generate a PIN code from the magnetic stripe data. To avoid having the security of the process depend entirely on the method, a secret key should also be involved in the computation.

Figure 7.44 on the next page illustrates a method similar to the one used for German Eurocheque cards. Its inputs consist of a bank routing code, an account number, and a card serial number. This method uses the DES algorithm with a secret key to generate a four-digit PIN code.

This method suffers from the previously mentioned shortcoming: it generates PIN codes that are not distributed uniformly over the entire number space ('0000' ... '9999' with four-digit PIN codes). This is due to the mapping rule used to convert the hexadecimal numerals ('A', 'B', 'C', 'D', 'E', and 'F') into decimal numerals after the encryption process. This undesirable property could be easily corrected by using a better mapping rule. The DES algorithm is used because the key rather than the procedure must be kept secret and because it has the properties of confusion and diffusion.<sup>21</sup>

The PIN generation method used between 1981 and 1997 for German Eurocheque cards produced PIN codes that were not uniformly distributed over the entire number space.<sup>22</sup> This meant that some PIN codes were significantly more probable than others, and this could be used as a basis for attacks, although they were generally not successful. For this reason, it is important to check methods used to generate PIN codes to ensure that the resulting PIN codes are distributed uniformly over the available number space.

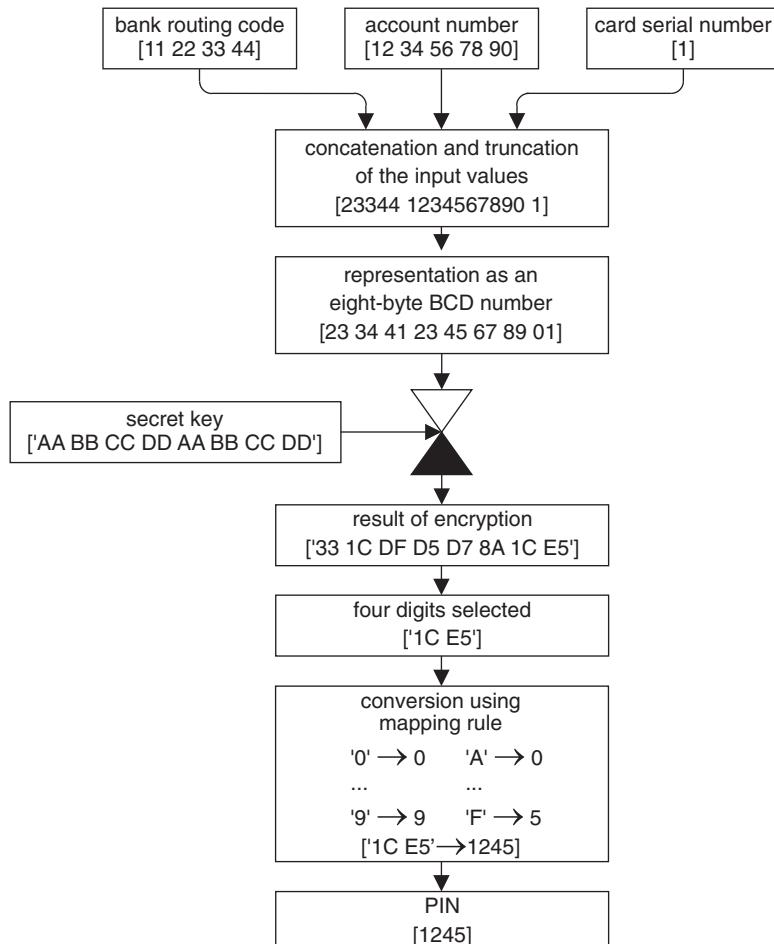
## 7.8.5 Verifying that a terminal is genuine

As is well known, PIN entry is used to verify the identity of the user. However, the user would also like to be able to verify that the terminal is genuine. For example, consider a dummy cash dispenser. Swindlers can use such machines to obtain PIN codes entered by unsuspecting users. If they then steal the users' cards, they can use them to withdraw money from the users' accounts. This is only possible because the cardholders cannot verify that the cash dispenser is genuine.

There is a simple method that can be used to defend against this form of attack. It involves storing a password in a file in the smart card. This password, which can be a name or a number chosen by the user, is known only to the card user and can be changed only by the user. The

<sup>21</sup> See also Section 7.1.1, 'Symmetric cryptographic algorithms', on page 138

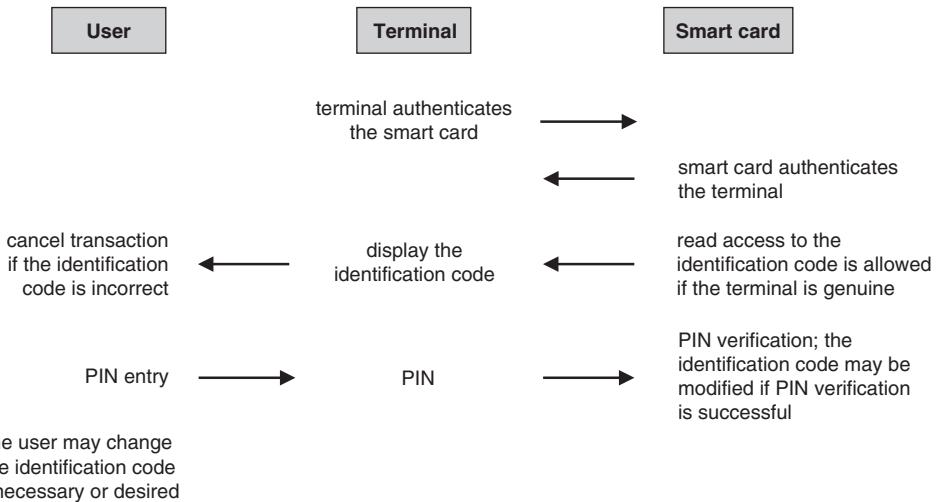
<sup>22</sup> See [Kuhn 97]



**Figure 7.44** Example of a method for generating four-digit PIN codes using the DES algorithm and three card-specific data items (bank routing code, account number, and card serial number). The illustrated method has the disadvantage that it produces PIN codes with a nonuniform distribution (some PIN codes occur more often than others) due to the mapping rule that is used. Sample values are shown in square brackets. This method is remotely similar to the method used for German Eurocheque cards and is based on two articles in *Die Datenschleuder* [Müller-Maguhn 97a, Müller-Maguhn 97b]

smart card operating system allows read access to this file only after the terminal has been authenticated by the card.

After the user inserts the smart card into the terminal, the first operation that is performed is mutual authentication of the smart card and the terminal. If this is successful, each party knows that the other party is genuine. The card then allows read access to the file containing the user's password, and the password is displayed on the terminal screen. The user sees his or her password and thus knows that the terminal is genuine, as otherwise it could not have accessed the file containing the user's password. Now the user can confidently enter his or her PIN code.



**Figure 7.45** A method for ensuring that a PIN code can only be entered in a genuine terminal. A prerequisite is that the user does not enter the PIN code until the right password has been displayed by the terminal

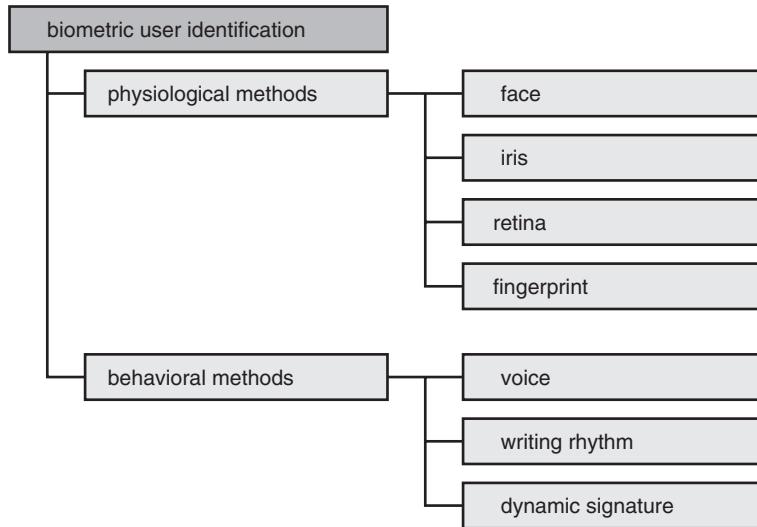
This method, which is illustrated in Figure 7.45, provides a simple way to prevent users from entering their PIN codes in manipulated terminals. Any desired word or number can be used as the password. It should be possible for the user to change the password as desired in order to prevent potential attackers from discovering it. This method can also be extended or modified as necessary to meet other requirements of a similar nature.

### 7.8.6 Biometric methods

The steadily increasing use of passwords and PINs leads to significant user resistance to this form of identification. Few people find it especially difficult to remember a few frequently used combinations of numbers or letters, but if a PIN code for a particular card is used only rarely, such as once every few months, most people find it difficult to remember the PIN code. The fear that the machine will confiscate the card if an incorrect PIN code is entered three times in a row only makes things worse.

This is certainly one of the main reasons why biometric methods are drawing increasing interest in many areas. They are not necessarily faster or more secure than PIN entry, but they can make things much easier for users. If the level of security provided by biometric methods is equivalent to that provided by PIN codes, system operators will also be willing to use them, especially because biometric features cannot be transferred to other people as easily as PIN codes. This means that what is identified is the actual person instead of a secret (the PIN code) shared by the user and the system operator.

A biometric identification method is a method that can unambiguously identify a person by means of unique, personal bodily features. As indicated in Figure 7.46, a distinction can be made between methods based on physiological features and methods based on behavioral features. If the features checked by the method are directly connected to the person's body and



**Figure 7.46** The most important biometric methods for user identification

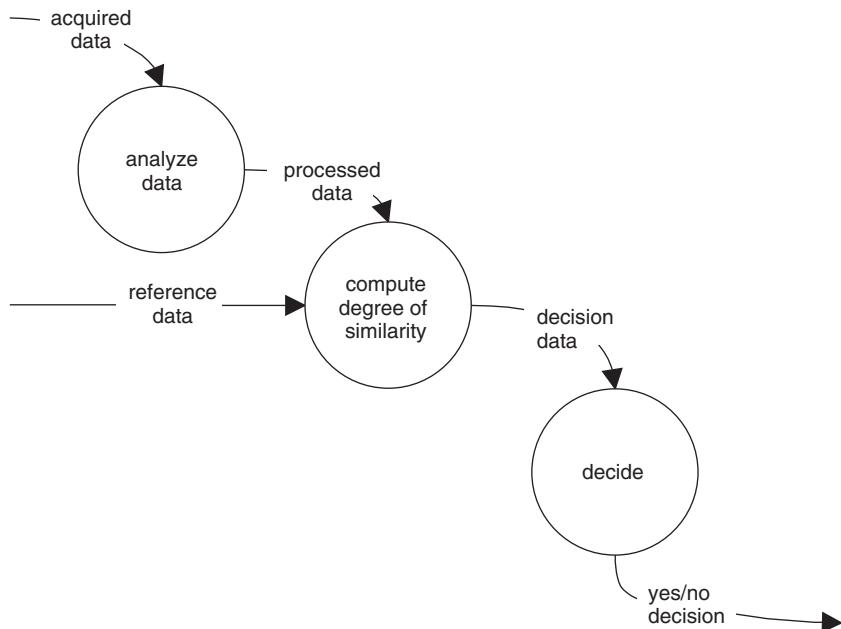
are fully independent of conscious behavior patterns, they are called physiological biometric features. Biometric methods based on behavioral features, by contrast, are based on certain features that can be consciously altered within certain limits but are nevertheless characteristic of a particular person.

An essential aspect of using biometric features for identification purposes is the question of user acceptance. If the method is similar to existing, well-known methods, users will be more willing to accept and use it. A typical example is a handwritten signature, which has been used for generations in almost all cultures for identification and for indicating agreement or consent.

Another factor that must be respected is concerns that users may have regarding medical and hygienic aspects. For instance, users may be afraid of acquiring a communicable disease from optical scanning of their retinas, or they may be afraid that the laser beam will damage their eyes. Although such fears may be entirely subjective and lack any scientific basis, they can strongly influence user behavior and ultimately affect user acceptance of the method. These aspects should be carefully considered before any sort of biometric method is used.

There is yet another difference between biometric and knowledge-based identification methods, which can be regarded as either an advantage or a disadvantage according to one's point of view. Biological features cannot be transferred to another person. Systems that use biometric methods for identification thus do not allow practices such as lending your card and your PIN code to someone you trust so they can use them as they see fit. System operators naturally find such actions utterly shocking, since revealing PIN codes is prohibited in almost all systems. However, nearly everyone knows how loosely such prohibitions are observed in practice.

Biometric features are usually not modifiable, which is precisely what makes them attractive for the unambiguous identification of persons. However, this nonmodifiability can certainly lead to major complications if a system is compromised. In combination with the fact that some biometric features can be recorded at reasonable effort and expense without the consent of the person concerned, this nonmodifiability can also lead to serious problems.



**Figure 7.47** Basic data flow for computer evaluation of a biometric feature

This can be clearly illustrated using fingerprints as an example. Suppose the fingerprints of a person are taken illicitly by scanning an object that the person held while eating in a restaurant, and the fingerprint data is then posted on the Internet so that anyone with suitable equipment can make copies of the fingerprints. For the rest this person's life, any identification based on his or her fingerprints as biometric features would be dubious because it would never be possible to be sure that a particular fingerprint actually originated directly from this person.

Biometric features can be classified as open, slightly concealed, concealed, or strongly concealed, according to the ease with which they can be acquired or viewed. Depending on the classification, they can be detected more or less easily by other parties without the consent of the person concerned. Open features, such as a person's face, can be acquired by simple observation. A slightly concealed feature, such as a fingerprint, can be acquired using simple tools or equipment without the awareness of the person concerned. Relatively elaborate equipment is needed to view retinal patterns, and it is nearly impossible to do so without the awareness of the person concerned, so retinal patterns are classified as concealed features. Strongly concealed features are often behavioral features, since in most cases they must be consciously revealed.

In addition to testing whether a person knows a particular secret, entering a PIN code is a legally binding equivalent of saying 'I consent'. This circumstance is very important if it is desired to use another method in place of PINs. If the retinal pattern of a user is scanned at a distance of several meters and used for identification, this can certainly not be regarded as representing the consent of the person concerned to any sort of action. In almost all countries, only a conscious manual action of the user can be regarded as a declaration of intent. For example, breaking the seal of a cardboard packet containing software is an unambiguous indication that the user agrees to the license conditions printed on the packet.

Biometric methods based on fully passive examination of the person in question must therefore be supplemented by suitable user guidance regarding the element of declaration of intent.

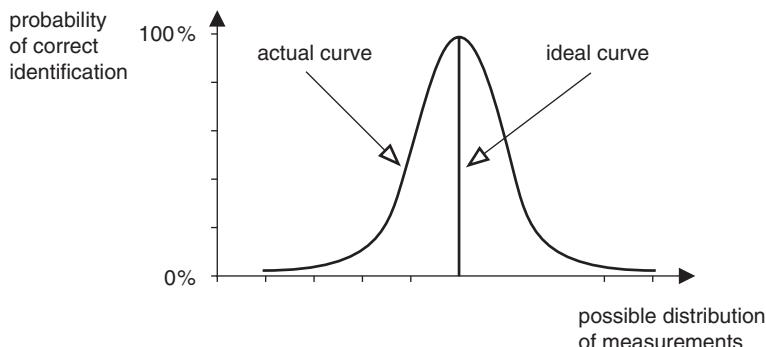
Naturally, not all biometric features are suitable for personal identification. A feature must satisfy at least the following criteria before it can be reasonably used:

- measuring the feature is technically feasible in terms of the method used, the time required, and the cost;
- it must be possible to unambiguously associate the feature with a particular person;
- the feature must be widely distributed in the population;
- it must not be possible to alter the feature with fraudulent intent;
- the volume of reference data generated must be small (a few hundred bytes to at most several thousand bytes);
- natural changes to the feature over time must be so small that the feature can always be measured correctly;
- the selected measuring method and feature must be acceptable to the users.

With any sort of measurement, the results are not always the same, but instead vary from one measurement to the next. This occurs with even the simplest forms of measurement. For example, if you measure the width of a sheet of paper several times, each result will be slightly different. There are many reasons for this, but it does not create difficulties in practice because the average value of the measurements will approximate the true value.

Experience shows that the amount of variation between individual measurements depends on the difficulty of making the measurement. For example, it is significantly easier to measure the weight of a bar of chocolate than to measure the distance between the earth and the moon. Measurements performed on human beings are always difficult and have a wide range of variation.

Figure 7.48 shows an example of the results of measuring a biological feature, such as the length of a finger. The extent of variation in the measured values is plotted on the horizontal axis, while the probability of correct identification based on the measured biometric feature is

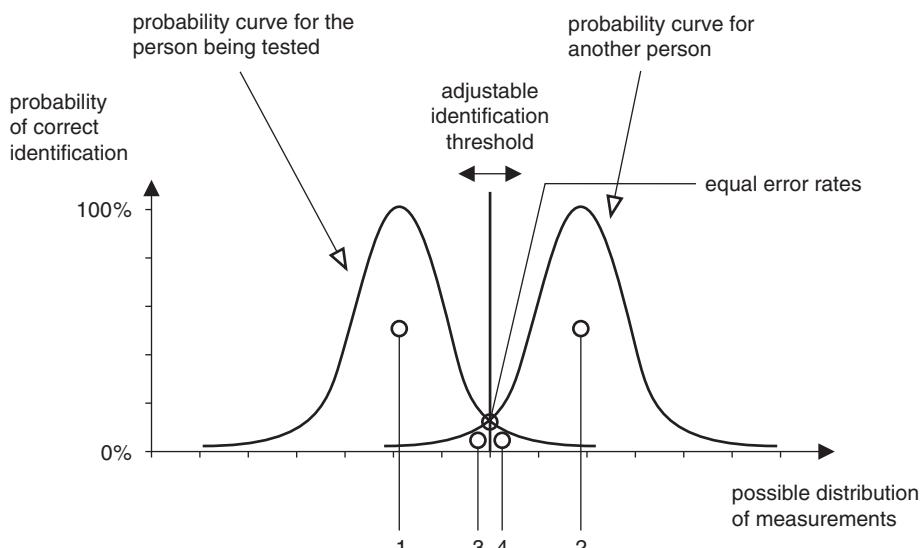


**Figure 7.48** Probability distribution of repeated measurements of a biometric feature of a person

indicated on the vertical axis. With an ideal biometric feature and an ideal measuring method, there would be no variation, and the curve would be reduced to a vertical line. However, the combination of a real feature and a real method results in the Gaussian bell curve shown in the figure. If the actual measured value differs from the reference value, it is not possible to be absolutely certain that the person to be identified has been correctly recognized.

Figure 7.47 on page 196 depicts the basic data flow in systems that use biometric features for identification. Before a biometric feature can be tested, the feature of the person concerned must be acquired. This can be done by making repeated measurements and calculating the average value. This yields a reference value, which is then stored in the smart card. After this, the smart card can be used to test whether the result of an actual measurement supplied to it matches the reference value. Depending on the biometric method used, it may first be necessary to use a powerful computer to process the actual measurement data into a form that the card can use for comparison. As identification cannot be established with absolute certainty, a threshold must be used to decide whether the person in question should be recognized as genuine. This level of this threshold depends on the specific method and application.

If we take the previously mentioned probability distribution and add a curve for a second person to it, we obtain the plot shown in Figure 7.49. The additional curve represents an arbitrary person whose measurement curve for the biometric feature concerned is close enough to that of the first person for it to affect the identity decision. As both curves approach the horizontal axis asymptotically, they have an intersection point. At this point, the probability of correct identification is equal to the probability of incorrect identification. Consequently, biometric identification systems use a sliding threshold level that indicates the probability above which identification is assumed to be correct. The threshold level shown in the figure divides the two curves into four regions. These regions indicate the decision to be taken regarding the identity of the person as deduced from the biometric feature.



**Figure 7.49** Probability distribution and decision regions for biometric feature measurement: (1) correct positive identification (true acceptance); (2) correct negative identification (true rejection) (3) incorrect positive identification (false acceptance); (4) incorrect negative identification (false rejection)

The key message of this figure is that a user can never be identified with absolute certainty. It is only possible to assume, with high probability, that the person has been identified correctly. The level of this probability can be set using the threshold value. However, in practice the threshold value cannot be set at any desired level because a high standard for correct identification results in a large number of false rejections.

The basic parameters for assessing a biometric method are its false acceptance rate (FAR) and its false rejection rate (FRR). The FAR is the probability of incorrect acceptance of the wrong person, while the FRR is the probability of incorrect rejection of the right person. Naturally, these two probabilities cannot be freely chosen; they are primarily properties of the biometric method that is used and can be adjusted only within certain limits. In addition, the FAR and FRR are mutually dependent in the sense that a low FRR causes a high FAR and vice versa.

For users, a high FRR means that they may be rejected despite presenting a legitimate feature, which naturally degrades user acceptance. For their part, system operators want to have not only a low FRR, but also a low FAR in order to prevent false positive identifications.

PIN testing does not require complex algorithms in the smart card, since it only involves comparing supplied and stored PIN codes. Unfortunately, things are not this easy with biometric features. The reference value is of course stored in the smart card, but the comparison with the actual measurement usually cannot be performed in the card, due to large amount of computing power needed to evaluate biometric features. Smart cards usually do not have adequate computing power for this, so compute-intensive preprocessing of the measured data is performed externally. The result is then sent to the smart card, which evaluates the preprocessed data using special algorithms that do not need large memory capacity or computing power and the card then makes a yes/no decision based on the stored reference value. This method is called oncard matching or matching on chip (MOC). The duration of the matching process depends on many factors, such as the biometric method used and data preprocessing (if any). For example, it takes approximately two seconds to test fingerprint data in a smart card with an eight-bit processor using data that has been preprocessed in the terminal.

Biometric features are personal data and should be protected accordingly. This represents a very good application for smart cards, since the reference data needed for testing never has to leave the card, which makes attacks significantly more difficult. If the reference data is instead stored in a nonsecure environment, it can be manipulated and read as desired. In this case a biometric identification method would not provide any significant advantage. The steadily increasing computing power of microcontrollers and the possibility of integrating sensors for biometric data acquisition into the card may open up new application areas for smart cards.

# 8

## Communication with Smart Cards

A prerequisite for all interactions between smart cards and terminals is communication between the two parties. With the conventional ISO/IEC T = 0 and T = 1 protocols, only one line is available for this. This electrical connection forms the basis for exchanging data between the card and the terminal. As there is only one line, the terminal and the card must take turns transmitting, with the opposite party acting as the receiver. This process of alternating transmission and reception is called half-duplex communication.

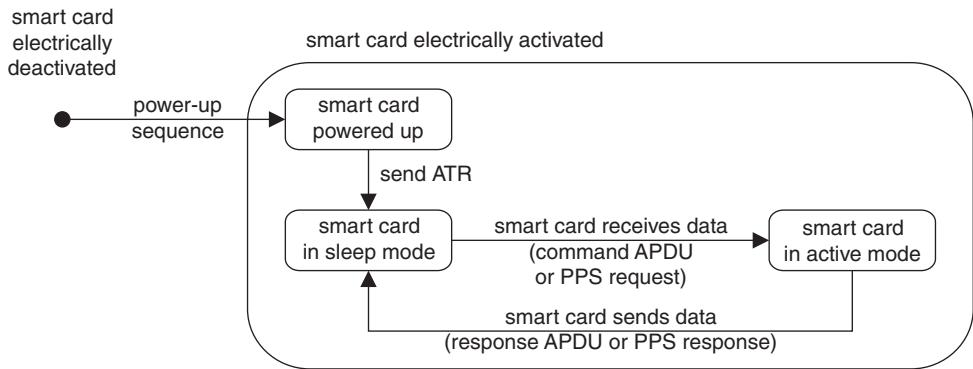
Full-duplex communication, in which both parties can transmit and receive concurrently, is not presently implemented in the smart card world. However, many smart card microcontrollers have two I/O ports, and two of the eight contacts are reserved for future use, such as a second I/O link or a USB interface, so a full-duplex link would be technically possible.

Communication with the card is always initiated by the terminal. The card always responds to commands received from the terminal, which means that the card never sends data without an external stimulus. This yields a pure master–slave relationship, with the terminal as master and the card as slave. The proactive command<sup>1</sup> method used with telecommunication smart cards, which allows the smart card to send commands to the terminal, is also based on the standard master–slave relationship. After the smart card has processed the command sent to it and sent the response to the terminal, it returns to the low-power sleep mode. In this mode, it can only be awakened by another command from the terminal. This is illustrated by the state diagram in Figure 8.1 on the following page.

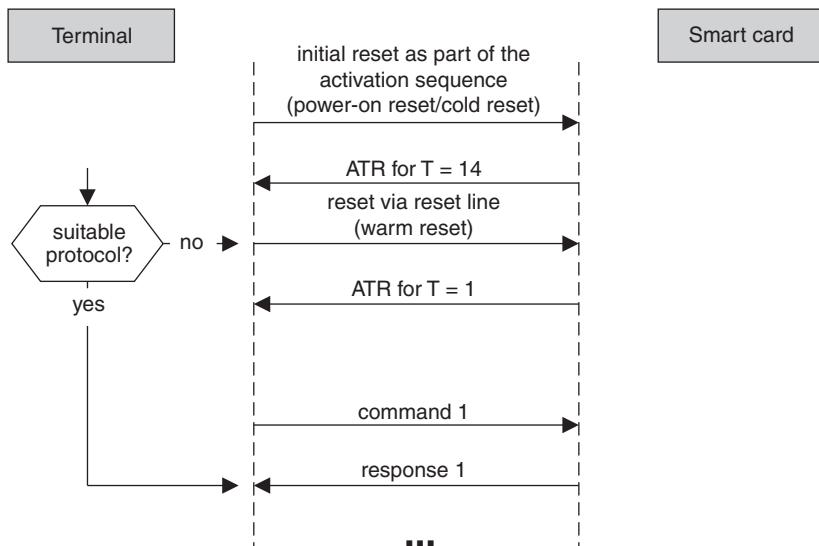
When a card is inserted in a terminal, its contacts are first mechanically connected to those of the terminal. The five active contacts are then electrically enabled in the correct sequence.<sup>2</sup> Following this, the card automatically executes a power-on reset and then sends an Answer to Reset (ATR) to the terminal. The terminal evaluates the ATR, which contains various parameters related to the card and data transmission, and then sends the first command. The card processes the command and generates a response, which it sends back to the terminal. This back-and-forth interplay of commands and responses (illustrated in Figure 8.2 on the next page) continues until the smart card is deactivated.

<sup>1</sup> See also Section 19.4.4, ‘TMSI’, on page 811

<sup>2</sup> See also Section 4.6, ‘Activation and Deactivation Sequences’, on page 70



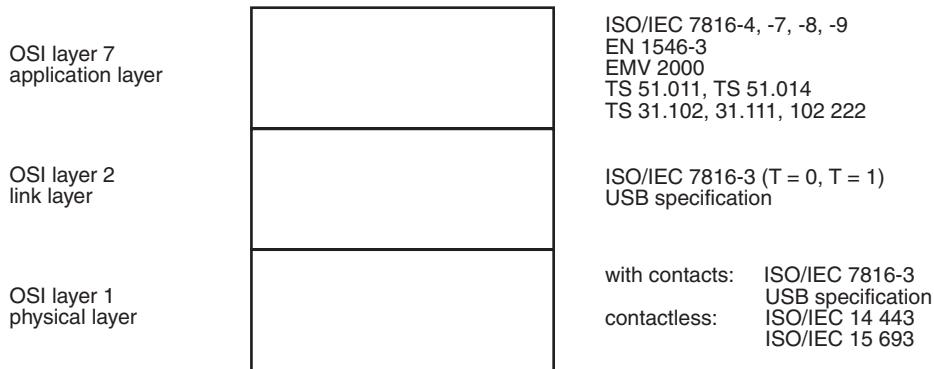
**Figure 8.1** General states of a smart card during activation and communication with the terminal



**Figure 8.2** Communication between the terminal and the smart card: Answer to Reset (ATR), Protocol Parameter Selection (PPS), and the first command–response pair

Between the ATR and the first command sent to the card, the terminal can send a Protocol Parameter Selection (PPS) command. The terminal can use this command, which like the ATR is independent of the transmission protocol, to configure various parameters of the card's transmission protocol.

The entire process of data transmission to and from the smart card can be represented using the OSI layer model. This model distinguishes between the electrical events on the I/O line, the logical processes in the transmission protocol, and the activities of the application that uses these processes. The behavior and the interactions in and between these layers are specified in several international standards. These relationships are illustrated in Figure 8.3 on the facing page.



**Figure 8.3** OSI model of communication between the terminal and the smart card and the most significant associated standards

In this chapter, the functions of the asynchronous transmission protocols are described with reference to the relevant standards. All parameters and settings allowed within the context of the protocols are described. In practice, smart cards often do not support all options of the transmission protocols, due to the limitations of available memory.

From a functional perspective, the options can be regarded as simply a set of available features from which an optimum set can be selected for a particular application or smart card. Here it is important to ensure that the selected parameters are not exotic, so that the card can communicate with as many different terminals as possible.

On the terminal side, the situation with regard to data transmission protocols is somewhat different. There the full functionality of the relevant standard is usually implemented, since sufficient memory is available.

The processes described above apply to the  $T = 0$  and  $T = 1$  transmission protocols, which are used almost exclusively with smart cards. More recent protocols such as USB<sup>3</sup>, MMC<sup>4</sup>, and SWP<sup>5</sup> utilize other processes and protocol elements, which are described in detail in the corresponding sections of this book.

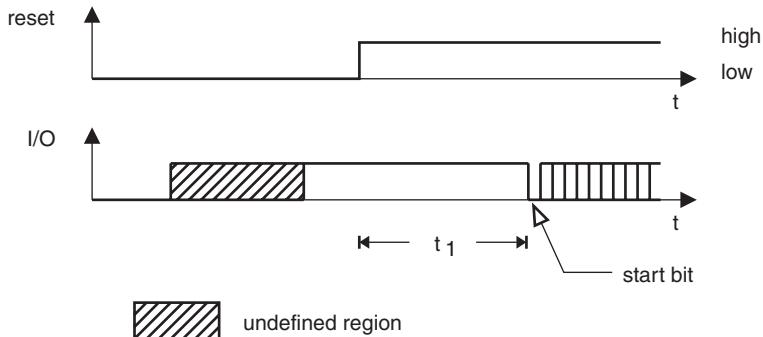
## 8.1 ANSWER TO RESET (ATR)

After the supply voltage, clock signal and reset signal have been applied, the smart card sends an Answer to Reset (ATR) on the I/O line. This data string, with a maximum length of 33 bytes, contains various parameters related to the transmission protocol and the card, and it is always sent with a divisor value (clock rate conversion factor) of 372 in compliance with the ISO/IEC 7816-3 standard. This divisor value must be used, even if a different value (such as 64) will be used for the transmission protocol after the ATR. This ensures that an ATR from any card can always be received, regardless of the parameters of the transmission protocol that is ultimately used.

<sup>3</sup> See also Section 9.4, ‘USB Transmission Protocol’, on page 272

<sup>4</sup> See also Section 9.5, ‘MMC Transmission Protocol’, on page 277

<sup>5</sup> See also Section 9.6, ‘Single-Wire Protocol’, on page 278



**Figure 8.4** Timing of the reset signal and the start of the ATR in accordance with ISO/IEC 7816-3, with the time constraint  $400 \text{ clock cycles} \leq t_1 \leq 40\,000 \text{ clock cycles}$

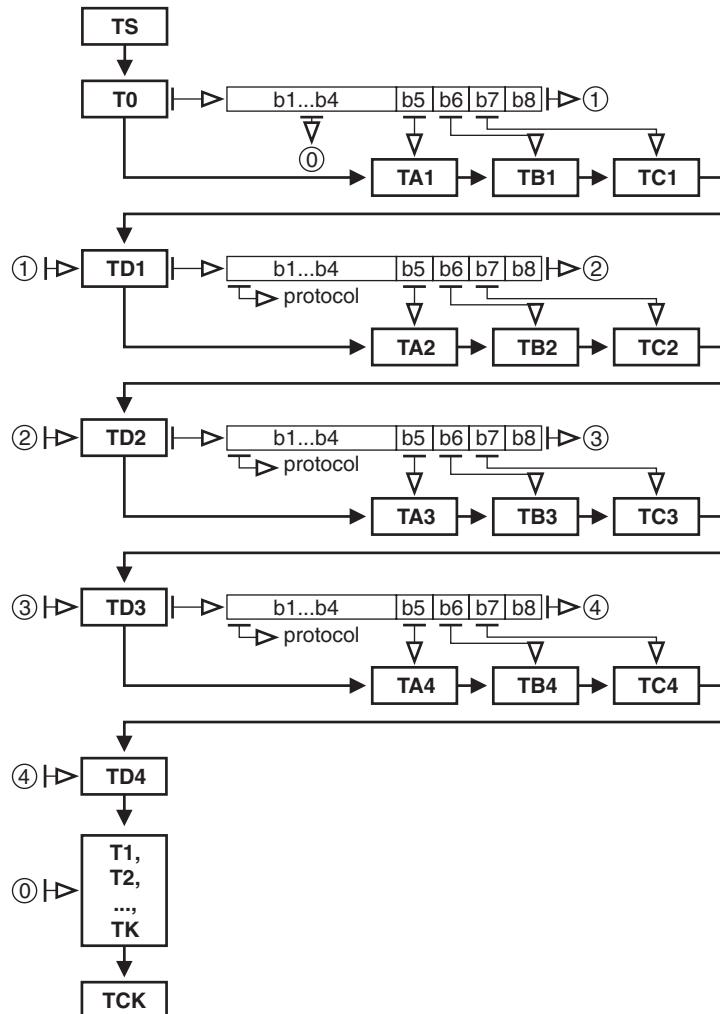
It is rare for the ATR to have the maximum allowable length. It most often consists of only a few bytes. Particularly in applications where the card should be usable very soon after the activation sequence, the ATR should be short. A typical example is paying a road toll with an electronic purse in the form of a smart card. No matter how quickly the vehicle passes through the toll gate, reliable debiting of the purse must be possible within the short time available after card activation.

The start of ATR transmission must occur between 400 and 40 000 clock cycles after the terminal asserts the reset signal, as illustrated in Figure 8.4. With a clock frequency of 3.5712 MHz, this corresponds to an interval of 112 to 11.20 ms, while at 4.9152 MHz the interval is 81.38  $\mu\text{s}$  to 8.14 ms. If the terminal does not receive the start of the ATR within this interval, it usually repeats the activation sequence up to three times while trying to detect an ATR. If all of these attempts fail, the terminal assumes that the card is defective and responds accordingly.

ISO/IEC 7816-3 specifies a maximum interval of 9600 etu between the leading edges of two successive bytes during ATR transmission. This interval is called the initial waiting time, and it is exactly 1 s with a clock frequency of 3.5712 MHz. This means that the standard allows a full second between the transmission of successive bytes of the ATR to the terminal. In some smart card operating systems, this time is utilized for internal computations and EEPROM write operations. The internal write buffer for atomic operations is often flushed at this time.<sup>6</sup>

The data string and data elements of the ATR are defined and described in detail in the ISO/IEC 7816-3 standard. The basic structure of the ATR is shown in Figure 8.5 on the next page, and the data elements are listed in Table 8.1 on the facing page. The first two bytes, designated TS and T0, define several basic transmission parameters and indicate the presence of subsequent bytes. The interface characters specify special transmission parameters for the protocol, which are important for subsequent data transmission. The historical characters describe the basic functional scope of the smart card. Depending on the transmission protocol used, the check character TCK, which is a checksum of the previous bytes, may be sent as the last byte of the ATR.

<sup>6</sup> See also Section 13.10, ‘Atomic Operations’, on page 480



**Figure 8.5** Basic structure and data elements of the ATR

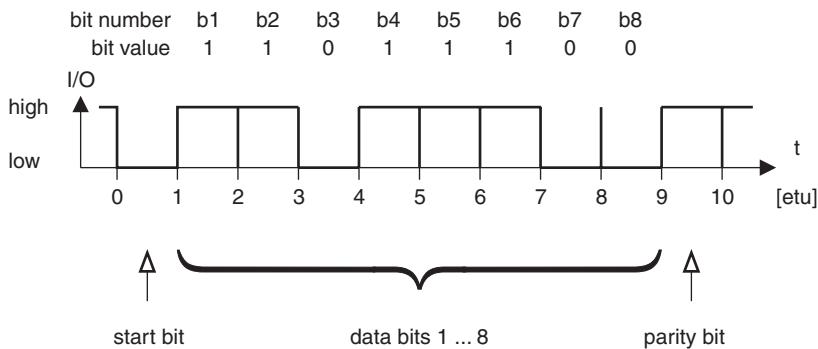
**Table 8.1** The data elements of the ATR and their designations according to ISO/IEC 7816-3

Data element	Designation
TS	Initial character
T0	Format character
TA <sub>1</sub> , TB <sub>1</sub> , TC <sub>1</sub> , TD <sub>1</sub> , ...	Interface characters
T1, T2, ..., TK	Historical characters
TCK	Check character

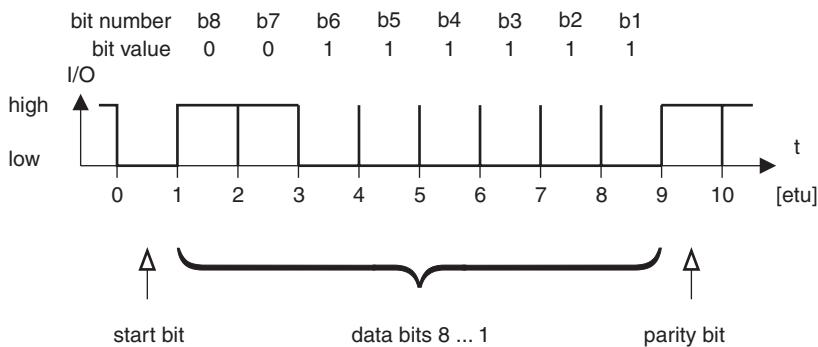
### 8.1.1 The initial character

This byte, designated TS, specifies the convention used for all of the data in the ATR and the subsequent communication protocol. The TS byte also contains a characteristic bit pattern that can be used by the terminal to determine the divisor value. For this purpose, the terminal can measure the time between the first two falling edges of TS and divide it by three. The result is the duration of one elementary time unit (etu). As the divisor has a fixed value of 372 for the ATR, the terminal usually does not evaluate this synchronization pattern. The first byte is a mandatory component of the ATR and must always be sent. Only two codes are allowed for this byte: '3B' with the direct convention or '3F' with the inverse convention.

The direct convention is normally used in Germany, while the inverse convention is used primarily in France. The convention does not affect the quality of data transmission. Of course, various operating system producers have their own preferences based on historical reasons, but all terminals and many smart cards support both conventions. The bit sequence of the initial character TS with the direct convention and indirect convention is shown in Figures 8.6 and 8.7, respectively, and the coding of TS is described in Table 8.2.



**Figure 8.6** Timing of the initial character TS with the direct convention, which is indicated by the value '3B' = 0011 1011



**Figure 8.7** Timing of the initial character TS with the inverse convention, which is indicated by the value '3F' = 0011 1111

**Table 8.2** Coding of the initial character TS

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
				'3B'				Direct convention
				'3F'				Inverse convention

**Table 8.3** Coding of the format character T0

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
...	...	...	...	X				Number of historical characters [0 ... 15]
...	...	...	1		...			TA <sub>1</sub> transmitted
...	...	1	...		...			TB <sub>1</sub> transmitted
...	1	...	...		...			TC <sub>1</sub> transmitted
1	...	...	...		...			TD <sub>1</sub> transmitted

### 8.1.2 The format character

The second byte, T0, contains a bit field that indicates which interface characters follow it in the ATR. It also indicates the number of historical characters after the interface characters. Like the TS byte, this byte must be present in every ATR. The coding of the format character is described in Table 8.3.

### 8.1.3 The interface characters

The interface characters specify all the transmission parameters of the protocol currently being used. They consist of the TA<sub>i</sub>, TB<sub>i</sub>, TC<sub>i</sub>, and TD<sub>i</sub> bytes. However, these bytes are optional in the ATR, and they may be omitted as appropriate. Default values are defined for all of the transmission protocol parameters, so the interface characters are often not necessary in the ATR for conventional communication.

The interface characters can be classified into global interface characters and specific interface characters. The global interface characters specify basic parameters of the transmission protocol, such as the divisor, that apply to all subsequent protocols. The specific interface characters specify parameters for a specific transmission protocol. The work waiting time for T = 0 is a typical example of such a parameter.

The global interface characters fundamentally apply to all protocols, but for historical reasons (since originally only the T = 0 protocol was defined in the ISO standards), several of these characters are only relevant to the T = 0 protocol. If T = 0 is not implemented, they can be omitted, in which case the default values apply.

The only purpose of a TD<sub>i</sub> byte is to provide a link to the following interface characters. For this purpose, the upper nibble of TD<sub>i</sub> contains a bit pattern that indicates which of the TA<sub>i+1</sub>, TB<sub>i+1</sub>, TC<sub>i+1</sub> and TD<sub>i+1</sub> interface characters are present, using the same coding as the format character T0. The lower nibble of TD<sub>i</sub> identifies the associated available transmission

**Table 8.4** Coding of the  $TD_i$  bytes

b8	b7	b6	b5	b4	b3	b2	b1	Meaning	
...	...	...	...	$X$					Transmission protocol number [0 ... 15]
...	...	...	1	...					$TA_{i+1}$ transmitted
...	...	1	...	...					$TB_{i+1}$ transmitted
...	1	...	...	...					$TC_{i+1}$ transmitted
1	...	...	...	...					$TD_{i+1}$ transmitted

**Table 8.5** Coding of  $TA_1$ 

b8	b7	b6	b5	b4	b3	b2	b1	Meaning	
$X$				...					FI
...				$X$					DI

protocol. The  $TD_i$  byte is omitted if the  $TA_{i+1}$ ,  $TB_{i+1}$ ,  $TC_{i+1}$  and  $TD_{i+1}$  interface characters are not transmitted. The coding of the  $TD_i$  byte is described in Table 8.4.

The other interface characters ( $TA_i$ ,  $TB_i$ , and  $TC_i$ ), which are not used for linking, define the available transmission protocol(s). Their meanings according to ISO/IEC 7816-3 are described below.

### 8.1.3.1 Global interface character $TA_1$

The coding of the  $TA_1$  byte is described in Table 8.5. Parameter FI in the upper nibble codes the divisor (clock rate conversion factor) F. Parameter DI in the lower nibble codes the bit rate adjustment factor D.

The coding parameters for the divisor  $F$  and the bit rate adjustment factor  $D$  allow typical transmission rates to be specified in accordance with the standard by using the following formulas. The bit interval of the ATR and PPS, which is called the initial etu, is defined by the formula

$$\text{initial etu} = \frac{372}{f} \text{ s}$$

The bit interval of the transmission protocol used after the ATR and PPS can be defined independently of the ATR. This interval is called the work etu and is defined by the formula

$$\text{work etu} = \frac{F}{D} \times \frac{1}{f} \text{ s}$$

The bit rate adjustment factor  $D$  and the clock rate conversion factor  $F$  allow the transmission rate to be modified and adapted to specific conditions. The frequency of the applied clock signal (in hertz) is indicated by the symbol  $f$  in the above formulas. The maximum allowable clock frequency is designated  $f_{\max}$ . The standard value of  $f_{\max}$  is 5 MHz. Table 8.6 describes the coding of the  $F_i$  and the associated values of  $f_{\max}$ , while Table 8.7 describes the coding of  $D$ .

**Table 8.6** Coding of  $F_i$  and  $f_{\max}$ 

Bits 8 . . . 5	0000	0001	0010	0011	0100	
$F_i$	372	372	558	744	1 116	
$f_{\max}$	4 MHz	5 MHz	6 MHz	8 MHz	12 MHz	
Bits 8 . . . 5	0101	0110	0111	1000	1001	
$F_i$	1 488	1 860	RFU	RFU	512	
$f_{\max}$	16 MHz	20 MHz	—	—	5 MHz	
Bits 8 . . . 5	1010	1011	1100	1101	1110	1111
$F_i$	768	1 024	1 536	2 048	RFU	RFU
$f_{\max}$	7.5 MHz	10 MHz	15 MHz	20 MHz	—	—

**Table 8.7** Coding of Di

Bits 4 . . . 1	0000	0001	0010	0011	0100	0101	0110	0111
D	RFU	1	2	4	8	16	32	64
Bits 4 . . . 1	1000	1001	1010	1011	1100	1101	1110	1111
D	12	20	RFU	RFU	RFU	RFU	RFU	RFU

**Table 8.8** Coding of X

Bits 8–7	00	01	10	11
X	Not supported	Low state	High state	No preferred state

**Table 8.9** Coding of Y. All unlisted values are reserved for future use (RFU)

Bits 6–1	00 0001	00 0010	00 0100
Y	Voltage class A4.5–5.5 V	Voltage class B2.7–3.3 V	Voltage class C1.62–1.98 V
Bits 6–1	00 0011	00 0110	00 0111
Y	Voltage classes A and B	Voltage classes B and C	Voltage classes A, B and C

### 8.1.3.2 Global interface character $TA_i$

The value of  $TA_i$  is interpreted as  $X||Y$  if  $i > 2$  and  $T = 15$  ('F') in  $TD_{(i-1)}$ .  $TA_i$  codes the clock stop indicator  $X$ , which indicates the logical state the clock line must assume when the clock is stopped (see Table 8.8), and the class indicator  $Y$  (see Table 8.9), which specifies the supply voltage class.

### 8.1.3.3 Global interface character $TC_1$

$TC_1$  codes an extra guard time, designated  $N$ , as an unsigned hexadecimal integer as described in Table 8.10. This extra guard time is defined as an extension to the duration of the stop bit.

**Table 8.10** Coding of TC<sub>1</sub>

b8	b7	b6	b5	b4	b3	b2	b1	Description			
				X	Extra guard time $N$ , with a range of 0 . . . 254 etu						
1	1	1	1	1	1	1	1	X = 255 and T = 0: guard time = 2 etu			
								X = 255 and T = 1: guard time = 1 etu			

**Table 8.11** Coding of TA<sub>i</sub> for  $i > 2$ 

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
X								IFSC

The value  $N$  indicates how many additional etus are to be added to the guard time. TC<sub>1</sub> is coded linearly with the exception of  $N = \text{'FF'}$ , which has a special meaning: with the T = 1 protocol, the normal guard time of 2 etu is reduced to 1 etu if  $N = \text{'FF'}$ . With the T = 0 protocol, the normal guard time of 2 etu is retained to allow an error to be indicated as usual by a low level within the guard time interval. In practice, reducing the guard time to 1 etu with the T = 1 protocol increases the effective data transmission rate by nearly 10 %, since only 11 bits have to be sent for each character instead of 12.

#### 8.1.3.4 Specific interface character TC<sub>2</sub>

This specific interface character is defined for the T = 0 transmission protocol. This data element contains a positive integer WI that codes the waiting time WT. The waiting time is the maximum interval between the leading edges of two consecutive bytes. If TC<sub>2</sub> is not present in the ATR, the waiting time is determined by the default value of the waiting time integer (WI = 10).

$$WT = WI \times 960 \times \frac{F_i}{f} \text{ work etu}$$

#### 8.1.3.5 Specific interface character TA<sub>i</sub> ( $i > 2$ )

The TA<sub>i</sub> byte contains the maximum length of the information field that can be received by the card (IFSC), coded as described in Table 8.11. This value must be in the range of 1 to 254, including both limit values. The default value of IFSC is 32 bytes.

#### 8.1.3.6 Specific interface character TB<sub>i</sub> ( $i > 2$ )

The TB<sub>i</sub> byte is coded as described in Table 8.12. The lower nibble (bits b4 to b1) contains the code CWI for the character waiting time CWT,<sup>7</sup> which is defined by the formula

$$CWT = (11 + 2^{CWI}) \text{ work etu}$$

<sup>7</sup> See also Section 9.3.2, ‘The T = 1 transmission protocol’, on page 260

**Table 8.12** Coding of TB<sub>i</sub> for i > 2

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
...	X				X	...		CWI BWI

**Table 8.13** Coding of TC<sub>i</sub> for i > 2

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
...	...	...	...	...	...	...	0	LRC is used
...	...	...	...	...	...	...	1	CRC is used
0	0	0	0	0	0	0	...	Reserved for future use

The upper nibble contains the value BWI, which defines the block waiting time BWT<sup>8</sup> using the formula

$$\text{BWT} = 11 \text{ work etu} + 2^{\text{BWI}} \times 960 \times \frac{Fd}{f}$$

#### 8.1.3.7 Specific interface character TC<sub>i</sub> (i > 2)

The TC<sub>i</sub> byte is coded as described in Table 8.13. Bit b1 codes the method used to calculate the error detection code. It distinguishes two methods: LRC and CRC.

#### 8.1.3.8 Global interface character TA<sub>2</sub>

The TA<sub>2</sub> byte indicates the allowed modes for the PPS and is coded as described in Table 8.12. This is explained in more detail in the description of the PPS in Section 8.2 on page 217.

### 8.1.4 The historical characters

For a long time, the historical characters were not defined by any standard. As a result, there are many specifications for these characters that are only valid for a particular operating system.

Many companies use the available bytes to identify the operating system and the associated version number of the ROM mask or flash OS. The coding is usually ASCII, which makes it easy to interpret. Historical characters are not required to be present in the ATR, so they may be omitted entirely. This can be advantageous in some situations because it shortens the ATR and thus reduces its transmission time.

<sup>8</sup> See also Section 9.3.2, ‘The T = 1 transmission protocol’, on page 260

**Table 8.14** Coding of TA<sub>2</sub>

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	...	...	...		...			Switching between negotiable mode and specific mode is possible.
1	...	...	...		...			Switching between negotiable mode and specific mode is not possible.
...	0	0	...		...			Reserved for future use.
...	...	...	0		...			Transmission parameters are explicitly defined by the interface characters.
...	...	...	1		...			Transmission parameters are implicitly defined by the interface characters.
...	...	...	...		X			Protocol T = X is to be used.

**Table 8.15** Coding of the category indicator in T1

Value	Meaning
'00'	Status information is located at the end of the historical characters
'10'	Reference to a DIR file (directory file)
'80'	Any status information present is held in a compact TLV-coded data object
'81' ... '8F'	RFU
all other values	Application-specific (proprietary)

The ISO/IEC 7816-4 standard provides for an ATR file in addition to the historical characters. This file, with the reserved FID '2F01', is located below the MF and contains additional data regarding the ATR. It is intended to be an extension to the historical characters, which are limited to 15 bytes. The data in this file, whose structure is not specified by the standard, is ASN.1 coded.

The data elements in the ATR file or the historical characters may contain complex information regarding the smart card and the operating system used in the card. For example, they can indicate which file selection and implicit selection functions are supported by the smart card and provide information about the logical channel mechanism. They can also hold additional information on the card issuer, the card and chip serial numbers, the ROM mask version, the chip, and the operating system. The coding of the relevant data objects is defined in the ISO/IEC 7816-4 standard.

According to ISO/IEC 7816-4, the historical characters can contain the following three data fields: an obligatory category indicator, one or more optional data blocks in compact TLV format, and an optional status indicator.

The category indicator is transferred in T1. It contains information about the structure of the data in the ATR. The compact TLV format has a tag in the first nibble and the length of the following data in the second nibble. The data following the category indicator holds information about the functions and services provided by the smart card operating system (the major functions and services are described in Tables 8.15 to 8.19 on page 214). The status indicator indicates the life cycle phase of the smart card.

**Table 8.16** Coding of application-independent card services with compact TLV-coded data object ‘31’ (tag || length)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	...	...	...	...	...	...	...	Application selection with full DF name
...	1	...	...	...	...	...	...	Application selection with partial DF name
...	...	1	...	...	...	...	...	Data objects available in the DIR file
...	...	...	1	...	...	...	...	Data objects available in the ATR file
...	...	...	...	1	...	...	...	Data objects can be read from the DIR or ATR file with READ BINARY
...	...	...	...	0	...	...	...	Data objects can be read from the DIR or ATR file with READ RECORD
...	...	...	...	...	0	0	0	Not used

**Table 8.17** Coding of card functions (first software function table) in compact TLV-coded data objects. The combined tag/length byte can be ‘71’, ‘72’ or ‘73’, depending on the number of data bytes

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	...	...	...	...	...	...	...	DF selection with full DF name
...	1	...	...	...	...	...	...	DF selection with partial DF name
...	...	1	...	...	...	...	...	DF selection by specifying a path
...	...	...	1	...	...	...	...	DF selection with FID
...	...	...	...	1	...	...	...	Implicit DF selection (with the desired application)
...	...	...	...	...	1	...	...	Short FIDs are supported
...	...	...	...	...	...	1	...	Record numbers are supported
...	...	...	...	...	...	...	1	Record identifiers are supported

**Table 8.18** Coding of card functions (second software function table) in compact TLV-coded data objects. The combined tag/length byte can be ‘71’, ‘72’ or ‘73’, depending on the number of data bytes

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
...	0	0	...	...	...	...	...	The WRITE command acts as a one-time write function
...	0	1	...	...	...	...	...	WRITE command behavior is application-specific
...	1	0	...	...	...	...	...	The WRITE command acts as a logical OR function
...	1	1	...	...	...	...	...	The WRITE command acts as a logical AND function
...	...	...	...	1	...	...	...	Implicit DF selection (with the desired application).
...	...	...	...	...	x	x	x	Data unit size in nibbles modulo 2 ( $001 = 2^2 = 2 \text{ nibbles} = 1 \text{ byte}$ )
0	...	...	0	0	...	...	...	Not used

**Table 8.19** Coding of the card functions (third software function table) in compact TLV-coded data objects. The combined tag/length byte can have a value of '71', '72' or '73', depending on the number of data bytes

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	...	...	...	...	...	...	...	Command chaining is allowed.
...	1	...	...	...	...	...	...	Extended L <sub>c</sub> and /L <sub>e</sub> fields.
...	...	...	1	0	...	...	...	Logical channels are assigned by the card when the MANAGE CHANNEL command is used.
...	...	...	0	1	...	...	...	Logical channels are assigned by the terminal when the MANAGE CHANNEL command is used.
...	...	...	0	0	...	...	...	Logical channels are not supported.
...	...	x	...	...	y	z	t	Maximum number of logical channels (8x + 4y + 2z + t + 1).

### 8.1.5 The check character

The final byte of the ATR is the check character TCK, which holds the XOR checksum of bytes from T0 to the last byte before TCK. This checksum can be used in addition to parity checking to verify that the ATR has been transmitted correctly. However, despite the apparent simplicity of the structure and computation of this checksum, there are significant differences between the different transmission protocols.

If only the T = 0 transmission protocol is indicated in the ATR, TCK is not allowed to be present at the end of the ATR. In this case, it is not sent because bytewise error detection using parity checking and retransmission of incorrect bytes is mandatory with the T = 0 protocol. By contrast, the TCK byte must be present in the ATR if the T = 1 protocol is used. The checksum is calculated starting with byte T0 and ending with the final interface character, or with the final historical character if historical characters are present.

### 8.1.6 Practical examples of ATRs

Tables 8.20 to 8.25 show practical examples of several types of smart card ATRs. They are very useful as an aid to interpreting ATRs or defining your own ATRs.

**Table 8.20** Example smart card ATR with the T = 1 transmission protocol in direct convention

Designation	Value	Meaning	Remarks
TS	'3B'	direct convention	
T0	'B5'	Y1 = 1011 = 'B' K = '5'	TA <sub>1</sub> , TB <sub>1</sub> and TD <sub>1</sub> follow 5 historical characters
TA <sub>1</sub>	'11'	FI = 0001 = '1' DI = 0001 = '1'	F = 372
TB <sub>1</sub>	'00'	II = 0 PI1 = 0000 = '0'	D = 1 I = 0 Vpp contact not used
TD <sub>1</sub>	'81'	Y2 = 1000 = '8'	TD <sub>2</sub> follows
TD <sub>2</sub>	'31'	T = 1 Y2 = 0011 = '3'	TA <sub>3</sub> and TB <sub>3</sub> follow
TA <sub>3</sub>	'46'	I/O buffer size = 70 bytes	ICC I/O buffer size (layer 7)
TB <sub>3</sub>	'15'	BWI = '1' CWI = '5'	block waiting time character waiting time
T1	'56'	'V'	'V 1.0'
T2	'20'	'.'	
T3	'31'	'1'	
T4	'2E'	'.'	
T5	'30'	'0'	
TCK	'1E'	check character	XOR checksum of T0 to T5

**Table 8.21** Example ATR of a STARCOS smart card with the T = 1 transmission protocol in direct convention, with the operating system not yet completed

Designation	Value	Meaning	Remarks
TS	'3B'	direct convention	
T0	'9C'	Y1 = '9' = 1001 K = 'C' = 12	TA <sub>1</sub> and TD <sub>1</sub> follow 12 historical characters
TA <sub>1</sub>	'11'	FI = 0001 = '1' DI = 0001 = '1'	F = 372 D = 1
TD <sub>1</sub>	'81'	'8' = 1000 T = 1	TD <sub>2</sub> follows T = 1 is used
TD <sub>2</sub>	'21'	'2' = 0010 T = 1	TB <sub>3</sub> follows T = 1 is used
TB <sub>3</sub>	'34'	BWI = 3 CWI = 4	block waiting time character waiting time
T1 ... T12	'53'    '43'    '20'    '53'    '56'    '20'    '31'    '2E'    '31'    '20'    '4E'    '43'		"SC SV 1.1 NC"
TCK	'0F'	check character	XOR checksum of T0 to T12

**Table 8.22** Example ATR of a STARCOS smart card with the T = 1 transmission protocol in direct convention, with the operating system completed

Designation	Value	Meaning	Remarks
TS	'3B'	direct convention	
T0	'BF'	Y1 = 'F' = 1001 K = 'B' = 11	TA <sub>1</sub> and TD <sub>1</sub> follow 11 historical characters
TA <sub>1</sub>	'11'	FI = 0001 = '1' DI = 0001 = '1'	F = 372 D = 1
TD <sub>1</sub>	'81'	'8' = 1000 T = 1	TD <sub>2</sub> follows T = 1 is used
TD <sub>2</sub>	'21'	'2' = 0010 T = 1	TB <sub>3</sub> follows T = 1 is used
TB <sub>3</sub>	'34'	CWI = 3 BWI = 4	character waiting time block waiting time
T1 … T15	'53'    '54'    '41'    '52'    '43'    '4F'    '53'    '32'    '31'    '20'    '43'		"STARCOS21 C"
TCK	'43'	check character	XOR checksum of T0 to T15

**Table 8.23** Example ATR of a Visa Cash smart card with the T = 1 transmission protocol in direct convention

Designation	Value	Meaning	Remarks
TS	'3B'	direct convention	
T0	'E3'	Y1 = 'E' = 1110 K = 3	TB <sub>1</sub> , TC <sub>1</sub> und TD <sub>1</sub> follow 3 historical characters
TB <sub>1</sub>	'00'	PI1 = 00000 II = 00	Vpp not used; the EEPROM programming voltage is generated internally
TC <sub>1</sub>	'00'	N = 0	no extra guard time
TD <sub>1</sub>	'81'	'8' = 1000 T = 1	TD <sub>2</sub> follows T = 1 is used
TD <sub>2</sub>	'31'	'3' = 0011 T = 1	TA <sub>3</sub> and TB <sub>3</sub> follow T = 1 is used
TA <sub>3</sub>	'6F'	IFSC = '6F' = 111	The information field size of the smart card is 111 bytes
TB <sub>3</sub>	'45'	BWI = 4 CWI = 5	block waiting time character waiting time
T1	'80'	category indicator	A data object in compact TLV format follows (compliant with ISO/IEC 7816-4)
T2	'31'	card service data	Tag ('3') and length ('1') of the data object holding the card service data
T3	'C0'	'C0' = 1100 0000	Application selection with a full or partial DF name
TCK	'08'	check character	XOR checksum of T0 to T3

**Table 8.24** Example ATR of a GSM smart card with the T = 0 transmission protocol in direct convention

Designation	Value	Meaning	Remarks
TS	'3B'	direct convention	
T0	'89'	Y1 = '8' = 1000 K = 9	TD <sub>1</sub> follows 9 historical characters
TD <sub>1</sub>	'40'	'4' = 0100 T = 0	TC <sub>2</sub> follows T = 0 is used
TC <sub>2</sub>	'14'	WI = '14'	work waiting time (WWT) = 20
T1 ... T9	'47'    '47'    '32'    '34'    '4D'    '35'    '32'    '38'    '30'		"GG24M5280"

**Table 8.25** Example ATR of a GSM smart card with the T = 0 transmission protocol in inverse convention

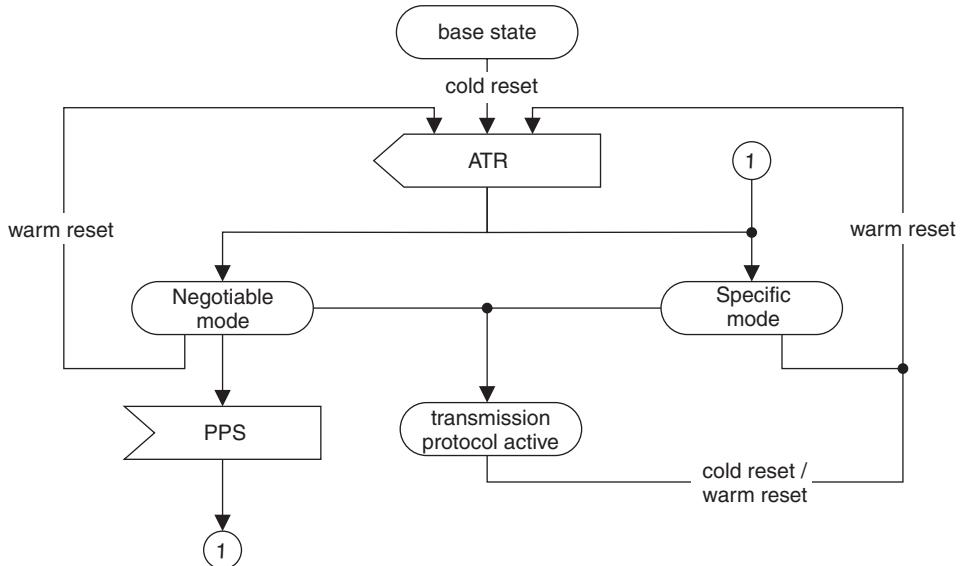
Designation	Value	Meaning	Remarks
TS	'3F'	inverse convention	
T0	'2F'	Y1 = '2' = 0010 K = 'F' = 15	TB <sub>1</sub> follows 15 historical characters
TB <sub>1</sub>	'00'	PI1 = 00000 II = 00	V <sub>pp</sub> not used; the EEPROM programming voltage is generated internally
T1 ... T15	'80'    '69'    'AE'    '02'    '02'    '01'    '36'    '00'    '00'    '0A'    '0E'    '83'    '3E'    '9F'    '16'		Specific data of the smart card producer

## 8.2 PROTOCOL PARAMETER SELECTION (PPS)

The smart card specifies various data transmission parameters in the interface characters of the ATR, such as the transmission protocol and the character waiting time. If the terminal wants to modify one or more of these parameters, it must perform a Protocol Parameter Selection (PPS) process in accordance with ISO/IEC 7816-3 before the transmission protocol is actually used. This allows the terminal to modify certain protocol parameters if this is allowed by the card. Prior to 1997, Protocol Parameter Selection was called Protocol Type Selection.

PPS can be performed in two different modes, as depicted in Figure 8.8. In negotiable mode, the standard values of the divisor  $F$  and the bit rate adjustment factor  $D$  remain unchanged until a PPS has been successfully executed. If the card uses specific mode, the values of  $F$  and  $D$  specified in the ATR must also be used for PPS transmission. The card indicates which of these two modes it supports in the TA<sub>2</sub> byte.

The PPS request must be sent immediately after the ATR has been received by the terminal. If the card allows the requested changes to the protocol parameters, it sends the received PPS bytes back to the terminal. This essentially amounts to echoing the received data. Otherwise the card sends nothing, and the terminal must perform a new reset sequence to cause the card



**Figure 8.8** State diagram of the two PPS modes specified by ISO/IEC 7816-3: negotiable mode and specific mode

**Table 8.26** PPS data elements and their designations according to ISO/IEC 7816-3

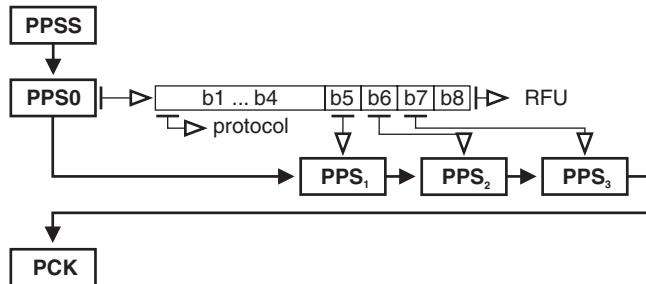
Data element	Designation
PPSS	Initial character
PPS0	Format character
PPS <sub>1</sub> , PPS <sub>2</sub> , PPS <sub>3</sub>	Parameter characters
PCK	Check character

to exit this state. The PPS process may be performed only once, immediately after the ATR. Repeated transmission of the PPS is prohibited by ISO/IEC 7816-3.

In practice, PPS is always used when the transmission time is a distinctly perceptible part of the overall application time. For example, PPS is usually used with SIMs in GSM and UMTS mobile telephones to accelerate the voluminous data transfers between the mobile equipment and the smart card.

The elements of the PPS are listed in Table 8.26, and the basic structure of the PPS is shown in Figure 8.9. The first PPS byte is the initial character (PPSS), which unambiguously informs the card that the terminal is initiating a PPS request immediately after the ATR. It has a fixed value of 'FF' and is a mandatory element of the PPS.

The data element following the PPSS is the format character (PPS0), whose coding is described in Table 8.27. It is also a mandatory element of the PPS. It may optionally be followed by up to three bytes, which are called the parameter characters and are designated PPS<sub>1</sub>, PPS<sub>2</sub>, and PPS<sub>3</sub>. They code various parameters of the transmission protocol to be used after the PPS. The coding of PPS<sub>1</sub> is described in Table 8.28.

**Figure 8.9** Basic structure and data elements of the PPS**Table 8.27** Coding of PPS0

b8	b7	b6	b5		b4	b3	b2	b1	Meaning
...	...	...	...			X			Transmission protocol to be used
...	...	...	1			...			PPS1 is present
...	...	1	...			...			PPS2 is present
...	1	...	...			...			PPS3 is present
0	...	...	...			...			Reserved for future use

**Table 8.28** Coding of PPS1

b8	b7	b6	b5		b4	b3	b2	b1	Meaning
		X				...			F (same coding as TA1)
		...				X			D (same coding as TA1)

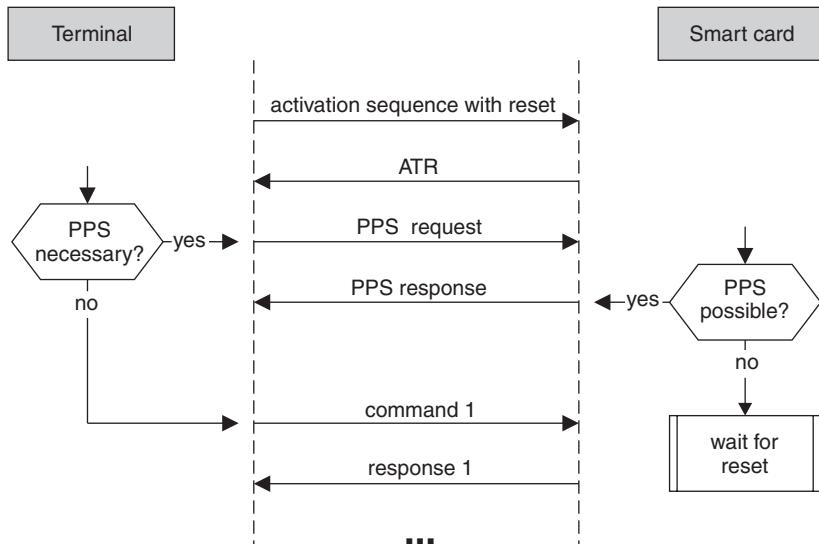
PPS<sub>2</sub> enables the terminal to switch to the SPU (standard or proprietary use) contact. This can be used to activate another protocol, such as USB. Data element PPS<sub>3</sub> is reserved for future use and thus not described here.

The final byte of the PPS is called the control character (PCK). It contains the XOR checksum of all previous bytes starting with PPSS. PCK (along with PPSS and PPS0) is a mandatory element of the PPS; the other data elements are optional.

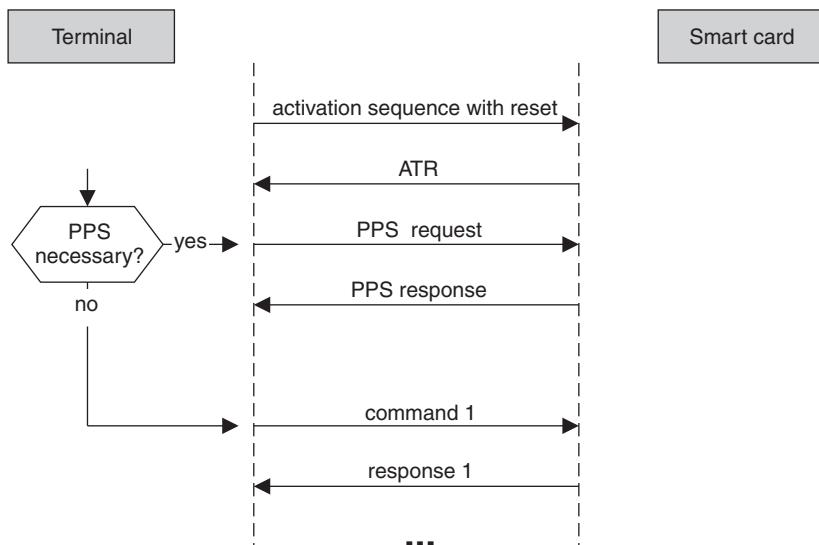
Figure 8.10 shows a typical PPS sequence. If the card can interpret the PPS and modify the transmission protocol accordingly, it acknowledges this by sending the received PPS back to the terminal. If the card cannot implement the changes requested in the PPS, it simply waits until the terminal executes a reset. The main disadvantage of this process is that a large amount of time can be lost before the actual transmission protocol is used.

The PPS process described above is not suitable for changing protocols with a terminal that has its own specific protocol but cannot execute a PPS. This is precisely the situation with card phones in Germany, for example. A special process has been devised to enable protocol switching in such cases, as depicted in Figure 8.11.

As all terminals perform multiple reset sequences if they do not detect an ATR, it was decided that the smart card should switch transmission protocols after each reset. This can be



**Figure 8.10** A typical PPS sequence with a SIM card



**Figure 8.11** A possible sequence for switching between two transmission protocols supported by a smart card without using a PPS. With the sequence outlined here, the terminal does not have to perform an explicit PPS, but can nevertheless switch between the two protocols by initiating a reset

illustrated by an example. After the first reset, the card sends an ATR for  $T = 14$  and is then ready to communicate using the  $T = 14$  protocol. After the second reset, it sends an ATR for  $T = 1$  and is ready to communicate using the  $T = 1$  protocol. After the third reset, it is again ready to use the  $T = 14$  protocol. This solution is not ideal from a technical perspective, since a device should always behave the same after each reset, but it is certainly a pragmatic solution for a heterogeneous terminal world.

It is possible to mitigate the drawbacks of this solution by having the smart card always respond with the same ATR after a power-on reset (cold reset). A cold reset always occurs immediately after the card has been inserted in a terminal and the activation sequence has been completed. By contrast, a reset initiated by a signal on the reset line (warm reset) causes the card to switch to a different transmission protocol. The card thus behaves the same after every ‘real’ reset, while any additional reset trigger causes it to switch to a different transmission protocol.

## 8.3 MESSAGE STRUCTURE: APDUS

APDUs are used to exchange all data that passes between the smart card and the terminal. ‘APDU’ stands for ‘application protocol data unit’, which designates an internationally standardized data unit of the application layer (layer 7 in the OSI model). In smart cards, this layer is located directly above the transmission protocol layer. The protocol-dependent data units of the transmission protocol layer are called transmission protocol data units (TPDUs).

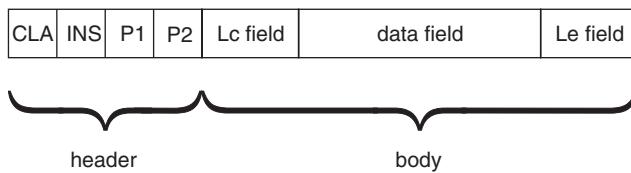
A distinction is made between command APDUs (C-APDUs), which represent commands sent to the card, and response APDUs (R-APDUs), which represent replies to these commands that are returned by the card. In simplified terms, an APDU is a sort of container that holds a complete command to the card or a complete response from the card. APDUs are transferred transparently by the transmission protocol, which means without modification or interpretation.

APDUs that comply with the ISO/IEC 7816-4 standard are designed to be independent of the transmission protocol. Consequently, the content and format of an APDU can remain unchanged when a different transmission protocol is used. This applies in particular to the two standardized protocols  $T = 0$  and  $T = 1$ . The requirement for transmission protocol independence affects the structure of the APDUs, since it must be possible to transmit them transparently using the byte-oriented  $T = 0$  protocol as well as the block-oriented  $T = 1$  protocol.

### 8.3.1 Command APDU structure

As depicted in Figure 8.12, a command APDU is composed of a header and a body. The body may have a variable length, or it may be entirely absent if the associated data field is empty.

The header consists of four elements: the class byte (CLA) described in Table 8.29, the instruction byte (INS), and two parameter bytes (P1 and P2). The class byte is also used to identify applications and their specific command sets, as described in Table 8.30. For example, the class byte for GSM is ‘A0’, while ‘8X’ is most often used for private-use (proprietary) commands. ISO-compliant commands are identified by the class byte ‘0X’. The standards also specify class bytes for secure messaging and logical channels, which is still compatible with the designated use of the class byte as an application identifier.



**Figure 8.12** Command APDU structure, consisting of a header and a body

**Table 8.29** Principal class byte (CLA) codes defined by ISO/IEC 7816-4

b8 . . . b5	b4	b3	b2	b1	Meaning
...	...	...	X	X	Logical channel number
...	0	0	...	...	Secure messaging not used
...	0	1	...	...	Secure messaging using a private method (not ISO compliant)
...	1	0	...	...	ISO-compliant secure messaging; header not authentic
...	1	1	...	...	ISO-compliant secure messaging; header authentic
'0'	...	...	...	...	Structure and coding as per ISO/IEC 7816-4/-7/-8
'8', '9'	...	...	...	...	Structure as per ISO/IEC 7816-4; user-specific coding and meaning of commands and responses (private use)
'A'	...	...	...	...	Structure and coding as per ISO/IEC 7816-4, specified in additional documents (e.g. TS 51.011)
'F'	1	1	1	1	Reserved for PPS

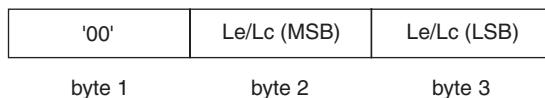
The next byte in the command APDU is the instruction byte, which codes the actual command. Almost the entire address space of this byte can be exploited, with the restriction that only even codes can be used. This is because the T = 0 protocol originally allowed the external programming voltage to be activated by returning the instruction byte incremented by 1 in the procedure byte. Since around 2004, this mechanism is no longer supported by the ISO/IEC standards. Instead, bit 1 is used to indicate BER-TLV coded commands with command chaining, which is also called layer-7 chaining. This can be used to transfer data that is too long for a single command.

The two parameter bytes are primarily used to provide more information for the command selected by the instruction byte. They thus serve mainly as switches that select various command options. For example, they can be used to select the various options of SELECT FILE or to specify the offset with READ BINARY.

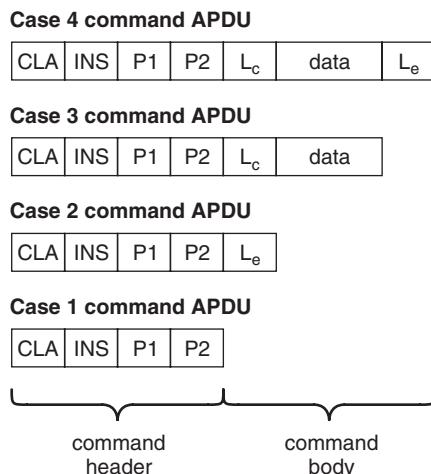
The section after the header is the body, which can be omitted except for a length parameter. The body has two purposes. First, it specifies the length of the data section sent to the card (in the L<sub>c</sub> (length command) field) and the length of the data section to be returned by card (in the L<sub>e</sub> (length expected) field). Second, it contains the command data that is sent to the card. If the value of L<sub>e</sub> is '00', the terminal expects the card to return the maximum amount of data available for the command. This is the only exception to the numerical description of the length specifications.

**Table 8.30** Assignment of class bytes to specific applications

Class	Application
'0X'	Standardized commands compliant with ISO/IEC 7816-4/-7/-8
'80'	Electronic purses compliant with EN 1546-3
'8X'	Application-specific and company-specific commands (private use)
'8X'	Credit cards with chips compliant with EMV
'A0'	GSM mobile telecommunication system (TS 51.011)



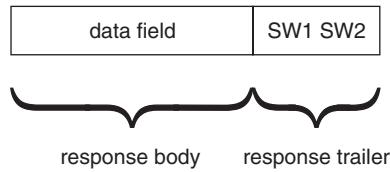
**Figure 8.13** Structure of an extended  $L_c$  or  $L_e$  field



**Figure 8.14** The four possible cases of the command APDU

The  $L_c$  and  $L_e$  fields usually have a length of one byte, but they can be converted into fields with a length of three bytes as shown in Figure 8.13. This enables lengths up to 65 536 to be represented, since the first byte contains the escape sequence ‘00’. The standard defines this three-byte length specification as reserved for future use. However, there are already some smart card operating systems for microcontrollers with large nonvolatile memories that support three-byte length specifications.

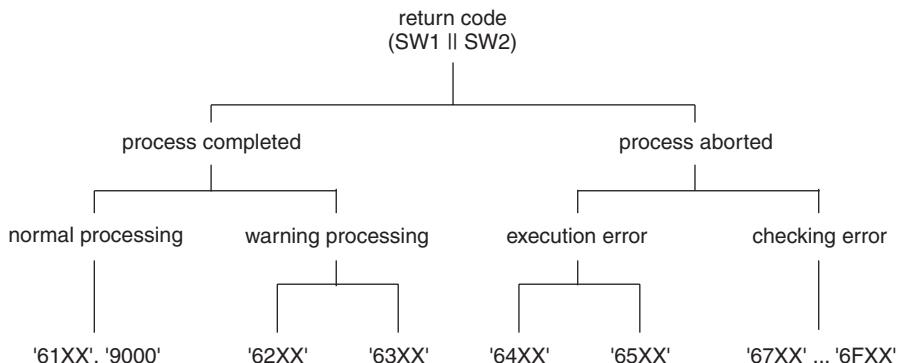
The previously described elements of the command APDU can be combined to produce four general cases, which are illustrated in Figure 8.14.



**Figure 8.15** Structure of the response APDU



**Figure 8.16** The two versions of the response APDU

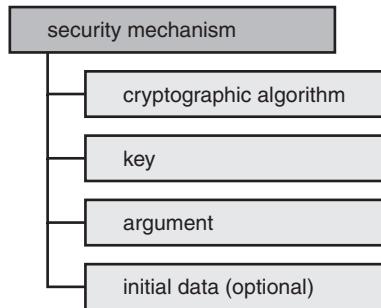


**Figure 8.17** Return code classification scheme defined by ISO/IEC 7816-4. Return codes '63xx' and '65xx' indicate that data in nonvolatile memory (EEPROM or flash memory) was modified, while the remaining '6X' codes do not indicate this

### 8.3.2 Response APDU structure

The response APDU sent by the card in reply to a command APDU consists of an optional body and a mandatory trailer, as shown in Figure 8.15. The body consists of the data field, whose length is specified by the  $L_e$  byte of the preceding command APDU. Regardless of the value specified in the  $L_e$  byte, the length of the data field can be zero if the smart card terminates command processing due to an error or incorrect parameters. This is indicated by the two single-byte status words SW1 and SW2 in the trailer (see Figure 8.16).

The card must always send a trailer in response to a command. The two bytes SW1 and SW2, which are also called the return code, contain the response to the command. For example, the return code '9000' means that the command was executed completely and successfully. There are more than 50 different codes, which are classified as shown in Figure 8.17.



**Figure 8.18** The data and functions necessary for secure data transmission mechanisms

If a ‘63xx’ or ‘65xx’ return code is received after a command has been executed, this means that the nonvolatile memory in the card (EEPROM or flash memory) was modified. If any other return code starting with ‘6x’ is received, it means that command execution was terminated prematurely without any changes in nonvolatile memory.

It should be noted here that although there is a standard for return codes, nonstandard codes are used in many applications. The only exception is the code ‘9000’, which almost always indicates successful processing. With all other codes, the relevant specification should always be consulted to ascertain their actual meaning.

## 8.4 SECURE DATA TRANSMISSION

All data exchanges between a terminal and a smart card are performed using digital electrical pulses on the I/O line of the smart card. It is relatively easy to solder a wire to the I/O contact, record all the communication activities during a session, and subsequently analyze them. This provides knowledge of all the data transferred in both directions.

A somewhat more difficult task is to electrically insulate the I/O contact, fit a dummy contact on top of it, and connect both contacts to a computer using thin wires attached to the contacts. With this arrangement, it is easy to allow only certain commands to reach the card or insert user-defined commands in the communication sequence.

These two typical forms of attack can succeed only if confidential data is transferred over the I/O line without any protection. Consequently, data transmission should always be designed such that an attacker cannot derive any advantage from eavesdropping on data transmission and/or inserting message blocks in the protocol.

Various mechanisms and techniques, which are collectively known as secure messaging (SM), can be used to defend against these attacks and even more sophisticated forms of attack (see Figure 8.18). They have been used for a long time in the field of data telecommunication and are not specific to smart cards. What is specific to the smart card realm is that the computing power of the communicating parties and the transmission rate are both relatively low. Consequently, the commonly used standard methods are downscaled to match the capabilities of smart cards, without in any way reducing the security of these methods.

The objective of secure messaging is to ensure the authenticity, and if necessary the confidentiality, of part or all of the transmitted data. A variety of security mechanisms are used to

meet this objective. A security mechanism is defined as a function that requires the following items: a cryptographic algorithm, a key, an argument, and initial data as necessary. A general condition must also be satisfied, which is that all security mechanisms must be entirely transparent with respect to existing protocol layers in order to ensure that existing standardized processes are not impaired by secure messaging. This applies in particular to the T = 0 and T = 1 transmission protocols, as well as the generally used and standardized commands for smart cards.

Before a secure messaging method can be used, both parties must agree on the cryptographic algorithm to be used and a shared secret key for this algorithm. In accordance with Kerckhoff's principle, the security of the algorithm relies entirely on this key. If it is revealed, secure messaging is reduced to nothing more than a generally known supplementary checksum that reduces the effective data transmission rate and at best can be used to detect transmission errors.

A variety of secure messaging methods have been known for many years. They were originally relatively rigid and tailored to specific applications. This generally did not lead to any objections with regard to security, but none of these methods has achieved sufficient worldwide use or demonstrated sufficient flexibility to cause it to be incorporated in an international standard.

The requirements of transparency with respect to existing commands, operation with two different transmission protocols, and the highest possible adaptability have resulted in the standardization of a very flexible but correspondingly complex and elaborate secure messaging method in ISO/IEC 7816-4, along with extended functions for this method in ISO/IEC 7816-8.

Other methods for cryptographic protection of data transmission have also become established. In the telecommunication sector, a very powerful and comprehensive method is specified in TS 43.048. It is primarily intended to protect end-to-end communication between the SIM or USIM and the background system. Global Platform also specifies secure messaging methods that provide a level of security comparable to that of the previously mentioned methods. They are designated Secure Channel Protocol 1 and Secure Channel Protocol 2. The CEN 14890 standard specifies a secure messaging method that is primarily intended to be used in the digital signature environment. If TCP/IP is used as the logical communication layer, Secure Socket Layer (SSL) or Transport Layer Security (TLS) is the logical choice for secure messaging.

Secure messaging in accordance with ISO/IEC 7816-4 is based on embedding all user data in TLV-coded data objects. Three different types of data objects are defined:

- data objects for plaintext, which hold data in plaintext form (such as the data section of an APDU);
- data objects for security mechanisms, which hold the results of a security mechanism (such as a MAC);
- data objects for auxiliary functions, which hold control data for secure messaging (such as the padding method used).

The class byte indicates whether secure messaging is used for the command. The two available bytes can code whether the method specified in ISO/IEC 7816-4 and ISO/IEC 7816-8 is used, and whether the header is also included in the cryptographic checksum (CCS).<sup>9</sup> If

<sup>9</sup> See also Section 8.3.1, 'Command APDU structure', on page 221 regarding the coding of the class byte

**Table 8.31** Tags for plaintext data objects

Tag	Meaning
'B0', 'B1'	BER-TLV coded; contains data objects included in secure messaging
'B2', 'B3'	BER-TLV coded; contains data objects not included in secure messaging
'80', '81'	No BER-TLV coded data
'99'	Secure messaging state data

**Table 8.32** Tags of data objects used for confidentiality

Tag	Meaning
'82', '83'	Cryptogram tag; the plaintext is BER-TLV coded and includes data objects for secure messaging
'84', '85'	Cryptogram tag; the plaintext is BER-TLV coded and does not include data objects for secure messaging
'86', '87'	Tag for the padding method used: 01: padding with '80 00 ...' 02: no padding

the header is included in the computation, it is authentic because it cannot be modified during transmission without this being noticed.

### 8.4.1 Data objects for plaintext

According to the standard, all data that is not BER-TLV coded must be encapsulated, which means it must be embedded in data objects. Various tags are used for this purpose, as listed in Table 8.31. Bit 1 of each tag indicates whether the data object is included in the computation of the cryptographic checksum. If this bit is not set (e.g. 'B0'), the data object is not included in the computation, while if it is set (e.g. 'B1'), the data object is included.

### 8.4.2 Data objects for security mechanisms

Data objects for security mechanisms are divided into objects used for authentication and objects used for confidentiality. The tags defined for this purpose are listed in Tables 8.32 and 8.33 on the following page.

In this context, all data objects related to cryptographic checksums and digital signatures are classified under authentication, while data encryption and the tags necessary for this purpose in secure messaging are classified under confidentiality. The tags used for secure messaging must be selected from the two tables above according to the method that is used.

**Table 8.33** Tags of data objects used for authentication

Tag	Meaning
'8E'	Cryptographic checksum.
'9A', 'BA'	Initial value for a digital signature.
'9E'	Digital signatures.

### 8.4.3 Data objects for auxiliary functions

Data objects for auxiliary functions are used in secure messaging to achieve agreement on general conditions. The two parties use these data objects to exchange information about the cryptographic algorithms and keys used, initial data, and similar basic data. In theory, these items can vary from one transmitted APDU to the next or even have different values for the command and the response. In practice, data objects for auxiliary functions are used only rarely because all of the general conditions for secure messaging are defined implicitly and do not have to be defined specifically during the communication process.

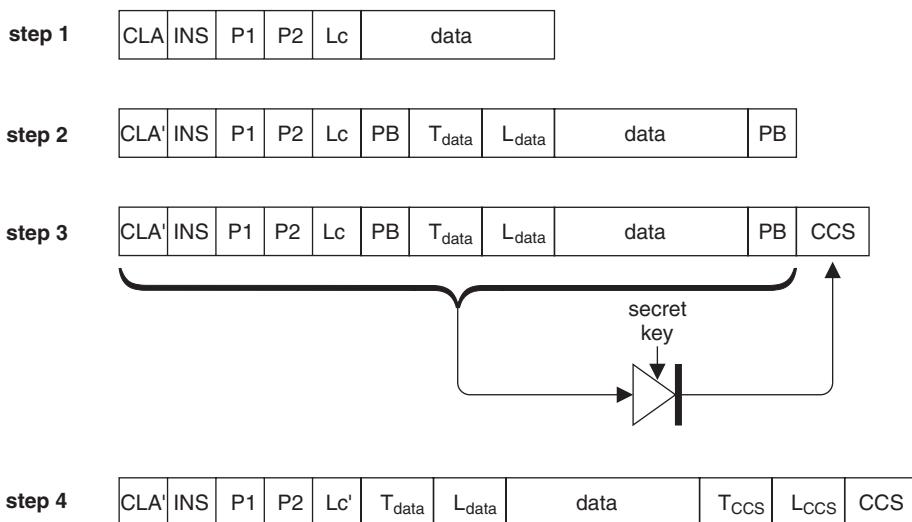
Based on the available options for secure messaging as specified in ISO/IEC 7816-4 and ISO/IEC 7816-8, which are briefly described above, here we describe two basic procedures. We have kept them as simple as possible in order to make it easier to understand these complex mechanisms. Due to the high flexibility of the defined method, there are many other possible combinations of security mechanisms, some of which are even more elaborate. The two procedures described here represent a sort of middle ground that combines simplicity with achievable security.

In the authentic mode procedure, a cryptographic checksum (CCS or MAC) is used to protect the application data (APDU) against manipulation during transmission. In the combined mode procedure, by contrast, the application data is fully encrypted so that an attacker cannot deduce anything regarding the data contained in the transmitted commands and the received responses. Utilization of a send sequence counter is shown for only one of these modes. This counter, whose initial value is a random number, is incremented with each command and each response. This enables both parties to determine whether a command or response has been lost or inserted. If a send sequence counter is used with the combined mode procedure, originally identical APDUs appear to be different. This effect is called diversity.

### 8.4.4 The authentic mode procedure

The authentic mode procedure guarantees authentic transmission of the APDUs, which means that the APDUs are protected against manipulation during transmission. The recipient of an APDU, which is a command or a response, can determine whether it has been altered during transmission. This makes it impossible for an attacker to modify data within an APDU without this being noticed by the recipient. Figure 8.19 on the next page shows the basic process for generating a command APDU in this mode.

A bit in the class byte indicates that the authentic mode procedure is being used, so that the recipient can act accordingly and check the received APDU for authenticity. The actual APDUs are sent in plaintext and are not encrypted. The transmitted data is thus still public, and



**Figure 8.19** Command APDU generation with the authentic mode procedure. Here a Case 3 command (such as UPDATE BINARY) is used, with its header included in the calculation of the cryptographic checksum (CCS). The response APDU is generated in a similar manner. The abbreviation PB indicates the padding bytes. Step 1 shows the initial format of the APDU, which is converted into TLV-coded data in step 2 with the data objects padded to an integer multiple of eight bytes. The cryptographic checksum (CCS) is calculated in step 3, and in step 4 a TLV-coded data object containing the CCS is appended to the APDU

with suitable manipulation of the transmission channel it could be intercepted and evaluated by an attacker. This is not necessarily a disadvantage, especially considering that transmitting confidential data over a public channel is undesirable with regard to data privacy legislation. With unencrypted data, the card user is at least theoretically given an opportunity to see what data is being exchanged between the user's smart card and the terminal.

In principle, any block encryption algorithm can be used to compute the cryptographic checksum. For practical reasons, in the following description we assume that AES is used with a fixed eight-byte block length. This means that the individual data objects must be extended to an integer multiple of eight bytes, which is known as padding. In this process, data objects that are already an integer multiple of eight bytes are nevertheless extended by one block. After padding, the cryptographic checksum (CCS) of the entire APDU is computed using the AES algorithm in CBC mode. This eight-byte checksum, with the four least significant bits omitted, is appended to the APDU as a TLV-coded data object. All padding bytes are deleted after the checksum has been computed. The reformatted APDU generated using this procedure is then sent over the interface. This procedure extends the length of the APDU by eight bytes, which only marginally reduces the effective data transmission rate with the usual transmission block sizes.

The algorithm that is used and the padding method can be explicitly indicated in the data objects of the control structures. Here we assume for the sake of simplicity that the smart card and the terminal implicitly know all the parameter values of the secure messaging method that is used. When the protected APDU arrives at the recipient, the latter again pads it to an integer

multiple of eight bytes and then computes the MAC of the APDU. By comparing the MAC it has generated with the received MAC generated by the sender, the recipient can determine whether the APDU has been altered during transmission.

A prerequisite for computing a cryptographic checksum is a secret AES key that is known to both parties. If this key were not secret, an attacker could breach authentic mode communication by intercepting an APDU, modifying it as desired, and computing a new MAC using the appropriate algorithm. After this, the attacker could simply replace the original MAC with the new one and send the newly generated APDU on its way.

In order to better protect the key used to generate the MAC against attacks based on known plaintext–ciphertext pairs, it is common practice to use dynamic keys. These keys are generated by encrypting a random number that has been previously exchanged between the terminal and the card. A secret key known to both parties is used for this encryption.<sup>10</sup>

The additional steps that are necessary for the transmission and reception of APDUs protected by the authentic mode procedure naturally reduce the effective data transmission rate. On average, it can be assumed that the resulting data rate will be approximately half the rate without protection.

#### 8.4.5 The combined mode procedure

The combined mode procedure represents the next higher level of security relative to authentic mode. In the combined mode, the data section of the APDU is transmitted in encrypted form instead of plaintext as in authentic mode. This procedure, which is shown in Figure 8.20 on the facing page, is an extension of the authentic mode procedure.

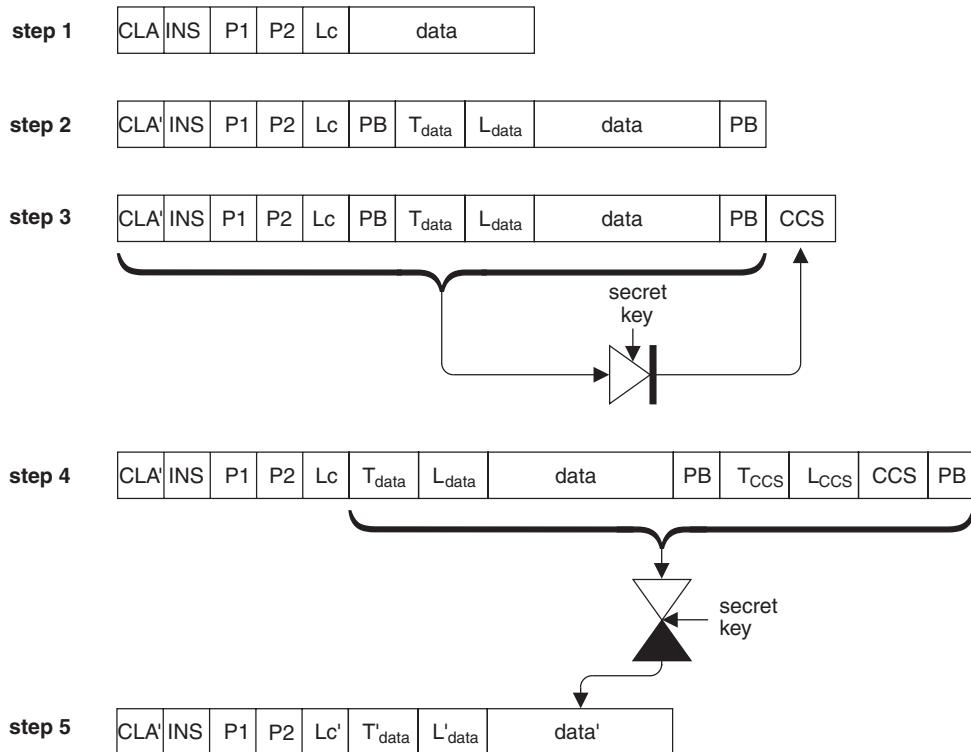
In the combined mode procedure as in the authentic mode procedure, the data objects to be protected with a cryptographic checksum are first padded to an integer multiple of eight bytes and then encrypted using the AES algorithm in CBC mode. To maintain compatibility with the T = 0 protocol, the header is excluded from this process. If it is desired to encrypt the header as well, so that the command being sent the card is unrecognizable, the T = 0 ENVELOPE command must be used. A bit in the class byte indicates that secure messaging is being used. After encryption, the data is transmitted over the interface. The recipient knows the secret key that was used for encryption and can use this key to decrypt the APDU. The recipient then checks the decryption for correctness by recomputing the appended cryptographic checksum in the same level of the transmission layer.

When this procedure is used, an attacker eavesdropping on the I/O line cannot discover which data is being exchanged between the card and the terminal in the commands and responses. It is also not possible to replace one of the encrypted blocks within the APDU, since the blocks are linked to each other by using the AES algorithm in CBC mode. Any replacement would be noticed immediately by the recipient.

With regard to the cryptographic algorithm, the remarks in the description of the authentic mode procedure apply here as well. In principle, any block encryption algorithm can be used. Just as with authentic mode, the keys should be dynamic, which means that a derived session key should be used in each session.<sup>11</sup> If only the security benefits are considered, general use

<sup>10</sup> See also Section 7.7.4, ‘Dynamic keys’, on page 182

<sup>11</sup> See also Section 7.7.1, ‘Derived keys’, on page 181



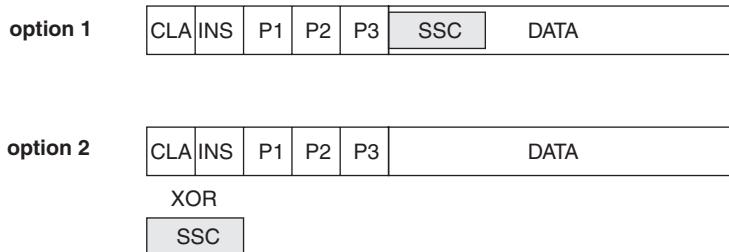
**Figure 8.20** Command APDU generation with the combined mode procedure. Here a Case 3 command (such as UPDATE BINARY) is used, with its header included in the calculation of the cryptographic checksum (CCS). The response APDU is generated in a similar manner. The abbreviation PB indicates the padding bytes. Step 1 shows the initial format of the APDU, which is converted into TLV-coded data in step 2 with the data objects padded to an integer multiple of eight bytes. The cryptographic checksum (CCS) is calculated in step 3, and in step 4 the data section of the APDU is encrypted. Finally, the modified APDU is transmitted in step 5

of the combined mode procedure for all APDUs would be worthwhile. However, increased security is accompanied by a reduction in the effective transmission rate.

The loss of effective transmission capacity is considerable. The transmission times of unprotected APDUs and APDUs protected using the combined mode procedure can be assumed to differ by a factor close to 4. The difference between authentic mode and combined mode thus amounts to a factor of 2. Accordingly, every situation should be examined carefully on a case-by-case basis to determine which data needs to be transmitted in such a secure but time-consuming manner.

#### 8.4.6 Send sequence counter

Using a send sequence counter for secure data transmission does not by itself constitute a security mechanism. It only makes sense to use a send sequence counter in combination with



**Figure 8.21** Two options for incorporating a send sequence count in a command APDU. In the first option, the send sequence count is a TLV-coded data object in the data section. In the second option, the send sequence counter is only linked to the APDU by an XOR operation for the computation of a CCS, after which the original version of the APDU is sent

an authentic mode or combined mode procedure, as otherwise any modification of the counter value by an attacker would remain undetected.

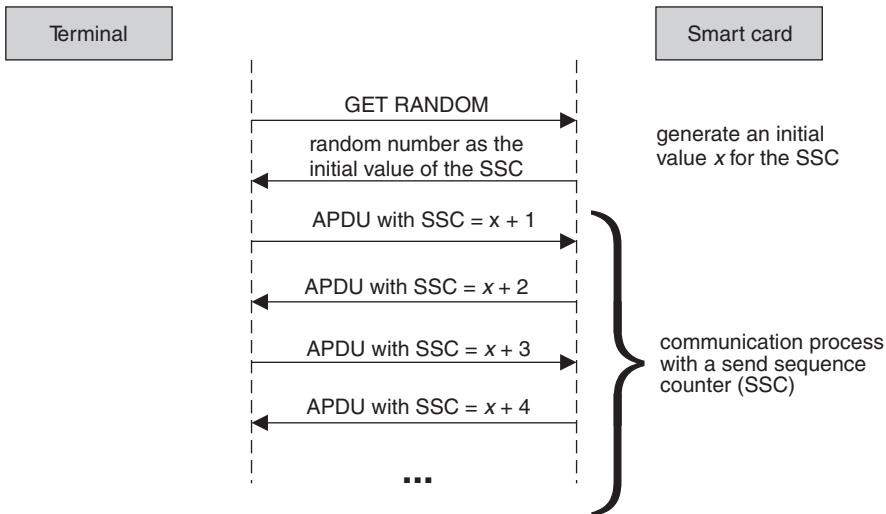
The operating principle of a send sequence counter is that each APDU includes a sequence number that depends on when it is sent. This allows the deletion or insertion of an APDU during the protocol to be noticed immediately, so that an appropriate action (such as terminating communication) can be taken by the recipient.

The mechanism is based on a counter that is initialized with a random number. This number is sent to the terminal by the card at the start of the communication process, in response to a request from the terminal. The counter is incremented each time an APDU is sent. The counter should not be too short, but it should also not be too long, since it creates additional transmission overhead. The following description assumes the commonly used value of two bytes, although longer counters may be used in practice.

As illustrated in Figure 8.21, there are two basic options for incorporating a send sequence counter in a command or response APDU. The first option is to embed the counter in the APDU as a numerical value in a data object, while the other option is to XOR the send sequence counter with the same number of bits of the APDU and then compute the cryptographic checksum of the resulting data, after which the APDU is restored to its original state. With the latter option, the recipient knows the expected value of the send sequence counter, and it can use this value to modify the APDU in the same way as the sender. It can then compute the cryptographic checksum and check the correctness of the received APDU.

Figure 8.22 depicts the sequence of events that occur during the communication process. The terminal first requests an initial counter value from the smart card. In response, the smart card returns a two-byte random number to the terminal. The terminal then sends the first security-related command to the card, augmented with the send sequence counter. Either the authentic mode or the combined mode procedure can be used to protect the counter and the command body. The smart card receives the protected APDU and first checks for an indication of manipulation as revealed by the authentic mode or combined mode procedure. It then compares the counter value with the expected value. If these values match, no APDU has been inserted or deleted in the transmission process.

It is apparent that using a send sequence counter is attractive not only when several commands must be processed in a particular order, but also with isolated commands, since each session is rendered unique when a send sequence counter is used. A counter primarily provides protection against replaying previously sent APDUs or deleting APDUs.



**Figure 8.22** APDU transmission using a send sequence counter (SSC)

If a send sequence counter is used together with the combined mode procedure, each encrypted block is different, which creates a condition known as diversity. This is a consequence of incrementing the counter for each APDU and the fact that, with a good encryption algorithm, changing a single bit in the plaintext affects the appearance of the entire ciphertext block.

## 8.5 LOGICAL CHANNELS

In smart cards with several independent applications, it is possible to access these applications via logical channels. When logical channels are used, up to 19 applications in a single card can exchange data directly with the terminal. The existing single serial interface is still used, but the applications can be addressed individually at the logical level.<sup>12</sup>

For reasons of compatibility with the previous version of the method defined in ISO/IEC 7816-4, a two-stage method is used to distinguish which command belongs to which application. Two bits in the class byte can be used to address four logical channels, which means that up to four sessions with associated applications can run in parallel in the smart card.<sup>13</sup> For the second stage, the standard specifies a special class byte with bit 8 set to 0 and bit 7 set to 1, whose lower nibble (bits 1 to 4) addresses logical channels. However, there is a limitation with regard to communicating with the various applications in the smart card, which is that the external processes that access the card must be coordinated with each other and are not allowed to interleave their commands and responses, since the response APDUs from the card do not identify the logical channel associated with the originating command. This means that it is impossible to determine externally which return code has been sent back in response to which command. Due to this absence of channel identification, no new command can be sent until the response to the previous command has been received.

<sup>12</sup> See also Section 11.12, ‘Commands for Data Transmission’, on page 398

<sup>13</sup> See also Section 8.3.1, ‘Command APDU structure’, on page 221

This powerful mechanism is used primarily to enable several applications to be used in parallel. This can be illustrated by an example scenario. Suppose a card user is conducting a telephone conversation and wants to send the opposite party a document in secure form during the conversation. The first logical channel is opened for the GSM application, where it is used for cyclic authentication of the SIM with respect to the mobile telecommunication network (among other things). The second logical channel is opened for another application in the multiapplication smart card that provides all the mechanisms and keys necessary for secure communication with another party. This would typically be a WIM application.<sup>14</sup> Another example would be securely transferring electronic funds while conducting a phone conversation at the same time.

Although logical channels may appear very useful, they make the management tasks of the smart card operating system considerably more complicated. Each logical channel essentially represents a separate smart card with all of its states and conditions. This means that the operating system must concurrently manage all the data in its memory pertaining to several sessions running in parallel. The cost of this should not be underestimated, and it requires microcontrollers with large RAM capacities. If secure messaging and all possible types of authentication are also required for each logical channel, the memory demand very quickly rises to a level that can only be met by high-performance smart card microcontrollers.

## 8.6 LOGICAL PROTOCOLS

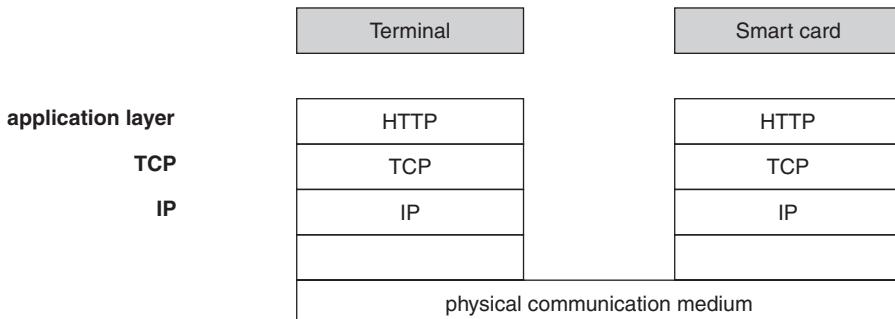
In addition to the transmission protocols that have been used for many years, such as T = 0 and T = 1, the TCP/IP and HTTP protocols, which are well established in the Internet world, began to penetrate the smart card world at the start of the new millennium. They are not used extensively at present, but these widely used protocols will become increasingly common in the smart card world due to the steadily increasing integration of smart cards in typical IT infrastructures. The main features of the relevant protocols are described briefly below. The associated literature should be consulted for more detailed information.

### 8.6.1 TCP/IP protocol

The Transmission Control Protocol (TCP) and the Internet Protocol (IP) jointly form part of the foundation of Internet communication. Both of these protocols were developed in the 1970s. The protocol stack with its four layers (application, transport, Internet, and link) is shown in Figure 8.23 on the next page.

The IP protocol belongs to the Internet layer in the layer model of the Internet protocol. It is a packet-switched protocol that operates reliably, which among other things means that incorrect data packets do not have to be requested anew. The functions provided by the IP protocol are unique addressing and grouping of computers in a network, forwarding data packets, and routing data packets. Version 4, which is specified in RFC 971 and often abbreviated as IPv4, is most commonly used at present. It has an address space of  $2^{32}$ , which is represented by four three-digit decimal numbers. As the 32-bit address space of IPv4 has now become to small

<sup>14</sup> See also Section 19.6, ‘The wireless identification module’, on page 854



**Figure 8.23** The TCP/IP protocol stack and the HTTP protocol based on this stack. The layer between IP and the physical communication medium depends on the individual operating system

in many regions, a gradual migration to the compatible IP Version 6 protocol is underway. Version 6, or IPv6 for short, is specified in RFC 2460 and has a 48-bit address space. Both versions are used with smart cards.

The link layer below the IP layer can take many forms, as is also true in the Internet. In the simplest case, it can consist of encapsulation of the IP packets in the data section of APDUs, followed by transmission using special commands with the T = 0 or T = 1 protocol. However, the usual approach is to use a USB interface and a suitable device driver to implement the TCP/IP link to the smart card. In the telecommunication sector, the BIP protocol<sup>15</sup> specified in TS 102 223 is also used. It allows direct TCP/IP communication with the smart card over the air interface.

TCP is a reliable protocol that enables error-free communication between two computers. It is a circuit-switched protocol that establishes a virtual channel between two endpoints. Data can be transmitted in both directions. TCP is accordingly assigned to the transport layer. TCP is specified in RFC 793 and usually utilizes the IP protocol. The TCP protocol provides the next higher layer with a full-duplex virtual connection, with which (among other things) transmission errors are corrected automatically by requesting retransmission of the affected data packets.

Connections are established with TCP by specifying two endpoints, each of which consists of an IP address and a port. These endpoints are also called sockets, and they form the interface to the next higher layer (such as HTTP). Ports are identified uniquely by 16-bit numbers. As a result, a TCP connection is defined uniquely by the IP addresses of the source and the destination and the two associated port numbers.

## 8.6.2 HTTP protocol

The Hypertext Transfer Protocol (HTTP) belongs to the application layer in the Internet protocol model and is specified in RFC 1945 (Hypertext Transfer Protocol – HTTP/1.0) and RFC 2616 (Hypertext Transfer Protocol – HTTP/1.1). In principle, it can be used to transfer

<sup>15</sup> See also Section 8.6.3, ‘BIP protocol’, on page 236

any desired type of data, although it is predominantly used to transfer HTML pages on the Internet. Figure 8.23 on the preceding page shows its position in the Internet protocol stack.

In operation, the HTTP protocol must be supported by a reliable underlying protocol – in other words, a protocol that provides error-free data blocks to the overlying layer. The TCP protocol is used almost exclusively for this purpose. HTTP operates as a connectionless protocol, which among other things implies that the connection between the two communication parties is not maintained after a data transfer takes place and it may be necessary to store data in the overlying layer across session boundaries.

The data blocks exchanged between servers and clients are called messages in the HTTP context. This process always takes the form of a request sent by the client to the server, which in turn sends a response to the client. Each message consists of two parts: the message header and the message body. The message header holds the administrative data for the message body, which holds the user data to be transferred.

The operating principle of the HTTP protocol is rather simple. The client sends an HTTP request to a server, which in turn returns the requested data. This is a pure master–slave relationship. It is also possible to transfer data from the client to the server by using suitable commands.

Only a few HTTP commands are necessary for displaying typical HTML pages. The client uses the GET command to request data from the server, such as an HTML page. The POST command, which can be used to transfer additional data to the server, operates in a similar manner. The PUT command allows data to be loaded from the client to the server, while the DELETE command allows data to be deleted. These two commands are rarely used on the Internet, but they play a distinct role in smart card web servers.<sup>16</sup>

Data is transmitted in plaintext form with the HTTP protocol, which is not tolerable for security reasons in many applications. The Hypertext Transfer Protocol Secure (HTTPS) extension enables cryptographically secured data transmission that provides protection against interception and manipulation of the data. For this purpose, an additional protocol layer is inserted between HTTP and the underlying TCP. It is called Transport Layer Security (TLS), and it is specified in RFC 2246. The predecessor of TLS is called Secure Socket Layer (SSL), for which reason this layer for cryptographic protection of data transfer is often called SSL/TLS. HTTP over TLS is specified in RFC 2818. The cryptographic algorithms commonly used with SSL/TLS are MD5, SHA-1, triple DES, AES, and RSA.

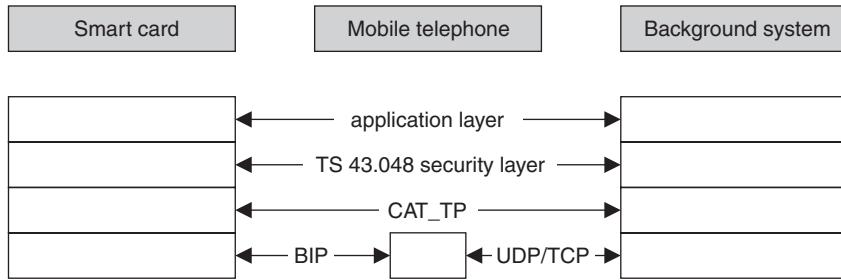
### 8.6.3 Bearer Independent Protocol (BIP)

Originally, the only option for data transfers between SIM cards and the background system was the Short Message Service (SMS). In the late 1990s, it became clear that the resulting data rate was far from being adequate for future requirements. For this reason, the Bearer Independent Protocol (BIP) was specified in the TS 102 223 standard.

BIP is based on the proactive commands<sup>17</sup> OPEN CHANNEL, CLOSE CHANNEL, SEND DATA, RECEIVE DATA, and GET CHANNEL STATUS, and it enables a smart card in a mobile telephone to use CSD, GPRS, IrDA, or Bluetooth as a bearer service for exchanging data with an associated system. The BIP protocol is limited to the interface between the SIM and the mobile equipment. Communication over the air interface with a background

<sup>16</sup> See also Section 23.3, ‘Smart Card Web Server (SCWS)’, on page 912

<sup>17</sup> See also Section 19.4.4.6, ‘SIM Application Toolkit’, on page 833



**Figure 8.24** Overview of the relationship between the BIP and CAT\_TP protocols for data transmission between the SIM, the mobile equipment, and the background system

system employs the User Datagram Protocol (UDP) specified in RFC 768 or TCP, as illustrated in Figure 8.24.

A transport layer based on BIP is also specified. It enables direct communication between the SIM card and the background system and utilizes BIP. This is called the Card Application Toolkit Transport Protocol (CAT\_TP). CAT\_TP allows the smart card as well as the background system to establish a connection, which makes it well suited to integrating SIM cards in a larger system. Another protocol specified in TS 43.048 can be used for cryptographic protection of the connection. This protocol allows data to be transmitted in both directions in either authentic or encrypted form. The BIP protocol also forms the basis for linking a smart card web server<sup>18</sup> to an HTML browser in the mobile equipment.

## 8.7 CONNECTING TERMINALS TO HIGHER-LEVEL SYSTEMS

In order to use smart cards in a PC environment, it is necessary to have a terminal that is connected to the PC and supported by the PC software. In the past, a separate software driver had to be installed on the PC for each type of terminal. Each driver also had its own software interface, so it was practically impossible to produce terminal-independent application software. In the mid-1990s, work began on the development of specifications of interfaces suitable for terminal-independent integration of smart cards with PC applications. This occurred in various countries and was initiated by a wide variety of organizations. Two industry standards have attained a strong position worldwide: Personal Computer/Smart Card (PC/SC) and Open Card Framework (OCF). The Multifunktionales Kartenterminal (MKT) specification has also been used for many years in Germany and some other countries, and it has become surprisingly popular in the German-speaking world. All three of these specifications are described in summary form below, and they can be obtained free of charge via the Internet.

### 8.7.1 PC/SC

The first efforts to generate an international specification for connecting cards to PCs began in May 1996. The companies Bull, Hewlett-Packard, Microsoft, Schlumberger, Siemens

<sup>18</sup> See also Section 23.3, 'Smart Card Web Server (SCWS)', on page 912

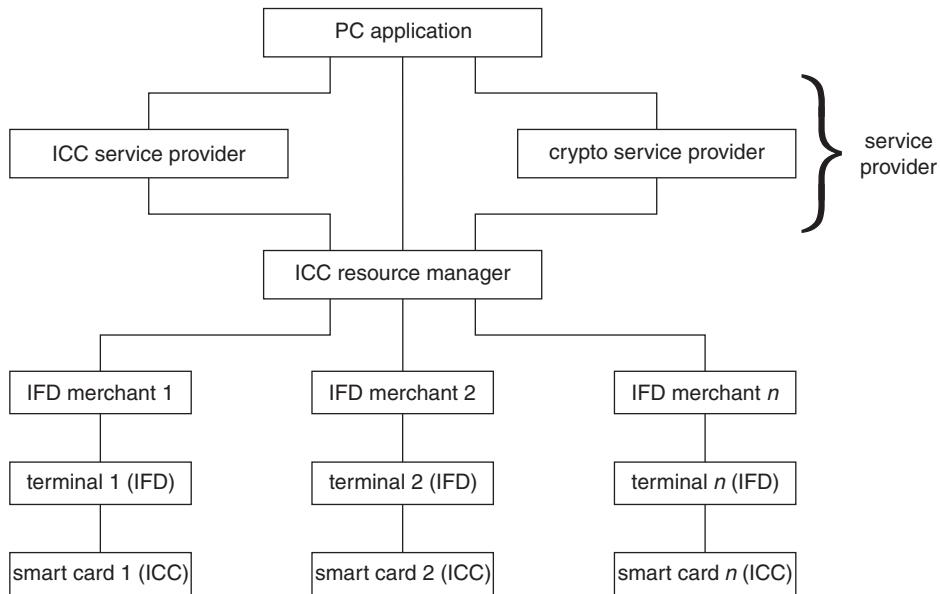
Nixdorf, Gemplus, IBM, Sun, Verifone, and Toshiba participated in the development of this specification.

Version 1.0 of the Interoperability Specification for ICCs and Personal Computer Systems was published in December 1997. It consists of eight parts, as described in Table 8.34, which are related to each other as shown in Figure 8.25. The working group was known as PC/SC (short for ‘personal computer/smart card’), and this abbreviation is commonly used to refer to the specification. It is available on the Internet from the WWW server of the specification group [PC/SC].

The PC/SC specification has at least a certain degree of platform independence because it is compatible with all Windows-based PCs, which make up the majority of personal computers. It allows smart cards to be integrated into any desired application in a manner that is largely independent of the programming language, since it supports widely used languages such as C, C++, C#, Java, and Basic. The only prerequisites are that a suitable driver must be available for the terminal concerned and the smart card must be compatible with PC/SC. The required degree of compatibility is fairly relaxed because the constraints have been kept relatively broad.

**Table 8.34** Summary of the eight parts of the PC/SC specification

PC/SC specification	Content
Part 1: Introduction and Architecture Overview	This is the basis for all other parts of the specification. It identifies the relevant standards, provides an overview of the system architecture and the hardware and software components, and lists definitions and acronyms.
Part 2: Interface Requirements for Compatible IC Cards and Readers	This defines the physical characteristics of contact smart cards. It specifies basic electrical properties, such as the supply voltage and the reset behavior, and defines the data elements, structures, and allowed processes of the ATR and PPS. The basics of data transfer at the physical level and the T = 0 and T = 1 protocols are described briefly.
Part 3: Requirements for PC-Connected Interface Devices	Describes the requirements on the terminal and the supported terminal features (display, keypad, etc.).
Part 4: IFD Design Considerations and Reference Design Information	Information for designing terminals, with reference to the PS/2 keyboard interface and the USB interface.
Part 5: ICC Resource Manager Definition	Detailed descriptions of the technical aspects of the ICC resource manager, including the associated classes.
Part 6: ICC Service Provider Interface Definition	Detailed descriptions of the software aspects of the ICC service provider and crypto service provider, including the associated classes.
Part 7: Application Domain and Developer Design Considerations	Describes the use of the PC/SC specification from the application perspective.
Part 8: Recommendations for ICC Security and Privacy Devices	A compilation and specification of recommended functions and mechanisms that should be supported by a PC/SC smart card. This includes the file system (MF, DF and EFs), the associated file access conditions, the necessary system files in the smart card (for keys, PINS, etc.), commands, return codes, and cryptographic algorithms.



**Figure 8.25** Overview of the software architecture of the PC/SC specification for linking smart cards to PC operating systems

The easiest way to gain a general understanding of the PS/SC specification is to view it in terms of the defined hardware and software components. The following seven components are described here in terms of their functions and mutual interfaces: ICC-aware application, ICC service provider, crypto service provider, ICC resource manager, IFD handler, IFD, and ICC. The current version of PC/SC also supports connections to terminals for contactless data transmission compliant with ISO/IEC 14443 Type A and Type B.

### **8.7.1.1 ICC-aware application**

This software component is an application that runs on a PC and wishes to use the functions and data of one or more smart cards. It can also be an application that is able to run under a multiuser operating system with multitasking and multithreading.

#### **8.7.1.2 Service provider**

The function of the service provider is to encapsulate the individual functions of a smart card, independent of the smart card operating system. For example, a file can be selected via the application programming interface (API) of the service provider without any need for the user to know the corresponding smart card command or any details of its coding.

The service provider component is split into the ICC service provider and the crypto service provider. This is done to avoid problems with export restrictions that many countries have with regard to cryptographic algorithms. The separate crypto service provider, which handles all

functions that can cause export problems, can be omitted. In this case, the PC/SC interface can be used for all functions except cryptographic functions.

The service provider does not have to be monolithic software. It can also consist of multiple software components linked by a network. For example, it is possible to locate the crypto service provider on a cryptographically secure or high-performance computer that is isolated from the rest of the PC/SC components.

#### **8.7.1.3 ICC resource manager**

The ICC resource manager is the most important component of the PC/SC architecture. It manages all resources that are necessary for linking smart cards to the operating system. It must provide three important functions.

First, it is responsible for recognizing connected terminals and smart cards. It must also recognize when a smart card has been inserted or removed from a terminal, and respond to such events by providing suitable messages.

Its second function is to manage the allocation of terminals to one or more applications. For this purpose, a terminal resource can be assigned exclusively to a particular application. However, if several applications access the same terminal concurrently, the terminal must be identified and managed by the ICC resource manager as a shared resource.

The third function is to provide transaction primitives. A transaction primitive comprises a group of commands used to perform a particular function. This ensures that these commands are executed in an uninterrupted sequence. Otherwise, it would be possible for two applications to access a smart card concurrently without mutual coordination, each with its own command sequence.

The problems that this would cause can most easily be illustrated by the following example. In a smart card, only one file can be selected at a time. If two different applications attempt to select different files at the same time with the SELECT command and then read data from the smart card with a read command (such as READ BINARY), the file that will actually be read is entirely uncertain. This depends only on the order in which the commands arrive at the smart card. A far more complicated situation, which is just as treacherous, arises when complex processes involving several applications that interact with a single smart card (such as paying with an electronic purse) are necessary. The ICC resource manager ensures that command sequences that belong together cannot be split up or interrupted by other commands, and so ensures that the individual processes are executed one after the other.

#### **8.7.1.4 IFD handler**

The IFD handler is a sort of driver that is specific to a particular terminal. Its tasks are to link the terminal to the specified interface of the PC and to map the individual characteristics of the terminal onto the PC/SC interface. In a manner of speaking, the IFD handler represents a data channel from the PC to a particular terminal.

#### **8.7.1.5 IFD (interface device)**

The IFD component of the PC/SC specification is a terminal connected to the PC via an interface. Any desired interface can be used; for example, the terminal can be connected to

the computer by an RS232 interface, a USB interface, or a PC Card interface (formerly called PCMCIA). The terminal must comply with the ISO/IEC 7816-1/-2/-3 standards, which among other things means that it must support both asynchronous transmission protocols ( $T = 0$  and  $T = 1$ ). Optionally, it may support synchronous transmission protocols (two-wire, three-wire, and/or I<sup>2</sup>C) for memory cards as specified by ISO/IEC 7816-10.

In the terminal, in addition to a display, the PC/SC specification supports a numeric keyboard, a fingerprint scanner, and other biometric sensors for user identification.

#### 8.7.1.6 ICC (*integrated chip card*)

An interface that complies with the PC/SC specification must support all smart cards that are compatible with the ISO/IEC 7816-1/-2/-3 standards. Memory cards compliant with ISO/IEC ISO/IEC 7816-10 can also be connected if they are supported by the terminal.

### 8.7.2 OCF

The Open Card Consortium [OCF] was founded in 1997 by a group of more than ten companies active in the smart card and PC areas. The objective was to create a PC interface for smart cards that was independent of the PC operating system (Windows, Unix, etc.) and independent of the actual application in the smart card. The result is Open Card Framework (OCF), a Java-based interface for PCs that enables applications running on PCs to access applications in smart cards.

OCF gives developers two basic options at the application level. The originally preferred approach was to group complex sequences of smart card commands into packages, which could be provided by the producer of the smart card operating system or the application developer. The advantage of this approach is that complex commands or command sequences, such as are found in electronic purse applications, can be triggered relatively easily at the application level in the PC.

However, this approach is seldom used in practice because developers of PC software like to have full control over the commands sent to the smart card. Consequently, OCF is now used primarily to send commands from Java environments to smart cards, with the commands directly coded as APDUs.

### 8.7.3 MKT

Development of a specification for a software interface between PCs and terminals began relatively early in Germany. The result was the Multifunktionales Kartenterminal (MKT) specification, which is published by Teletrust Deutschland. Several versions have been issued since the original version in 1994. It focuses primarily on the interests of the health care sector, but it now serves as the basis for many other types of terminals in Germany.

The MKT specification consists of seven parts. Part 1 describes the fundamental MKT concept and provides a basic overview of the software architecture and MKT terminals. Part 2 specifies the interface between the card terminal and the integrated chip card (CT-ICC), which is intended for contact smart cards that use synchronous or asynchronous data transmission.

Part 3 contains a description of an application-independent terminal interface called the card terminal application programming interface (CT-API). This interface is independent of any particular programming language and has a procedural structure. It provides the following three functions: CT\_init for initializing a connection, CT\_data for data exchange using an existing connection, and CT\_close for closing a connection.

Part 4 extends this functionality by specifying several basic, application-independent commands for controlling terminals, which are called the card terminal basic command set (CT-BCS).

Part 5 describes the ATR and general data fields for smart cards that use synchronous data transmission. Part 6 describes the associated transmission protocols and the corresponding general commands sent to the terminal. Based on this, Part 7 specifies the translation of ISO/IEC 7816-4 commands into commands for smart cards that use synchronous data transmission, which means memory cards.

The MKT specification was one of the first documents of its kind in the world, and it has achieved an extremely broad basis in Germany due to thousands of terminals used with 80 million health insurance cards. Although its technology is now somewhat outdated, it will remain a national industry standard for many years to come.

## 8.7.4 MUSCLE

As with all other operating systems for PCs, suitable drivers are necessary for using smart cards with Linux. In the early years of Linux, such drivers were not available, which made it rather cumbersome to use smart cards for actions such as logging in under Linux.

The first version of MUSCLE (Movement for the Use of Smart Cards in a Linux Environment), which is intended to fill exactly this gap, was published in 2000. In terms of its architecture, MUSCLE is strongly based on PC/SC, but in contrast to PC/SC its source code is openly available [MUSCLE] under a GPL license, which means that it can be modified and extended by other parties. MUSCLE defines a Linux API that allows smart cards to be accessed in a relatively uncomplicated manner using a connected terminal.

# 9

## Data Transmission with Contact Cards

Data transmission using the six or eight contacts of a smart card module is still the most commonly used form of message exchange between terminals and smart cards. The main technical advantages of data transmission using contacts instead of data transmission using radio-frequency signals are that it is significantly simpler and does not require the integration of an antenna in the card body.

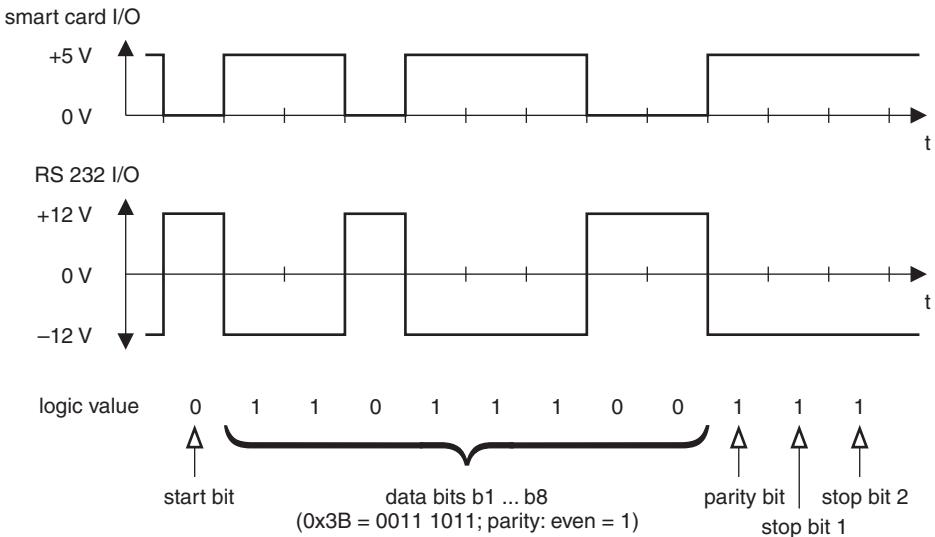
### 9.1 PHYSICAL TRANSMISSION LAYER

The general parameters of the physical transmission layer are specified in the ISO/IEC 7816-3 standard for smart cards. This is the fundamental international standard for all aspects of communication at the physical level.

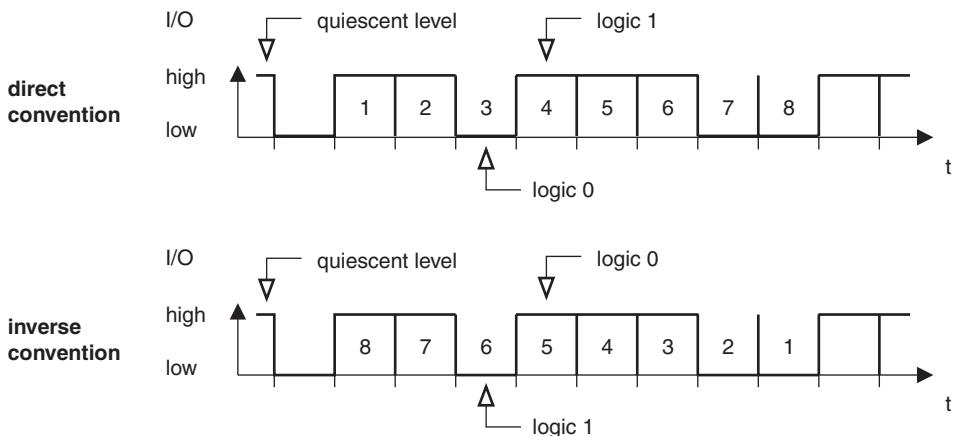
Serial communication with smart cards at the electrical level is based on the RS232 interface. This is clearly illustrated in Figure 9.1 on the following page. However, the signal levels are adapted to the usual supply voltage levels of smart cards (+5 and 0 V) and the quiescent level is inverted to correspond to the logical data state. As a result, only relatively simple hardware is needed to use smart cards with an RS232 interface.

The entire data exchange with the smart card is digital, which means it employs only the logic values 0 and 1. The voltage levels used for this are the standard values used in digital technology, namely +5, +3 and +1.8 V, with 0 V as the reference level. Assignment of the physical high or low level to the logic 1 state is freely definable, with the actual assignment being indicated by the card in the first byte of the ATR. Here ‘direct convention’ means that the high voltage level (+1.8, +3 or +5 V) represents logic 1, while ‘inverse convention’ means that the high voltage level represents logic 0 (see Figure 9.2). In either case, the I/O line is always at the high level in the quiescent state when no data is being transmitted.

Data communication between a smart card and the outside world is serial, which means that data handled by the processor in the form of bytes must be converted into a serial bit stream. To this end, each byte is separated into its eight individual bits, which are then sent



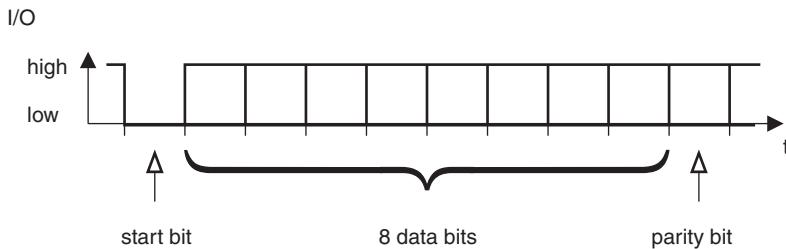
**Figure 9.1** Comparison of serial data transmission according to RS232 and according to ISO/IEC 7816-4, which clearly shows the similarity of the two methods. The essential difference is in the assignment of logic values to voltage levels. The quiescent level is  $-12\text{ V}$  with RS232 and  $+5\text{ V}$  with ISO/IEC 7816-4



**Figure 9.2** Data transmission conventions: (a) data transmission with the direct convention; (b) data transmission with the inverse convention

over the line one after the other. The bit order depends on the convention used. With the direct convention, the first data bit after the start bit is the least significant bit of the byte. With the inverse convention, the most significant bit is sent immediately after the start bit.

Data transmission between the card and the terminal is asynchronous, so each transmitted byte requires supplementary synchronization bits. As shown in Figure 9.3, the sender adds a start bit at the beginning of each transmitted byte to indicate the start of a transmission sequence to the receiver, and at the end of each byte, the sender adds a parity bit for error



**Figure 9.3** Structure of a character for data transmission

detection and one or two stop bits. The stop bit interval is called the guard time in the T = 0 protocol. The receiver and the sender can both use this time to prepare for the next transmitted byte. The parity of each byte must always be even. The parity bit is thus set to logic 1 if the number of ones in the byte is odd or logic 0 if the number of ones in the byte is even.

Smart card microcontrollers do not have timers that are independent of the applied clock signal, so it is not possible to specify an absolute time interval for individual data bits. Consequently, the bit interval is specified in terms of the clock frequency. For this purpose, a divider is defined to specify the number of clock cycles per bit interval. The duration of one bit is called an elementary time unit (etu).

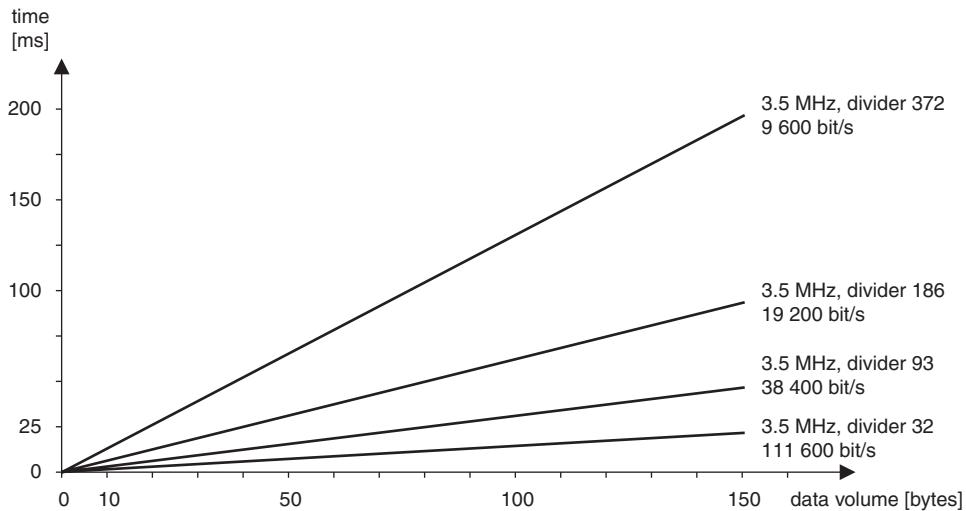
This means that the data transmission rate of smart cards cannot be stated as a fixed value such as 9600 bit/s, since the rate depends on the clock frequency. However, there are essentially only two divider values in worldwide use: 372 and 512. In modern systems, even smaller divider values are often used to increase the transmission rate. Reducing the divider value makes it more difficult for the card's operating system to receive or transmit data because the processor has less time to perform the associated tasks. For instance, if data is received using a divider value of 64, the processor has only 64 clock cycles to recognize each bit and transfer it to the I/O buffer.

To calculate the transmission rates that can be achieved with the standard divider values, we only need to know the clock rate and the divider value, as illustrated by the following examples:

$$\frac{3.5712 \text{ MHz}}{372} = 9600 \text{ bit/s} \quad \frac{4.9152 \text{ MHz}}{512} = 9600 \text{ bit/s}$$

A data transmission rate of exactly 9600 bit/s can thus be obtained with the two commonly used clock frequencies (3.5712 and 4.9152 MHz).<sup>1</sup> This transmission rate of 9600 bit/s is the reason for the ‘odd’ divider values. In the early days of smart card technology, inexpensive quartz crystals were available for only a few frequencies. Accordingly, standard crystals made for television sets were used and smart card divider values were chosen to obtain a transmission rate of 9600 bit/s, which was a common value at that time. A clock frequency of 4.77 MHz

<sup>1</sup> Here and in the rest of the book, the unit ‘bit/s’ is always used for the data transmission rate. In the literature and in quite a few standards, the unit ‘baud’ is sometimes erroneously used as equivalent to ‘bit/s’. The term ‘baud’ refers to the number of state changes per second during data transmission. Depending on the transmission method used, one or more data bits can be conveyed by each state change. For this reason, the baud rate is equivalent to the data rate in bits per second only in the special case that only one bit is conveyed by each state change



**Figure 9.4** Data transmission times with typical transmission rates, assuming 1 start bit, 8 data bits, 1 parity bit, and 2 stop bits per byte

was used in early personal computers for similar reasons – it was compatible with US color television sets, so in principle a personal computer could be connected to a television set.

If we assume that 5 MHz is the highest possible clock frequency and 32 is the minimum usable divider value, we obtain the current upper limit on the data transmission rate if this process is handled by software in the processor:

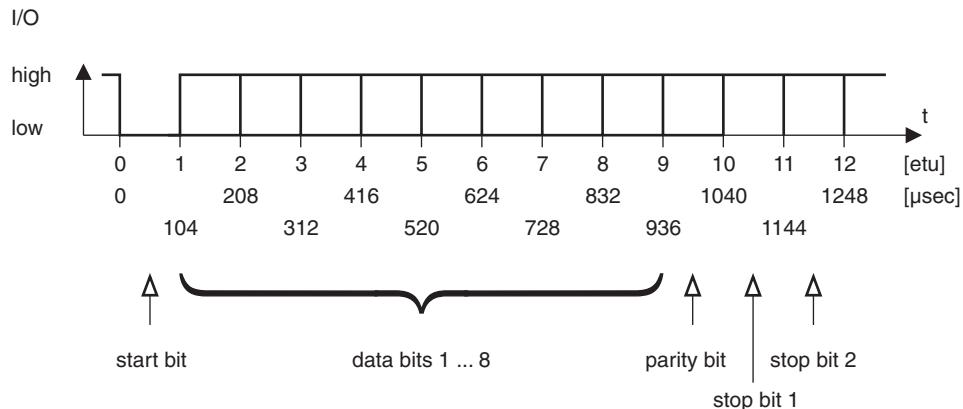
$$\frac{5 \text{ MHz}}{32} = 156\,250 \text{ bit/s}$$

Of course, it would be possible to reduce the divider value even further in order to boost the transmission capacity. However, this would significantly increase the amount of program code in the smart card, so it is not normally done due to the limited amount of available memory.

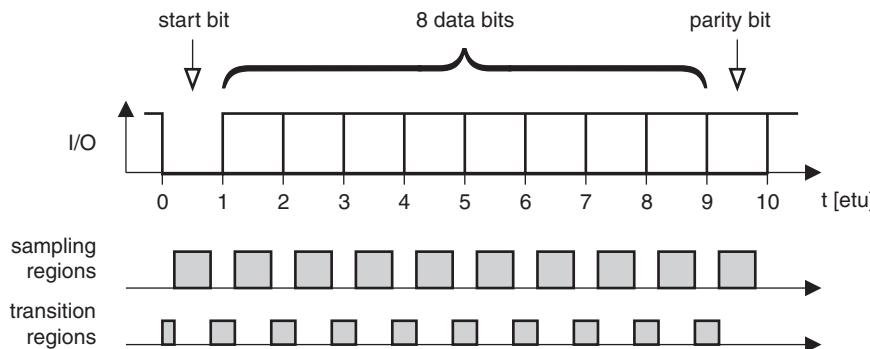
Many recent smart card microcontrollers have an on-chip hardware component (universal asynchronous receiver/transmitter, or UART) that handles data transmission via the serial interface. This reduces the amount of software needed in the card for the data transmission process, making it possible to use much higher data transmission rates. An integrated interface component of this sort can easily achieve the standardized transmission rate of 111.6 kbit/s.

The bit interval can be calculated from the card's clock frequency and the divider value. With a clock frequency of 3.5712 MHz and a divider value of 372, the bit interval is 104 µs, which by definition corresponds to one elementary time unit (etu) with this divider value. Using this information, we can construct the timing diagram shown in Figure 9.5 on the next page.

The timing of serial data transmission does not have to be exact. For technical reasons, certain tolerances are allowed. Many smart card microcontrollers do not have interface hardware, so the allowable tolerances must be utilized occasionally to enable practical implementation of the interface functions in software. The timing deviation between the falling edge of the start bit and the trailing edge of the  $n$ th bit may not exceed  $\pm 0.2$  etu. As far as the sender is concerned, this means that a timing deviation of up to  $\pm 0.2$  etu is allowed between individual



**Figure 9.5** Timing diagram for one character at 9600 bit/s, which corresponds to a clock frequency of 3.5712 MHz and a divider value of 372

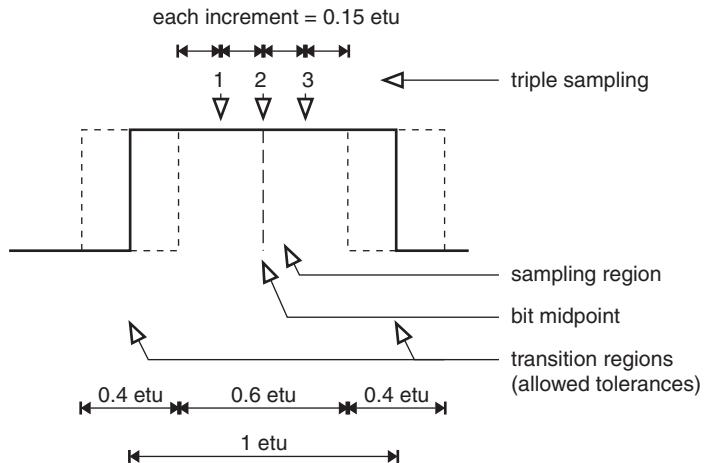


**Figure 9.6** Test zones (sampling intervals) and transition zones of a received data byte

bits, but the deviation over several bits must also remain within these limits. In other words, the net timing error over a group of bits must remain within the allowed tolerance.

Signal dropouts and overshoots are relatively common with data transmission via a physical conductor. Consequently, the received signal is sampled multiple times in defined sampling intervals (see Figure 9.6) rather than just once. Triple sampling followed by a 2-of-3 majority vote is a commonly used method. Small distortions in signal levels can thus be handled with relatively little effort. Increasing the number of samples to five or seven would make little sense, given the generally good quality of data transmission with smart cards and the amount of extra effort this would entail.

As shown in Figure 9.7, the three sampling points should be distributed as evenly as possible over the received bit interval, in order to best compensate for brief dropouts. This leads to sampling at the middle of the bit interval and at each end of the test zone defined by the maximum timing tolerances for byte transmission. The optimum sampling points for triple sampling are obtained by determining the edges of the test zone and the midpoint of the bit interval. However, this is not specified in any standard. Sampling in the transition zone is not allowed because the signal level is invalid in this zone.



**Figure 9.7** Example of triple sampling of a received bit

## 9.2 MEMORY CARD PROTOCOLS

Synchronous data transmission protocols are not used with microcontroller-based smart cards because they always communicate with terminals asynchronously. However, synchronous transmission is standard with memory cards, which are used in very large numbers in applications such as prepaid electronic purses for card phones. This widespread use justifies a description of synchronous data transmission.

Synchronous data transmission with memory cards is very closely linked to the chip's hardware and is designed to be as simple as possible. The transmission protocol does not include layer separation or logical addressing, so the application in the terminal must directly access memory locations in the chip. The protocol allows the memory the chip to be physically addressed and then read or written. This means that the data transmission process is tied to the memory addressing and memory management functions.

There is also no procedure for detecting or correcting errors during data transmission, since transmission errors between the card and the terminal occur very rarely. If the terminal application nonetheless detects a transmission error, it must re-read the relevant area in the card's memory. All these limitations result from the objective of allowing data to be transferred between the card and the terminal at high speed using only a small amount of logic circuitry.

As synchronous data transmission is only used to make data transmission as simple as possible in the chip, which means with a minimum of logic circuitry, an almost inevitable consequence is strong hardware dependence. As a result, synchronous transmission protocols are not uniform and sometimes vary greatly from chip to chip. Only the ATR is standardized. A terminal that has to communicate with various types of memory cards must incorporate a variety of synchronous data transmission protocol implementations.

The exact designation of the type of data transmission used with memory cards is 'clock-synchronous serial data transmission'. This clearly indicates the basic aspects of this type of communication. As with asynchronous communication, data is transmitted between the card and the terminal serially (bit by bit). However, the bits are transferred synchronous to a clock signal. This makes the transmission of start and stop information unnecessary.

In the case of a simple memory card, there is no error detection information, which means that neither a parity bit nor a supplementary checksum is sent with the data. The low probability of transmission errors results from the very low clock rate, which ranges from 10 to 100 kHz. As one bit is sent for each clock cycle, a clock rate of 20 kHz yields a transmission rate of 20 kbit/s. However, the effective data rate is somewhat lower because address information must also be sent to a memory card.

In order to provide an understandable explanation of synchronous data transmission with memory cards, we first need to describe some of the basic features of memory cards. In their simplest form, these cards have memories that are divided into two parts, consisting of a fixed ROM region and an EEPROM region that can be written and erased. Both regions are bit-addressable and can be freely read, while the EEPROM can also be written and erased.

The master–slave relationship is even more pronounced in memory cards than in microcontroller cards. For example, the terminal handles all physical addressing of the memory. The card itself can only globally block certain areas against erasing. This is controlled by hard-wired logic located ahead of the memory. This logic also manages the data transmission process.

### 9.2.1 Telephone chip protocol

Data transmission is illustrated here using a phone card with an Infineon SLESLE 4403 chip as an example. The memory in this IC is bit-oriented, which means that all operations are performed on individual bits. Other types of chips may have protocols that differ from the protocol described here. However, the basic data transmission principles are the same for all synchronous cards.

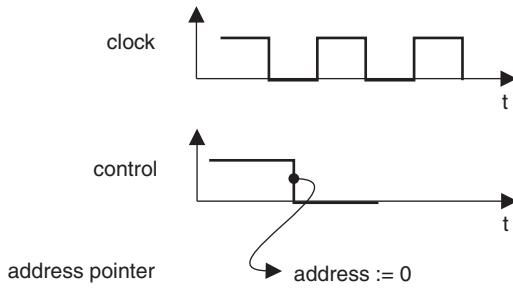
Data is transmitted using three lines. The bidirectional data line is used by the card and the terminal to exchange single-bit data. The clock line transmits the clock generated by the terminal to the card. This clock provides the timing reference for synchronous data transmission. The third connection needed for data transmission is the control line. It determines what the chip actually does, based on the states of the other two lines.

In principle, complete control of a memory card requires the chip's logic circuitry to decode four different functions: read, write, clear memory, and increment address pointer. A memory card has a global memory pointer that can be used to address all memory regions bit by bit. If the pointer reaches the upper memory boundary, it rolls over to zero. With a bit-oriented chip design, it then points to the first bit in memory. One of the functions of synchronous data transmission is to reset this pointer to an initial value, which is normally zero.

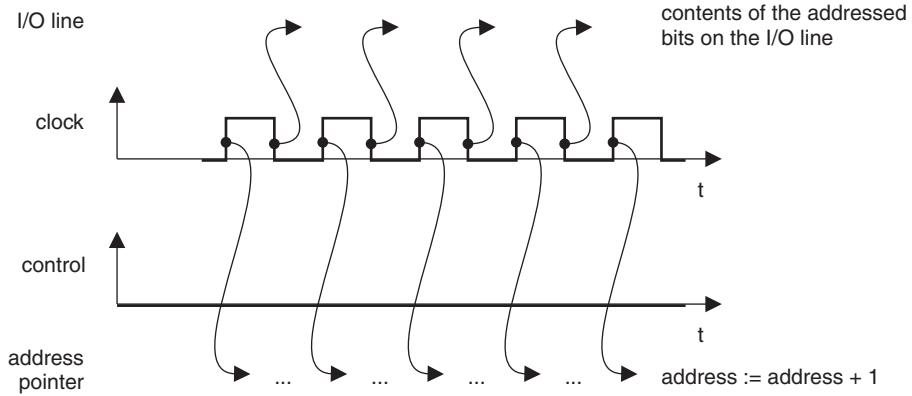
The next function is to read data from the memory. The other two functions are writing and erasing EEPROM bits. Erasing EEPROM bits, which would allow them to be rewritten, is of course blocked in phone cards, as otherwise they could be reloaded.

#### 9.2.1.1 Resetting the address pointer

The power-up logic of the card resets the address pointer to its initial value of zero if the clock line and the control line are both at a high level (see Figure 9.8). However, the pulse on the control line must be somewhat longer than the clock pulse in order to prevent the address from being incremented immediately. The address pointer has to be reset to its original value after each activation sequence, since it would otherwise point to an undefined address.



**Figure 9.8** Resetting the address pointer to zero



**Figure 9.9** Incrementing the address pointer and reading data

### 9.2.1.2 Incrementing the address pointer and reading data

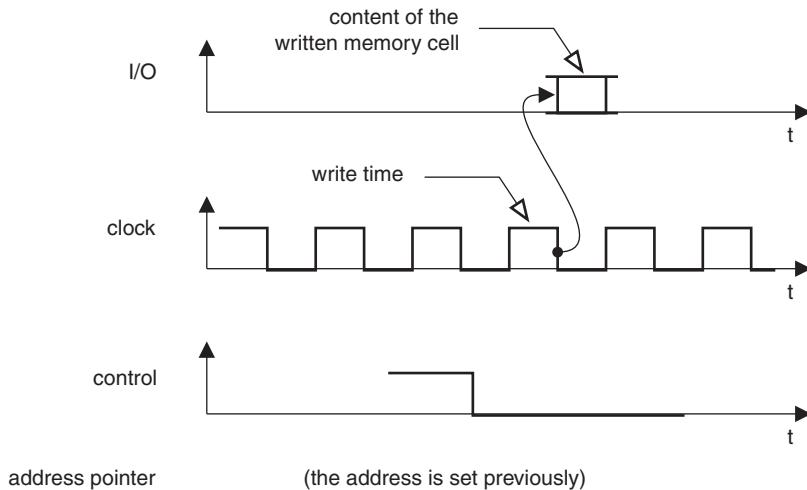
If there is a rising edge on the clock line while the control line is held low, the internal logic of the card increments the address pointer by one (see Figure 9.9). The falling edge of the clock signal causes the content of the address indicated by the pointer to be placed on the data line. If the pointer reaches its maximum value, which depends on the size of the memory, it rolls over to zero and thus starts over from the beginning.

### 9.2.1.3 Writing to an address

If the address pointer is in a writable EEPROM region, the value on the data line can be written to EEPROM by setting the control line high while the clock line is held low (see Figure 9.10). The length of the write cycle is determined by the duration of the immediately following clock pulse. If the bit is written correctly, the content of the written memory cell appears at the data output.

### 9.2.1.4 Erasing bytes

Part of the EEPROM memory of a typical phone card is always organized as a multidigit octal counter. If a byte in this counter has to be erased due to a carry to the next digit, this is secured by



**Figure 9.10** Writing a bit to an EEPROM memory

the logic circuitry. Erasing a byte in memory is thus somewhat complicated. The procedure used here is that when a bit in a byte is written twice in a row, the chip's hardware logic automatically erases the associated less significant byte. This ensures that a carry to the next higher digit occurs and the lower digit is erased without providing any opportunity for fraud. The four types of access described here may vary from chip to chip and from one manufacturer to the next.

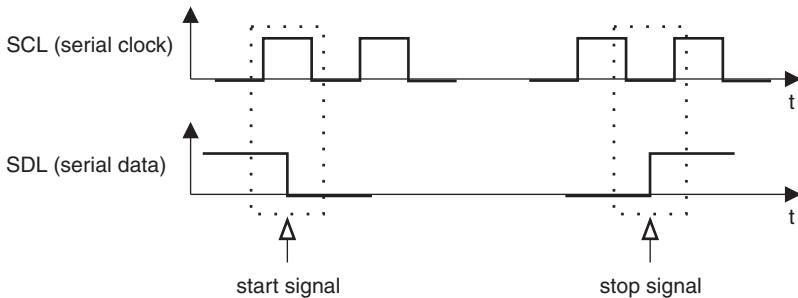
### 9.2.2 I<sup>2</sup>C bus

Another form of data transmission, which by contrast is standardized, is provided by the I<sup>2</sup>C bus. Many recent memory cards use this bus for communication with the terminal. This naturally has the advantage that a variety of chips made by different manufacturers can be used together in a single system. Problems due to different transmission protocols are thus eliminated, since all chips are mutually compatible at the transmission interface.

Synchronous serial data transmission protocols are used relatively frequently because they are uncomplicated and versatile. Components for use with the I<sup>2</sup>C (inter-integrated circuit) bus, which was developed by Philips, have been available since 1990. This bus uses a bidirectional serial data line and a serial clock line. The definition of the I<sup>2</sup>C bus encompasses both the hardware (the two bus lines) and the software in the sense of the data transmission format. Each component on the bus can take control of the bus and send requests to other components connected to the bus.

As memory cards are synchronous devices, the I<sup>2</sup>C bus established a position in the smart card world very quickly. A wide range of memory ICs has become available for use in cards. The following example is based on the SGS-Thomson ST24C04 memory chip, which has a 512-byte EEPROM that can be freely written and read. The EEPROM programming timing is handled internally by the chip, so it does not have to be controlled externally.

The I<sup>2</sup>C bus hardware consists of two lines between the terminal and the card. The SCL (serial clock) line carries the clock signal, which has a maximum frequency of 100 kHz. This



**Figure 9.11** Start and stop signals on the I<sup>2</sup>C bus

yields a data transmission rate of up to 100 kbit/s, which is relatively high for memory cards. The other line, serial data (SDA), is used for bidirectional data exchange between the card and the terminal. The SDA line is connected to the supply voltage ( $V_{CC}$ ) in the terminal by a pull-up resistor. Both communicating parties can only pull this line to ground. Sending a high level is therefore passive and is effected by the sender setting its output to the high-impedance state (tri-state), which allows the pull-up resistor to pull the SDA line to the supply voltage level.

In the smart card context, the terminal is always the I<sup>2</sup>C bus master and the card is always the slave. Data transmission on the I<sup>2</sup>C bus always uses single-byte packets. The most significant bit (bit 8) is sent first. Each transfer over the SDA line is initiated by a start signal and terminated by a stop signal (see Figure 9.11). The start signal consists of a falling edge on the SDA line while the level on the SCL line is high. Conversely, a rising edge on the SDA line while the level on the SCL line is high indicates a stop signal. The receiver must acknowledge receipt of each byte by pulling the SDA line low for one clock cycle.

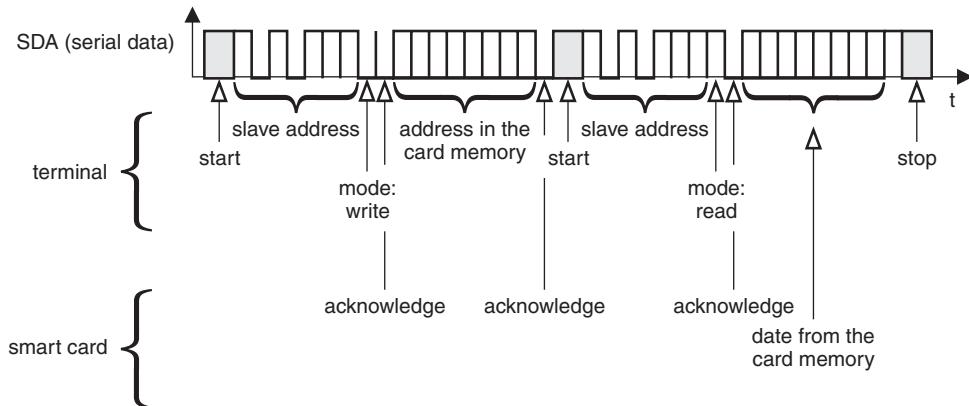
The first seven bits of the first byte after the start of communication are the receiver address. In our example, we assume for simplicity that the address has the binary value 1010 000. Of course, this may vary depending on the chip type, and with some memory ICs it can be set within certain limits. The last bit of the address tells the receiver whether data will be read or written. A one indicates reading, while zero indicates writing.

The following examples illustrate the general operation of the I<sup>2</sup>C bus as used with smart cards.

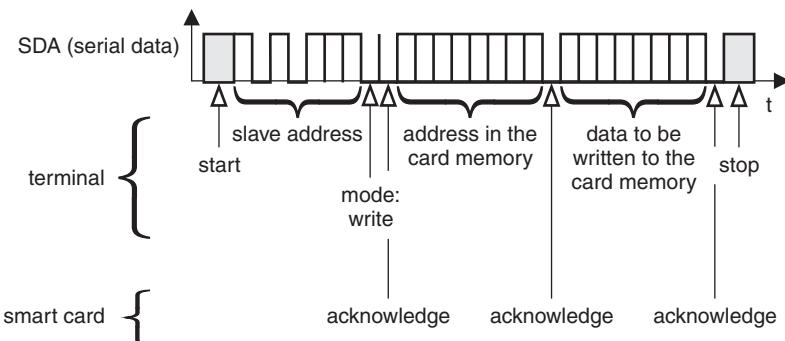
### 9.2.2.1 Reading from an address

There are several types of access for reading the EEPROM in a card. In the type described here, one byte is read at a time. However, it is also possible to read several bytes in succession.

The read sequence (see Figure 9.12) is initiated by the start signal. The subsequent bits contain the address of the card, with the control bit set to ‘write’. This indicates to the card that it must temporarily store the subsequent data in an internal buffer. This buffer is nothing more than a byte-oriented address pointer for the EEPROM. After the card receives the first byte, it sends an acknowledgment by pulling the SDA line to ground for one clock cycle. After this, the terminal sends the EEPROM address to the card. Once again, the card acknowledges receipt of the data. The terminal then sends a start signal and the card address with the read bit set. On receiving this, the card sends the data from the location addressed by the pointer to



**Figure 9.12** Unconstrained reading of a byte from memory using the I<sup>2</sup>C bus



**Figure 9.13** Unconstrained writing of a byte in memory using the I<sup>2</sup>C bus

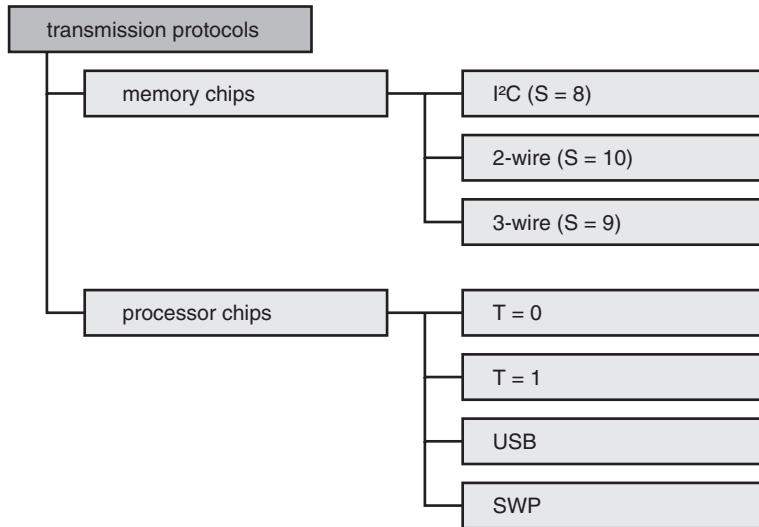
the terminal. The terminal does not have to acknowledge the receipt of the data; it only sends a stop signal to the card. This completes the read sequence for one byte.

#### 9.2.2.2 Writing to an address

As with reading data from the card's EEPROM, there are various modes for writing data. The simplest mode, which can be used to write a single byte anywhere in memory, is described here and illustrated in Figure 9.13.

Here again, the sequence begins with a start signal from the terminal. This is followed by the card's address, with the write bit set. The card acknowledges receipt and then receives from the terminal the address in the EEPROM where the data is to be written.

The card acknowledges this as well, and then receives the data. After the terminal receives the third acknowledgment, which indicates that the card has received the data, it sends a stop signal. Following this, the card starts to write the received data to the EEPROM, which does not require external timing signals. This completes the writing sequence, and the byte is now stored in the EEPROM.



**Figure 9.14** Classification of transmission protocols used with contact smart cards

### 9.3 ISO TRANSMISSION PROTOCOLS

The T = 0 and T = 1 protocols are commonly called the ISO protocols. They are primarily based on the provisions of the ISO/IEC 7816 family of standards, which explains their name.<sup>2</sup>

After a smart card has sent an ATR, possibly followed by a PPS, it waits for the first command from the terminal. The subsequent process always adheres to the master–slave principle, with the terminal as master and the card as slave. In specific terms, the terminal sends a command to the card, which executes the command and then returns a response. This back-and-forth interplay of commands and responses is never interrupted.

There are various ways in which communication with a smart card can be established. There are also a number of methods for re-synchronizing communication if a malfunction occurs. The exact implementation of the commands, the corresponding responses, and the procedures used in the event of transmission errors are defined in the form of transmission protocols.

There are 15 ISO/IEC data transmission protocols that are defined in terms of their basic functionality. They are designated as ‘T = ’ (where ‘T’ stands for ‘transmission protocol’) plus a sequential number, as summarized in Table 9.1 on the facing page.

Two of these protocols predominate in international use. The first is the T = 0 protocol, which became an international standard in 1989 (ISO/IEC 7816-3). The other is T = 1, which was introduced in 1992 in an amendment to an international standard (at the time ISO/IEC 7816-3 Amd. 1, now ISO/IEC 7816-3).

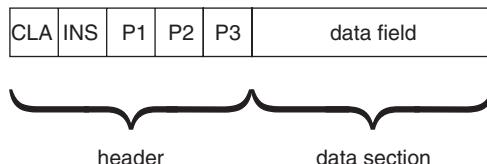
In Germany, a third protocol designated T = 14 was used in the widely distributed card-phone system. It is specified in an Deutsche Telekom company standard.

The data units transported by these transmission protocols are called transmission protocol data units (TPDUs). They can be regarded as protocol-dependent containers that carry data to and from the smart card. The actual application data is embedded in these containers.

<sup>2</sup> Strictly speaking, USB could also be called an ISO protocol because it is also standardized in ISO/IEC 7816

**Table 9.1** Transmission protocols defined in ISO/IEC 7816-3

Transmission protocol	Meaning
T = 0	Asynchronous, half-duplex, byte-oriented, specified in ISO/IEC 7816-3
T = 1	Asynchronous, half-duplex, block-oriented, specified in ISO/IEC 7816-3
T = 2, T = 3	Full duplex; not yet specified
T = 4	Full duplex; not yet specified
T = 5 . . . T = 13	Reserved for future use; not yet specified
T = 14	For national use; not standardized by ISO
T = 15	Reserved for future extensions

**Figure 9.15** Command structure with the T = 0 protocol

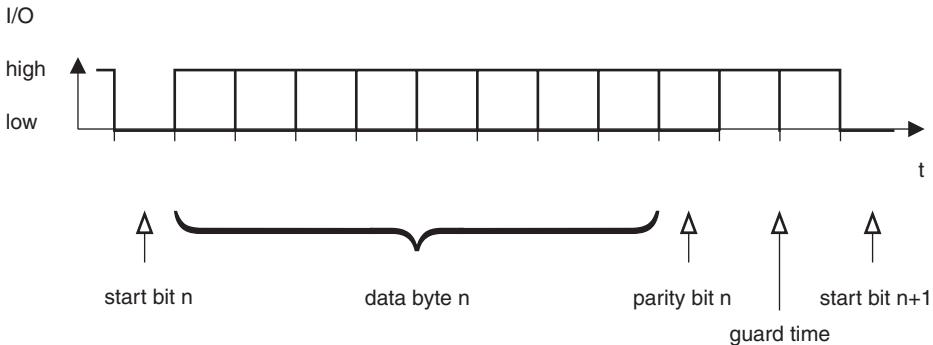
In addition to the technically complex transmission protocols for smart cards, there are a variety of simple synchronous protocols for memory cards. They are typically used with telephone cards, health insurance cards, etc. However, they do not have error-correction mechanisms, and they are based on hard-wired logic in the chip.

### 9.3.1 The T = 0 transmission protocol

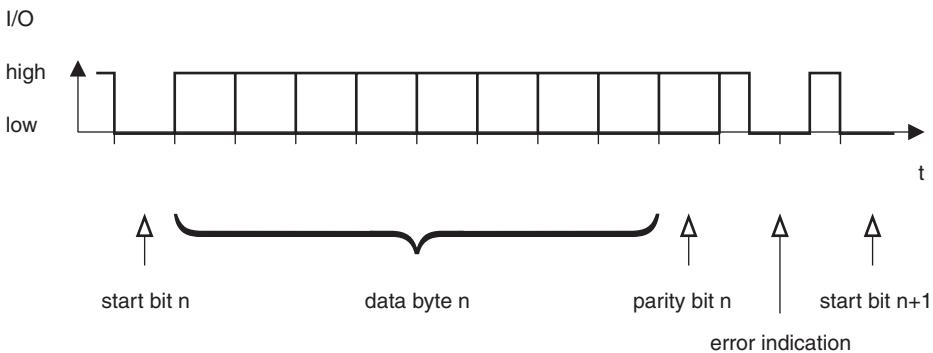
The T = 0 transmission protocol was first used in France in the initial phase of smart card development, and it was the first internationally standardized smart card protocol. It was developed in the early years of smart card technology and was thus designed for minimum memory usage and maximum simplicity. This protocol is used worldwide in GSM cards, which makes it the most widely used of all smart card transmission protocols. The T = 0 protocol is standardized in ISO/IEC 7816-3. Additional compatible specifications are found in the TS 51.011 and TS 102 221 specifications and the EMV standard.

The T = 0 protocol is byte oriented, which means that the smallest unit processed by the protocol is a single byte. As shown in Figure 9.15, the transmission data unit consists of a header containing a class byte, a command byte, and three parameter bytes, which is optionally followed by a data section (body). In contrast to the application protocol data unit (APDU) specified by ISO/IEC 7816-4, length information is provided only by parameter P3. It indicates the length of the command data or the response data, and it is also specified by the ISO/IEC 7816-3 standard.

Due to the byte orientation of the T = 0 protocol, retransmission must be requested immediately if a transmission error is detected in a received byte. With block protocols, by contrast, an entire block (a sequence of bytes) must be retransmitted if an error is detected. Detection of transmission errors with T = 0 is based exclusively on a parity bit appended to each sent byte (see Figure 9.16).



**Figure 9.16** An error-free byte transmitted via the I/O interface with the  $T = 0$  protocol



**Figure 9.17** Signaling a transmission error with the  $T = 0$  protocol by setting the I/O line low during the guard time

If the receiver detects a transmission error, it must set the I/O line to a low level for one etu interval, starting halfway through the first bit interval of the guard time of the faulty byte as depicted in Figure 9.17. This indicates to the other party that the most recently sent byte must be retransmitted. This byte repetition mechanism is very simple and has the advantage that it only incorrect bytes have to be repeated.

Unfortunately, this mechanism suffers from a severe disadvantage. Most interface ICs treat the etu interval as the smallest detectable time unit, so they cannot recognize a low level on the I/O line that is set halfway through a stop bit. Consequently, standard interface ICs are not suitable for use with the  $T = 0$  protocol. However, this is not a problem if each bit is received separately by software.

The  $T = 0$  protocol can also be used to control an external programming voltage for the smart card EEPROM or EPROM (enabling or disabling the programming voltage). This is done by adding 1 to the received command byte and returning it to the terminal as an acknowledge byte. This is why only even-valued command bytes are permitted, since otherwise this mechanism would not work. However, switching an external programming voltage is technically obsolete because all modern smart card microcontrollers generate the programming voltage inside the chip. We thus need not discuss this topic any further.

To illustrate the  $T = 0$  command-response sequence, let us assume that the terminal sends the card a command with a data section and the card responds by sending back data and a

Terminal (IFD)	Smart card (ICC)
5-byte command header [CLA, INS, P1, P2, P3]	→ Header received without error; request transmission of body (acknowledge) ← [ACK]
[body] with P3 = number of data bytes	→ Command processing → Command executed and data available; data length in SW2 ← [SW1    SW2]
GET RESPONSE with P3 = amount of data to be fetched [CLA, INS, P1, P2, P3]	→ ← [data    SW1    SW2]
Command-response sequence completed	

**Sequence Diagram 9.1** A typical T = 0 communication sequence with data in the command and the response (a Case 4 command, such as MUTUAL AUTHENTICATE)

return code. This process may appear complicated at first glance, so it is shown graphically in Sequence Diagram 9.1.

The terminal first sends the card a five-byte command header consisting of a class byte, a command byte, and the P1, P2 and P3 bytes. If the header is received correctly, the card returns an acknowledgment (ACK) in the form of a procedure byte (PB). This acknowledgment is coded the same as the received command byte. After receiving the procedure byte, the terminal sends exactly the number of data bytes indicated by the P3 byte. With this, the card has received the complete command, and it can process it and generate a response.

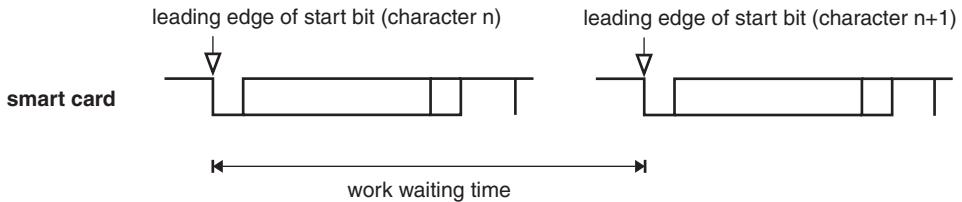
If the response includes data in addition to the two-byte return code, the card informs the terminal of this by means of a special return code with the amount of data indicated by SW2. After receiving this response, the terminal sends the card a GET RESPONSE command, which consists only of a command header that specifies the amount of data to be sent. The card then sends the terminal the requested amount of data generated in response to the first command, along with the appropriate return code. This completes one command sequence.

If a command is sent to the card and the card only generates a return code with no data section, the GET RESPONSE portion of this sequence is omitted. As an additional command from the application layer is needed to fetch data generated for a previous command, strict separation of the protocol layers is lacking. An application layer command (GET RESPONSE) must be used here to support the data link layer, which has certain effects on the application concerned.

The maximum time between the leading edges of two successive bytes is called the work waiting time (see Figure 9.18). It is coded in data element TC<sub>2</sub> of the ATR.

The guard time is primarily intended to separate the individual bytes during data transmission. This gives the sender and the receiver more time to perform the functions of the transmission protocol.

If the smart card returns a procedure byte containing the null value ('60') to the terminal, this does not have any effect on the actual protocol sequence, but it does inform the terminal that the smart card is still processing the last command that was sent. Sending a null value can



**Figure 9.18** Definition of the work waiting time

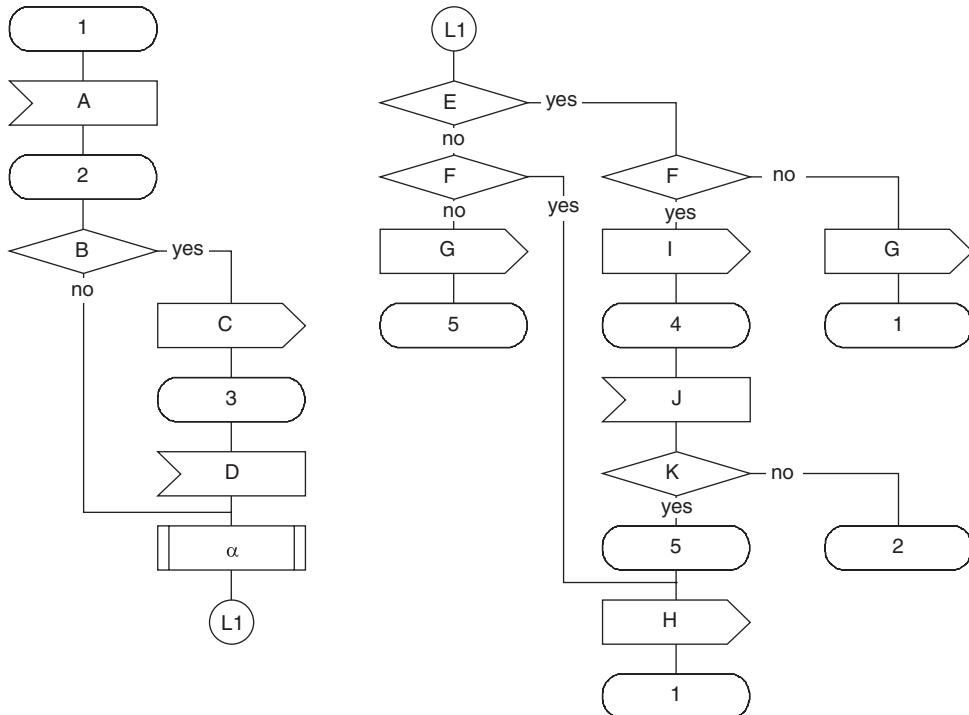
Terminal (IFD)		Smart card (ICC)
5-byte command header [CLA, INS, P1, P2, P3]	→	
[data byte 1]	→	Send one data byte
[data byte 2] <i>further transmission of single bytes</i>	→	Send one data byte
	←	<i>further reception of single bytes</i>
[remaining data bytes n to P3] (P3 = number of available data bytes)	→	Send all remaining data bytes
	←	Command processing [SW1    SW2]
Command-response sequence completed		

**Sequence Diagram 9.2** Single-byte reception with T = 0 and a Case 3 command, such as UPDATE BINARY

thus be used to achieve a sort of waiting time extension (WTX), although it is not standardized in this form. This process is also called ‘more time’ in colloquial language.

The T = 0 protocol allows the card to receive the bytes in the data section individually after it has received the header. To do so, it only has to send the inverted command byte to the terminal as a procedure byte, whereupon the terminal will send a single data byte. The next data byte follows the next procedure byte from the card. Sequence Diagram 9.2 shows the process of byte reception with a case 3 command. This bytewise transmission can continue until the card has received all the bytes in the data section, or until it sends the noninverted command byte to the terminal as a procedure byte. After receiving the noninverted command byte, the terminal sends all the remaining data bytes to the card, which will have then received the complete command. Figure 9.19 shows a state machine for communication using the T = 0 protocol.

According to the GSM standard, GET RESPONSE is requested by setting SW1 to ‘9F’, while according to the ISO/IEC standard the value for this is ‘61’. In each case, SW2 specifies the amount of data to be fetched. There is a difference between TS 51.011 and ISO/IEC 7816-3



**Figure 9.19** A smart card state machine for the  $T = 0$  communication process without error handling. The following states and state transitions are specified: 1: Ground state; 2: Header received with CLA, INS, P1, P2 and P3; 3: Wait for data section ( $P3 = \text{number of bytes}$ ); 4: Wait for a command (header with CLA, INS, P1, P2 and P3);  $P3 = \text{number of response bytes}$ ; 5: SW1 and SW2 sent, GET RESPONSE received; A: Receive header (5 bytes), B: Data section available? ( $P3 \neq 0$ ); C: Data section available, send procedure byte to terminal; D: Receive data section ( $P3 = \text{number of bytes}$ ); E: Did the command have a data section? (i.e. were C and D executed?); F: Is response data available? (no errors occurred?); G: Send SW1 and SW2; H: Send available response data plus SW1 and SW2; I: Send SW1 and SW2 ( $\text{SW2} = \text{amount of response data}$ ), J: Receive a command (header = 5 bytes), K: Is the received command GET RESPONSE?;  $\alpha$ : Command processing

with regard to fetching data with GET RESPONSE. The method described above corresponds to GSM practice and thus represents the largest worldwide application. The ISO/IEC standard specifies that a certain amount of data can be fetched using GET RESPONSE, but it does not define an indexing pointer that can be used to fetch a series of data packets. With the ISO/IEC standard, GET RESPONSE always starts with the first byte. This incompatibility can easily be remedied in the terminal by means of suitable software. The important thing is to be aware of it.

With any transmission protocol, the primary concerns of the user are ultimately the data transmission rate and the error detection and correction mechanisms.

Transmitting an eight-bit byte requires sending twelve bits due to the start bit, the parity bit, and two etu intervals for the guard time. Transmitting one byte thus takes 12 etu, which is equivalent to 1.25 ms with a 3.5712-MHz clock frequency and a divisor of 372. Table 9.2 on the following page lists data transmission times for some typical commands.

**Table 9.2** Data transmission times for some typical commands with the T = 0 protocol and a clock rate of 3.5712 MHz, a divisor of 372, 2 stop bits, and 8 data bytes per command (C = command, R = response)

Command	User data	Protocol data	Time (ms)
READ BINARY	C: 5 bytes R: 2 + 8 bytes	—	18.75
UPDATE BINARY	C: 5 + 8 bytes R: 2 bytes	—	18.75
ENCRYPT	C: 5 + 8 bytes R: 2 + 8 bytes	C: 5 bytes R: 2 bytes	37.50

The data transmission rate naturally drops if transmission errors occur. However, the single-byte repetition mechanism is very advantageous here, since only incorrectly received bytes have to be retransmitted.

The error detection mechanism of the T = 0 protocol consists only of a parity check at the end of each byte. This allows reliable recognition of single-bit errors, but two-bit errors cannot be detected. If a byte is lost during transmission from the terminal to the card, the consequence is an endless loop (deadlock) in the card because it is expecting a specific number of bytes and there is no provision for a timeout. The only practical way for the terminal to escape from this situation is to reset the card and establish communication again from the beginning.

The situation is very similar when the terminal is expecting more data than the smart card sends. This also unavoidably leads to a deadlock. For this reason, some implementations of the T = 0 protocol in terminals have a timer that triggers termination of communication after a configurable maximum interval. The mechanism used for this is similar to the block waiting time (BWT) mechanism with the T = 1 protocol. However, it is not standardized and is thus implementation-dependent.

In normal communication, the lack of full separation between the link and transport layers does not cause any major problems. The smooth operation of the GSM application is the best proof of this. However, problems quickly arise if secure messaging is used. With a partly encrypted header and a fully encrypted data section, it is not possible to support the T = 0 protocol using the previously described procedure without incurring large overheads. This is because an unencrypted command byte must be used as the procedure byte in the T = 0 protocol. This has been recognized by standards organizations and is taken into account in the standards for secure messaging, so all varieties of secure messaging are possible using the T = 0 protocol.

Due to the lack of layer separation and the obvious problems in the event of a bad connection, the T = 0 protocol is often considered to be outdated. However, transmission errors almost never occur in communication between the terminal and the card. The main advantages of the T = 0 protocol are its good average transmission rate, minimal implementation overhead, and widespread use.

### 9.3.2 The T = 1 transmission protocol

The T = 1 transmission protocol is an asynchronous half-duplex protocol for smart cards. It is based on the international ISO/IEC 7816-3 standard. The TS 102 221 specification and

**Table 9.3** Data transmission times for some typical commands with the T = 1 protocol and a clock rate of 3.5712 MHz, a divisor of 372, an XOR error detection code, 2 stop bits, and 8 data bytes per command (C = command, R = response)

Command	User data	Protocol data	Time (ms)
READ BINARY	C: 5 bytes	C: 4 bytes	28.75
	R: 2 + 8 bytes	R: 4 bytes	
UPDATE BINARY	C: 5 + 8 bytes	C: 4 bytes	23.00
	R: 2 bytes	R: 4 bytes	
ENCRYPT	C: 5 + 8 bytes	C: 4 bytes	38.75
	R: 2 + 8 bytes	R: 4 bytes	

the EMV standard are also relevant for this protocol. The T = 1 protocol is a block-oriented protocol, which means that one block is the smallest data unit that can be transmitted between the card and the terminal.

This protocol has strict layer separation and can be assigned to the data link layer (transport layer) in the OSI reference model. In this context, layer separation means that data destined for higher layers, such as the application layer, can be processed completely transparently by the data link layer. It is not necessary for layers other than the ones directly involved to interpret or modify the contents of the transmitted data.

Secure messaging (SM) in particular requires adherence to layer separation. This is essential for transmitting user data across the interface without resorting to complicated procedures or tricks. The T = 1 protocol is currently the only international smart card protocol that permits all varieties of secure data transmission without any compromises.

The transmission protocol process starts after the card has sent the ATR or after a successful PPS has been executed. The first block is sent by the terminal, and the next block is sent by the card. Communication then continues in this manner, with the transmit privilege alternating between the terminal and the card.

Incidentally, the T = 1 protocol is not only used for communication between smart cards and terminals. It is also used by many terminals to exchange application and control data with the computers to which they are connected.

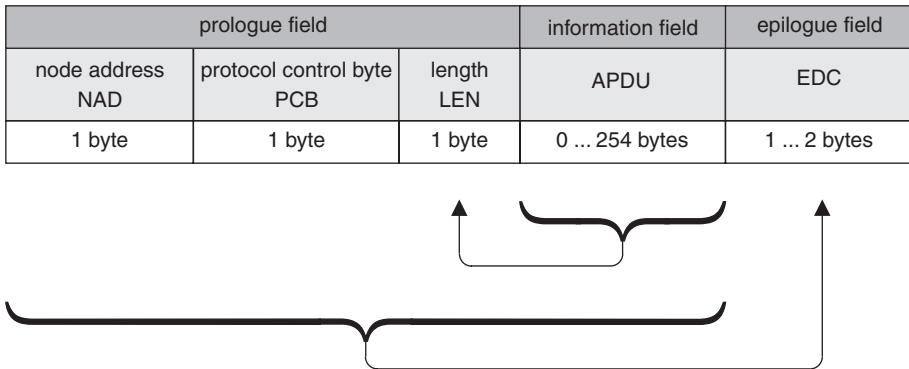
The data transmission rate is naturally of particular interest with any data transmission protocol. Table 9.3 lists transmission times for some typical commands using the T = 1 protocol.

### 9.3.2.1 Block structure

The transmitted blocks are basically used for two different purposes. One of these is the transparent transmission of application-specific data, while the other is transmitting protocol control data or handling transmission errors.

As illustrated in Figure 9.20, a transmission block consists of an initial prologue field, an information field, and a final epilogue field. The prologue and epilogue fields are mandatory and must always be sent. The information field is optional and contains data for the application layer, which is either a command APDU sent to the card or a response APDU from the card.

There are three fundamentally different types of blocks in the T = 1 protocol: information blocks, receipt confirmation blocks, and system blocks. Information blocks (I blocks) are



**Figure 9.20** The structure of a  $T = 1$  transmission block

used to exchange application-layer data transparently. Receipt confirmation blocks (R blocks), which do not have information fields, are used for positive or negative receipt acknowledgement. System blocks (S blocks) are used for control information related to the protocol itself. Depending on the specific control data, they may have an information field.

### Prologue field

The prologue field consists of three subfields: the node address (NAD), the protocol control byte (PCB), and the length (LEN). It is three bytes long and contains basic control and pointer data for the actual transmission block.

#### Node address (NAD)

The first byte in the prologue field is called the node address (NAD) byte, whose coding is described in Table 9.4. It contains the destination and source addresses for the block. Each address is coded using three bits. If an address is not used, its bits are set to 0.

Furthermore, for compatibility with older microcontrollers, control is provided for the EEPROM or EPROM programming voltage. However, there is no practical use for this because all smart card microcontrollers now have on-board charge pumps.

#### PCB field

The subfield following the node address is the protocol control byte (PCB), which is coded as shown in Table 9.5, 9.6 or 9.7 depending on the block type. As the name suggests, it serves to control and supervise the transmission protocol. This increases the complexity of the coding. The PCB field primarily encodes the block type, as well as associated supplementary information.

#### LEN field

The one-byte length field (LEN) indicates the length of the information field in hexadecimal form. Its value can range from '00' to 'FE'. The code 'FF' is reserved for future extensions and currently should not be used.

**Table 9.4** Node address (NAD field)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
X	...	...	...	X	...	...	...	Vpp control
...	X	X	X	...	...	...	...	DAD (destination address))
...	...	...	...	...	X	X	X	SAD (source address)

**Table 9.5** PCB field for an I block

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	...	...	...	...	...	...	...	I block identifier
...	N(S)	...	...	...	...	...	...	Send sequence number
...	...	X	...	...	...	...	...	Sequence data bit M
...	...	...	X	X	X	X	X	Reserved

**Table 9.6** PCB field for an R block

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	0	...	...	...	...	...	...	R block identifier
...	...	0	N(R)	0	0	0	0	No error
...	...	0	N(R)	0	0	0	1	EDC or parity error
...	...	0	N(R)	0	0	1	0	Other error

**Table 9.7** PCB field for an S block

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	1	...	...	...	...	...	...	S block identifier
...	...	0	0	0	0	0	0	Resync request (only from terminal)
...	...	1	0	0	0	0	0	Resync response (only from smart card)
...	...	0	0	0	0	0	1	Request information field size change
...	...	1	0	0	0	0	1	Response to information field size change request
...	...	0	0	0	0	1	0	Abort request
...	...	1	0	0	0	1	0	Abort request
...	...	0	0	0	0	1	1	Request waiting time extension (only from smart card)
...	...	1	0	0	0	1	1	Response to waiting time extension request (only from terminal)
...	...	1	0	0	1	0	0	Vpp error response (only from smart card)

### Information field

The information field in an I block serves as a container for application-layer data (OSI layer 7). The content of this field is transmitted completely transparently. This means that the content is conveyed directly by the transmission protocol without any analysis or evaluation.

The information field in an S block transfers data for the transmission protocol. This is the only case in which the content of this field is used by the transport layer.

According to the ISO standard, the size of the information field can be ‘00’ to ‘FE’ (254) bytes. The value ‘FF’ (255) is reserved by ISO for future use. The terminal and card may have I fields with different sizes. The default size of the terminal I field(IFSD = information field size for interface device) is 32 bytes; this can be modified via a special S field. The same default value applies to the card (IFSC = information field size for the card), but it can be modified by a parameter in the ATR.<sup>3</sup> In practice, the size of the I field for both the terminal and the card ranges from 50 to 254 bytes.

### Epilogue field

The epilogue field at the end of the block contains an error detection code computed from all previous bytes in the block. This code is computed using either an LRC (longitudinal redundancy check) or a CRC (cyclic redundancy check) method. The method used must be specified in the interface characters of the ATR. If it is not specified, by convention the LRC method is used implicitly. Otherwise the CRC computation is performed according to ISO 3309. The divider polynomial used,  $G(x) = x^{16} + x^{12} + x^5 + 1$ , is the same as for CCITT Recommendation V.41. Both of these error detection codes can only be used for error detection; they cannot be used to correct a block error.

The single-byte longitudinal redundancy checksum is computed using XOR concatenation of all previous bytes in the block. This computation can be executed very quickly, and its implementation is not code-intensive. It is usually performed on the fly during data transmission or reception. It is a standard part of practically all T = 1 implementations.

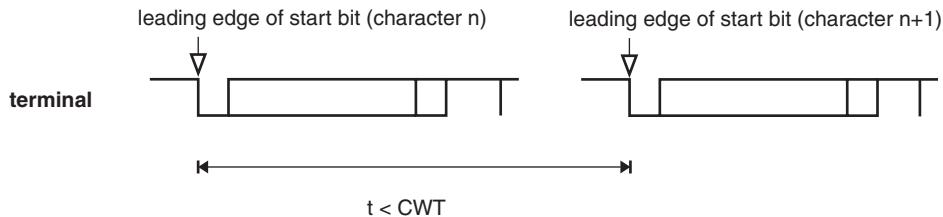
The CRC method for generating error detection codes yields a much greater probability of error detection than the relatively primitive XOR checksum method. However, the CRC method is presently not used in practice because the XOR checksum has become the established standard throughout the world. With the CRC method, the epilogue field must be extended to two bytes, which further reduces the effective data transmission rate.

#### 9.3.2.2 Send/receive sequence counter

Each information block in the T = 1 protocol has a send sequence number consisting of only one bit located in the PCB byte. This number is incremented modulo 2, which means it alternates between 0 and 1. The send sequence counter is also designated N(S). Its starting value at protocol initiation is 0. The counters in the terminal and the smart card are incremented independently of each other.

The main purpose of the send sequence counter is to support requests for resending blocks received with errors, since individual data blocks can be unambiguously addressed by N(S).

<sup>3</sup> ATR data element TA<sub>i</sub> (i>2)



**Figure 9.21** Definition of the character waiting time (CWT)

### 9.3.2.3 Waiting times

Several waiting times are defined to provide senders and receivers with precisely specified minimum and maximum intervals for various activities during data transmission. They also provide defined ways to terminate communication in order to prevent deadlocks in case of faults. Default values are defined for all of these waiting times in the standard, but they may be modified to maximize the transmission rate. The modified values are indicated in the specific interface characters of the ATR.

#### Character waiting time (CWT)

As shown in Figure 9.21, the character waiting time is defined as the maximum interval between the leading edges of two consecutive characters in a block. The receiver starts a countdown timer on each leading edge, using the character waiting time as the initial value. If the timer expires before the leading edge of a new bit is detected, the receiver assumes that the transmission block has been received in full.

Consequently, the CWT could generally be used for deadlock detection. This would considerably reduce the data transmission rate because each block time would be increased by the length of the CWT. It is thus better to detect the end of the block by counting received bytes.

The CWT is calculated using the CWI data element in the ATR using the formula

$$\text{CWT} = (2^{\text{CWI}} + 11)\text{work etu}$$

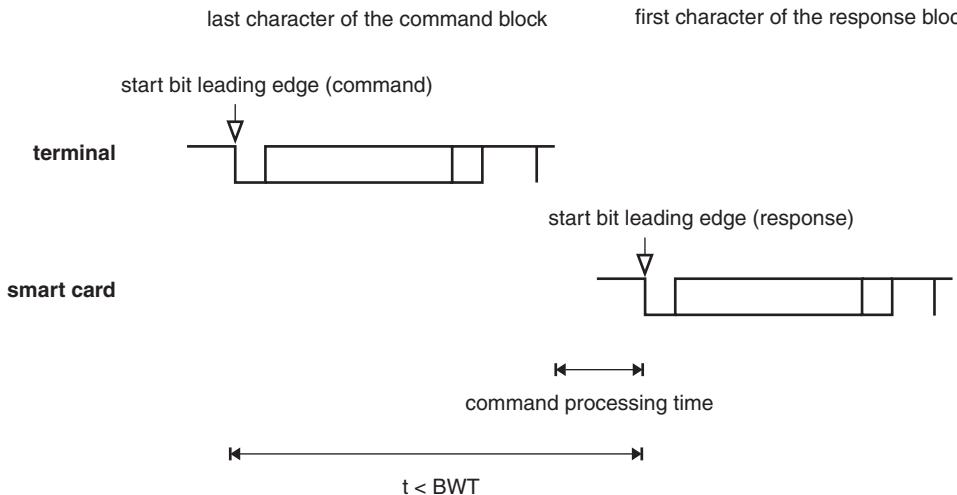
The default value of CWI is 13, which yields the following value for CWT:

$$\text{CWT} = (2^{13} + 11)\text{work etu} = 8\,203 \text{ work etu}$$

With a clock frequency of 3.5712 MHz and a divisor of 372, this yields an interval of 0.85 s.

This interval, which is specified in the standard as the default setting, is too long for fast data communication. In practice, CWI usually has a value of 3 to 5. This means that with a normal transmission sequence in which the characters follow each other without any time delay, the receiver waits for one to two byte intervals before detecting the end of a block or loss of communication.

The receiving routine usually detects the end of a block from the block length information in the LEN field. However, if the content of this field is incorrect, the character waiting time can be used as an additional way to terminate communication at the receiver end. This problem only occurs when the specified length is too long, as in this case the receiver will wait for additional characters that never arrive. This causes the transmission protocol to hang, and this



**Figure 9.22** Definition of the block waiting time (BWT)

condition can only be cleared by a card reset. The character waiting time mechanism gets around this problem.

### Block waiting time (BWT)

The purpose of the block waiting time is to allow communication to be terminated in a defined manner if the smart card does not respond. The block waiting time is the maximum allowed interval between the leading edge of the last byte of a block sent to the card and the leading edge of the first byte returned by the card, as illustrated in Figure 9.22.

In terms of a conventional  $T = 1$  block, this is the maximum time between the leading edge of the XOR byte in the epilogue field of the command block and the leading edge of the NAD byte in the response from the card.

If this time expires before the card sends a response, the terminal may assume that the card has a malfunction and initiate suitable a response mechanism. This could, for example, be a card reset, followed by a new attempt to establish communication.

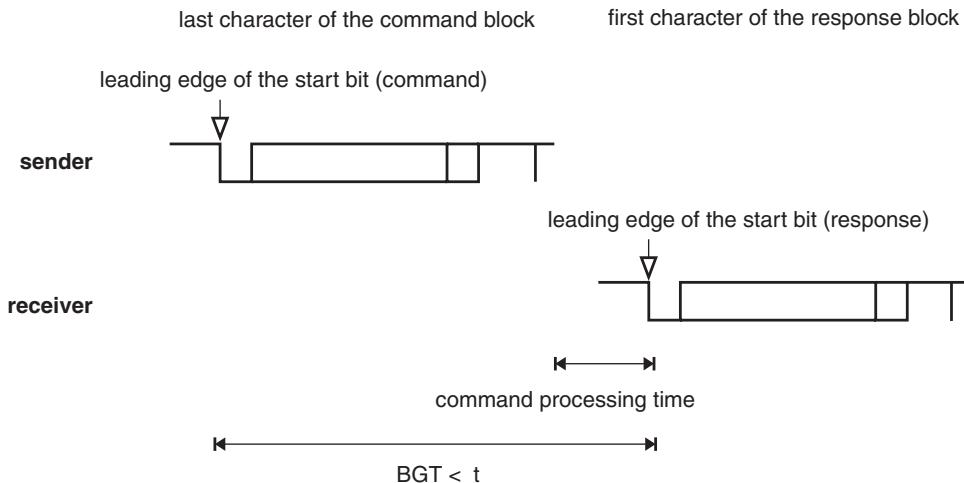
The BWT is specified in abbreviated form in the interface characters of the ATR by the BWI parameter. The value of the BWT is given by the formula

$$\text{BWT} = 2^{\text{BWI}} \times 960 \times \frac{372}{f} \text{ s} + 11 \text{ work etu}$$

If no BWI value is specified in the ATR, the default value of 4 is used. With a divisor of 372 at 3.5712 MHz, this yields a block waiting time of 1.6 s:

$$\text{BWT} = 2^4 \times 960 \times \frac{372}{3\,571\,200 \text{ Hz}} \text{ s} + 11 \text{ work etu} = 2^4 \times 0.1 \text{ s} + 11 \text{ work etu} \approx 1.6 \text{ s}$$

This value is clearly rather generous. In practice, a value of 3 is often used for BWI, which yields a block waiting time of 0.8 s. Typical command processing times in the card



**Figure 9.23** Definition of the block guard time (BGT)

are usually around 0.2 s.<sup>4</sup> A BWT of 0.8 s thus represents a compromise between normal command processing times and quick detection of a smart card that is no longer responding to commands.

### Block guard time (BGT)

As shown in Figure 9.23, the block guard time is defined as the minimum interval between the leading edge of the final byte sent in one direction and the leading edge of the first byte sent in the opposite direction. It is the opposite of the BWT, which is defined as the maximum time between these two leading edges. Another difference is that the block guard time is obligatory for both parties and must be observed, while the block waiting time is only significant for the smart card. The purpose of the block guard time is to provide the sender with a minimum time interval in which to switch over from transmitting to receiving.

The block guard time has a standard fixed value of 22 etu. In a smart card with a clock rate of 3.5712 MHz and a divisor of 372, this yields an interval of approximately 2.3 ms.

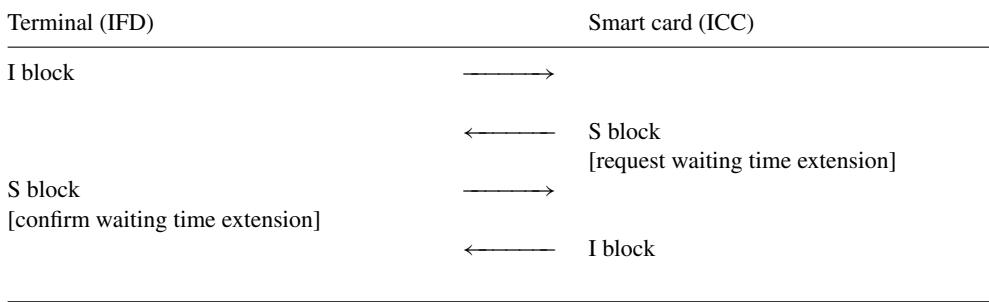
#### 9.3.2.4 Transmission protocol mechanisms

##### Waiting time extension

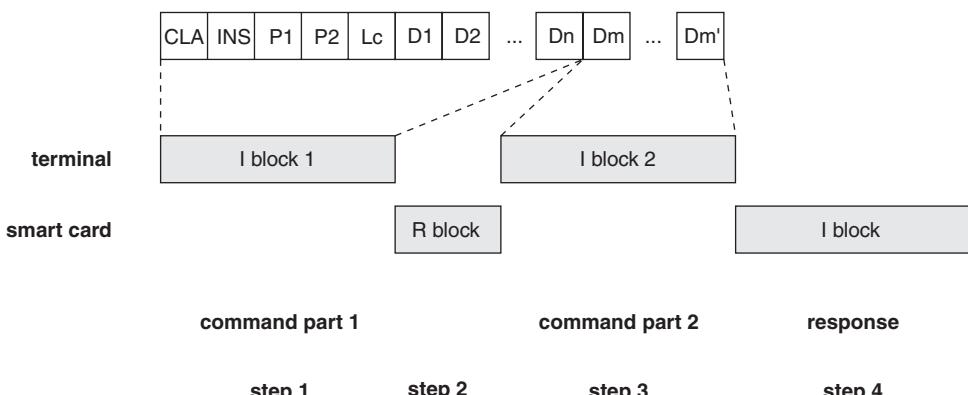
If the smart card needs more time to generate a response than the maximum time allowed by the block waiting time (BWT), it can request a waiting time extension from the terminal. As shown in Sequence Diagram 9.3, it does so by sending a specific S block with a request for an extension, and it receives a corresponding S block from the terminal in acknowledgment. The terminal is not allowed to refuse this request.

A byte in the information field informs the terminal of the duration of the extension. This byte, multiplied by the block waiting time, gives a new block waiting time. This extension only applies to the most recently sent I block.

<sup>4</sup> See also Section 11.17.1, ‘Processing time estimation’, on page 407



**Sequence Diagram 9.3** Procedure for extending the waiting time



**Figure 9.24** Example of block chaining for transmitting data from the terminal to the smart card

### Block chaining

One of the essential performance features of the T = 1 protocol is the block chaining function. This allows either party to send data blocks that are larger than its transmit or receive buffer size. This is particularly useful in light of the limited memory capacities of smart cards. Chaining is only allowed for information blocks, since only these blocks can contain large amounts of data. In the chaining process, the application data is partitioned into individual blocks that are sent to the receiver one after the other.

The application layer data must be partitioned such that none of the resulting segments is larger than the maximum block size of the receiver. The first segment is then placed in an information field in accordance with the T = 1 protocol, prologue and epilogue fields are added, and the block is sent to the receiver. The M bit ('more data' bit) is set in the block's PCB field to indicate to the receiver that the block chaining function is being used and chained data is contained in the following blocks.

After the receiver has successfully received this information block with the first segment of the user data, it indicates that it is ready to receive the next chained I block by returning an R block whose sequence count N(R) is the same as the send sequence count N(S) of the next I block. The next block is then sent to the receiver.

This exchange of I and R blocks continues as illustrated in Figure 9.24 until the sender issues an I block with the M bit set to 0 in the PCB field, which indicates that this block is the

**Table 9.8** Error handling stages with T = 1

Synchronization stage	Mechanism
Stage 1	Repeat the erroneous block
Stage 2	Resynchronize and then repeat the erroneous block
Stage 3	Reset the smart card and re-establish the connection

last one in the chain. After receiving this block, the receiver has all the application layer data and can process the full data block.

There is a restriction with regard to the block chaining mechanism. Within a single command-response sequence, chaining may be performed in one direction only. For example, if the terminal is sending chained blocks, the card may not send chained blocks in response.

There is another restriction that has nothing to do with the mechanism itself, but instead arises from the very limited amount of memory in smart cards. Implementing the block chaining mechanism involves extra overhead, and its usefulness is very limited because commands and responses are not especially long and thus do not normally require chaining.

If the card's receive buffer in RAM is not large enough to hold all of the data passed using block chaining, a buffer must be created in EEPROM. This causes a sharp reduction in the transmission rate because the EEPROM (unlike the RAM) cannot be written at the full speed of the processor.

Consequently, many T = 1 implementations do not support block chaining because it is often not justified by the cost/benefit ratio. This is a typical example of the fact that standards are often interpreted very liberally in practice. In this case, the interpretation amounts to regarding block chaining as a nonessential option of T = 1. This can lead to significant compatibility problems in practice.

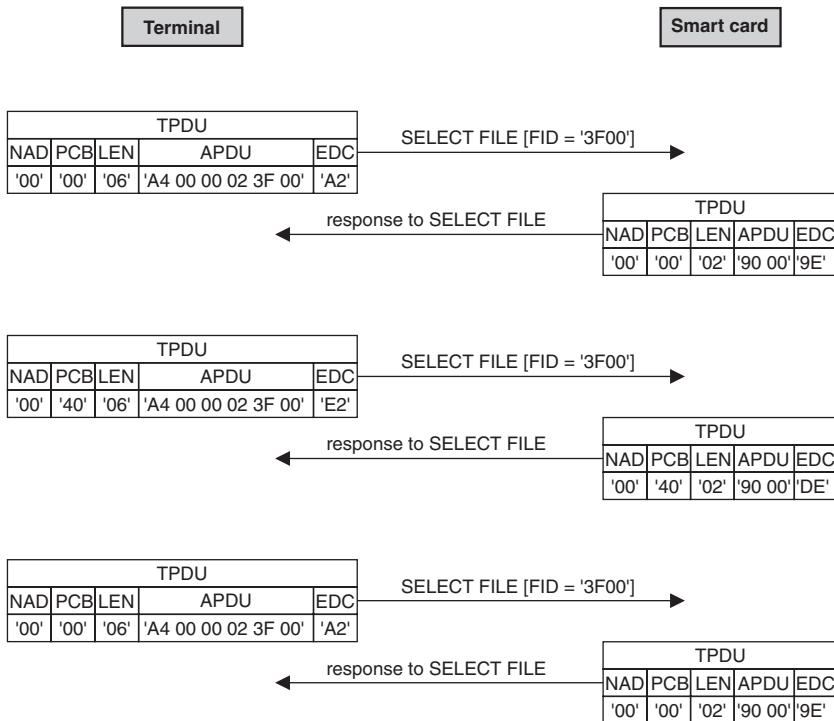
### Error handling

The T = 1 protocol has sophisticated error detection and handling mechanisms. If an invalid block is received, the protocol has precisely defined procedures for attempting to restore error-free communication.

There are three synchronization stages as seen from perspective of the terminal, as listed in Table 9.8. In the first stage, the sender of a faulty block receives an R block that signals an EDC or parity bit error or a general error. In response, the receiver of this R block (the original sender) must retransmit the last block that it sent.

If error-free connection cannot be restored using this mechanism, the next stage is invoked. This means that the smart card receives a resynchronization (resync) request from the terminal in an S block. The terminal expects a resync response in reply. In parallel with this, the terminal and the card reset their send and receive counters to zero, which corresponds to the protocol state immediately after the ATR. Starting from this initial state, the terminal attempts to re-establish communication.

The first two stages affect only the protocol layer. They have no effect on the actual application. However, the third synchronization stage affects all layers in the smart card. If the terminal cannot establish an error-free connection in the first two synchronization stages, it triggers a smart card reset via the Reset lead. This unfortunately causes all data and states



**Figure 9.25** Successful transmission of a TPDU using the  $T = 1$  transmission protocol. The XOR option is used for the error detection code (EDC). A SELECT command with an FID of '3F00', which selects the MF, is sent in the APDU. The incrementing of the send sequence count in the PCB byte for each transaction and the corresponding changes in the EDC are shown clearly here

of the current session to be lost. After the reset, communication must be re-established from the beginning.

If even this procedure fails to produce a working connection after three attempts, the terminal deactivates the card. The user then usually receives an error message to the effect that the card is defective.

### 9.3.2.5 Example of data transmission with the $T = 1$ protocol

As an example of data transmission using the  $T = 1$  protocol, Figure 9.25 shows the process for sending a SELECT command.

## 9.3.3 Comparison of the $T = 0$ and $T = 1$ transmission protocols

Two international transmission protocols have been described in the previous sections. The essential features of these protocols, along with their advantages and disadvantages, are summarized in Table 9.9 on the next page.

**Table 9.9** Comparison of the internationally standardized asynchronous transmission protocols

Criterion	T = 0	T = 1
Data transmission	asynchronous, half-duplex, byte-oriented	asynchronous, half-duplex, block-oriented
Standard	ISO/IEC 7816-3, TS 51.011, EMV	ISO/IEC 7816-3, EMV
Divisor	freely definable, usually 372	freely definable, usually 372
Block chaining	not possible	possible
Error detection	parity bits	parity bits, EDC at end of block
Memory required for implementation	≈ 300 bytes	≈ 1 100 bytes

A brief remark regarding achievable transmission rates is in order in connection with any comparison of these two protocols. Attempts are often made to compare the T = 0 and T = 1 protocols by calculating effective data transmission rates. However, such calculations are only valid for specific commands in specific contexts. If they are generalized, they become invalid and meaningless.

Both protocols have their strengths and weaknesses with regard to achievable transmission rates. These are strongly dependent on very many individual factors, such as the transmission error rate, the size of the I/O buffer in the card, and the specific implementation of the protocol. In short, it can be assumed that on average and in most applications, the effective transmission rates of both protocols will be nearly the same. If you want to increase the transmission rate, changing the protocol will have little effect. It is more effective to reduce the divisor value, since this yields significantly better results.

### 9.3.4 The T = 14 transmission protocol (Germany)

The ISO/IEC 7816-3 standard includes an ATR code for designating a national transmission protocol. The designation reserved for national protocols is T = 14. With the introduction of the C-Netz for mobile telephony and card phones in Germany, a protocol was needed for communicating with the smart cards used in these systems. The byte-oriented T = 0 protocol was considered undesirable, and at that time there was not yet a standardized block-oriented protocol. Consequently, in 1987 Deutsche Telekom decided to use a protocol developed by a DIN working group. This protocol was given the designation T = 14, which simply meant that it was a country-specific implementation. It had no significance outside of Germany, but it had enormous influence on the development of the internationally standardized T = 1 protocol because it formed the primary basis for the T = 1 protocol.

Due to its use in the C-Netz mobile telecommunication system and public card phones, the T = 14 was employed on a very large scale in Germany. With the discontinuation the C-Netz at the end of 2000 and the conversion of public card phones to the T = 1 protocol, the T = 14 protocol is no longer significant in Germany.

The T = 14 protocol has a block-oriented structure and works asynchronously to the clock signal. The divisor (clock rate conversion factor) has a value of 512, which yields a transmission rate of 9600 bit/s at a clock frequency of 4.9512 MHz. Data transmission in layer 2 (the data

link layer) always takes place using the direct convention. The buffer size for transmission blocks must be at least 50 bytes, with a maximum value of 255 bytes. There is no block chaining mechanism.

## 9.4 USB TRANSMISSION PROTOCOL

In the late 1990s, it became apparent that the T = 0 and T = 1 transmission protocols for smart cards were not future-proof in terms of their maximum data transmission rates. At that time, the Universal Serial Bus (USB) was on the eve of conquering the PC world, where it eventually supplanted interfaces such as RS232, the parallel interface and the PS/2 interface, which had long dominated the PC area. Although there were some efforts in the smart card industry to upgrade the maximum data transfer rates of the two traditional protocols, they failed to gain a following in the smart card market, and USB was added to the ISO/IEC 7816 family of standards as Part 12. With the decision by ETSI in mid-2007 to define the USB interface as an option for smart cards in the telecommunication sector, the USB protocol definitively joined the ranks of transmission protocols for smart cards.

The USB interface was originally developed in 1996 by Intel, with the objective of unifying the diverse PC interfaces. This idea had already been put into practice in the 1980s by Apple in the form of the Apple Desktop Bus (ADB), although at that time it had a data rate of only 10 kbit/s. By contrast, the first USB specification provided for a data rate of 12 Mbit/s. After the specification process was opened up to other companies in the customary manner of the IT industry, the relevant specifications were rapidly upgraded and extended. Version 1.1 appeared in 1998, and the currently applicable version of the USB specification was published in 2000. Among other things, it provides for a maximum data rate of 480 Mbit/s.

The basic USB specification is ‘Universal Serial Bus Specification Version 2.0’ (2000), which concentrates primarily on the PC environment. This document is published by the USB Implementers Forum [USB], and the extensive specification can be downloaded from their website. Three additional documents from the same source are of importance for USB in smart cards: ‘Device Class: Smart Card CCID’, ‘Device Class: Smart Card ICCD’, and ‘Inter-Chip USB’. These specification describe the special features of USB in the smart card environment. In particular, ‘Inter-Chip USB’ specifies lower current consumption than the regular USB specification, as well as additional voltage classes. Although the ISO/IEC 7816-12 standard and the ETSI TS 102 600 specification are the principal references in their individual areas, the above-mentioned documents from the USB Implementers Forum are the primary literature.

In order to support USB communication with a smart card microcontroller, the microcontroller needs a special hardware component. This component implements the electrical part of the communication protocol, in the same way as a UART does for serial communication with T = 0 or T = 1. As a rule, semiconductor manufacturers provide library routines to support USB communication using the hardware-level elements of their microcontrollers. In this case, integration with the smart card operating system is performed by the operating system producer.

For compatibility reasons, the previous protocols (T = 0 and T = 1) are usually still supported by the smart card operating systems concerned, with USB available as an additional communication option. It is often possible to use T = 0 or T = 1 at the same time as USB, which means that the smart cards can communicate using both transmission channels in parallel. Of course, this must be supported by the chip hardware.

The first USB smart cards supported only the Low-speed option (1.5 Mbit/s) defined in Version 1.1 of the USB specification. Due to the small speed advantage relative to T = 0 or T = 1 and other reasons, they were not widely used. Modern smart card microcontrollers typically work with the USB Full-speed option, which provides a data rate of 12 Mbit/s. The USB High-speed option, which operates at 480 Mbit/s, will not penetrate the smart card sector in the foreseeable future due to the complexity of the required circuitry.

As the timing requirements for USB have rather tight tolerances, it is not always possible to fulfill them using single-chip microcontrollers without external components. The main difficulty is meeting the timing requirements over the full temperature range. If these requirements cannot be met, a quartz crystal connected to the processor chip must be included in the module. Although this increases the cost of the smart cards, in some cases it is necessary in order to achieve the frequency stability needed for USB communication.

Compared with the original transmission protocols for smart cards, the USB interface is rather complex. The USB 2.0 specification alone fills 650 pages. Here we wish to provide a selective overview of USB with the emphasis on smart cards. With regard to the technical implementation details and coding, we refer the reader to the above-mentioned standards and specifications in order to keep the size of this book within bounds.

Implementation of USB in smart card microcontrollers naturally requires suitable hardware support. Semiconductor manufacturers usually provide accompanying library routines for this purpose, which can be linked into user implementations or used as examples. With this approach, the volume of program code for USB is approximately 10 KB, with a RAM requirement of around 100 bytes.

#### 9.4.1 Electrical connection

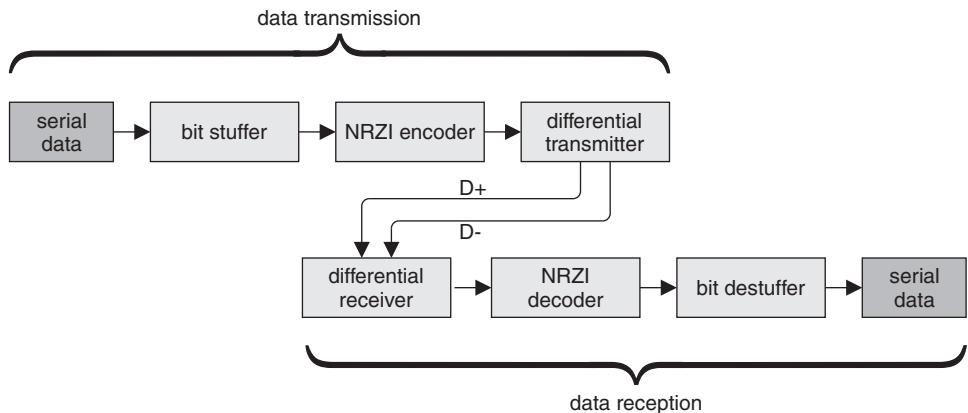
Despite what the name Universal Serial Bus suggests, USB is not a physical data bus to which devices can be connected in parallel. With USB, the designation ‘bus’ refers to the logical topology instead of the electrical connections. The electrical connections are always point-to-point with USB.

Data transmission is bit serial with USB, which means one bit after the other. Transmission at the electrical level is differential using a pair of signal lines (D– and D+). This means that the levels on the two lines are always opposite, and the receiver recognizes the received data bits from the difference between the potentials on the two lines. The advantage of this transmission method is that it provides significantly better transmission reliability compared with single-ended (nondifferential) signals.

The previously unused contacts C4 (AUX1) and C8 (AUX2) of the eight-contact module are used for the electrical connections of the USB interface. Figure 9.26 shows the signal

C1		C5	Vcc		GND
C2		C6	NC		NC
C3		C7	NC		NC
C4		C8	D+		D-

Figure 9.26 Module contact assignments with the USB interface (NC = not connected)



**Figure 9.27** Data coding and decoding at the bottom transmission level

assignments of the two contacts. In principle, it is also possible to power a smart card from the USB bus, and this actually occurs with pure USB smart cards (USB ICC). This is done by connecting contacts C1 (Vcc) and C8 (GND) to the four-lead USB bus. The voltage levels on the two data lines ( $D+$  and  $D-$ ) are 3.3 and 0 V, and they are actively driven by the current sender.

To increase transmission reliability and enable simple clock recovery, nonreturn to zero inverted (NRZI) coding is used for data transmission. With this method, a 0 bit in the serial data stream causes a polarity reversal, while a 1 bit leaves the polarity unchanged. As a result, a series of consecutive 0 bits produces a corresponding series of polarity reversals on the data lines. The receiver recovers the bus clock from these polarity reversals, which makes a separate clock line unnecessary. With this technique, it is essential to avoid extended periods without polarity reversals due to long series of 1 bits, as this could prevent the receiver from maintaining synchronization. This is avoided by a bit stuffer ahead of the NRZI coding stage, which inserts a 0 in the data stream if a series of six consecutive 1 bits occurs. This causes a polarity reversal in the subsequent NRZI coding process. In the receiver, an NRZI decoder followed by a bit-destuffer restore the original data stream. Figure 9.27 shows the operating principle of this process in the bottom USB data transmission layer.

#### 9.4.2 Logical connection

USB provides logical data channels called pipes, which are based on the physical transmission layer and terminate in individual endpoints. From a hardware perspective, an endpoint (EP) is a send or receive buffer that can be addressed uniquely by a four-bit endpoint number and the transfer direction. Endpoint 0 (EP0) is called the control endpoint and is the only bidirectional endpoint, which means it can both send and receive data. The other endpoints are only unidirectional, which means they can either send or receive data, but not both. The transfer direction is always defined relative to the host, which is always the terminal in a smart card system. This means that in case of data transfer from a terminal to a smart card, an Out endpoint in the terminal always has a counterpart In endpoint in the card, and these two endpoints are logically connected by a pipe.

**Table 9.10** USB transfer modes

Transfer type	Error correction	Properties	Typical use in a PC environment	Use with smart cards
Control	yes	USB device control and configuration	USB device configuration	available
Interrupt	yes	Small data volumes; guaranteed latency	Keyboard and mouse data	not available
Bulk	yes	Large data volumes	Printer data	available
Isochronous	no	Large, time-critical data volumes	Real-time audio and video data	not available

The USB communication process is controlled by the host, which means that pure master-slave operation prevails. Consequently, devices connected to the host respond only after they have received requests from the host.

#### 9.4.2.1 Transfer modes

USB provides four different transfer modes as listed in Table 9.10, each optimized for a specific application scenario. As its name suggests, the control transfer mode is used to control and configure connected USB devices. Control transfers are always initiated by the EP0 endpoint and are always acknowledged in both directions. This way both the sender and the receiver can always be sure that the transmitted data actually arrived.

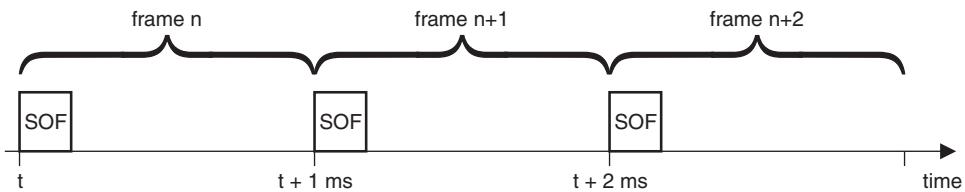
Interrupt transfer mode is used to transfer small amounts of data, such as typically originate from a keyboard or mouse, with low latency. Error correction can be applied to interrupt transfers if necessary. The term ‘interrupt’ in connection with USB is misleading, since the USB bus does not operate under interrupt control, but instead operates with polling by the master at fixed intervals. Bulk transfer mode is provided for transferring large amounts of data that is not time-critical. The data is corrected if a transmission error occurs, which makes this transfer mode suitable for sending data to a printer.

Isochronous transfer mode can be used to transfer relative large amounts of time-critical data, such as streaming audio or video data. However, data packets with transmission errors are not repeated in this mode. The higher-level layer only receives a message indicating that a data packet was lost. The priority with this transfer mode is real-time behavior, not freedom from errors.

Only small amounts of data have to be transferred in case of devices that use the Low-speed USB option. Consequently, only control transfer mode and interrupt transfer mode are supported for devices of this sort.

#### 9.4.2.2 Data packets

The least significant bit is sent first with USB data transmission. The remaining bits follow in increasing order of significance. In case of data items composed of several bytes, the least significant byte is sent first, followed by the other bytes in ascending order (little-endian).



**Figure 9.28** USB frames sent at 1 ms intervals. The start of each frame is marked by an SOF (start of frame) token

Data is transferred in frames at fixed time intervals as illustrated in Figure 9.28. In Low-speed and Full-speed operation, the interval is 1 ms with a tolerance of only  $\pm 0.0005\text{ ms}$ . Each frame begins with a start of frame (SOF) token containing a packet identifier (PID), a frame number, and a CRC.

USB communication is performed entirely using data packets. Each packet is protected by a CRC<sup>5</sup> to enable detection of transmission errors. The start and end of each packet are identified by special codes called SOP (start of packet) and EOP (end of packet). There are four packet types: token, special, data, and handshake. A token packet specifies the source address, target address, and transfer direction. Special packets are used for a variety of purposes, including split transactions between a host, a hub, and an end device that operate at different transfer speeds. Data packets are used to transfer user data, while handshake packets are used for transfer security and flow control.

The USB bus can be put in sleep mode to save energy. This is called ‘suspend’ and is initiated by suspending the transmission of SOF tokens. The ‘resume’ signal wakes a device in the suspend state and restores it to active operation. This is done by placing an inverted idle level on the D– and D+ lines. The resume signal can be generated by the host or by an end device.

When a USB device is connected to a host, the device is first identified and configured by the host. This is called enumeration. For this purpose, the host accesses the device (such as a smart card) using the default address 0, and the device responds by sending its device descriptor to the host. The host then assigns a different address to the device.

#### 9.4.3 Device classes

USB devices with the same features and properties are grouped into device classes. The purpose of this is to enable the host to use a generic driver for each device class, so that there is no need to provide a separate driver for each device. The device classes are described in corresponding additions to the basic USB specification. The term ‘composite device’ is also important in this connection. A composite device is a USB end device that informs the host during enumeration that it supports more than one device class.

The CCID (integrated circuit(s) cards interface device) and ICCD (integrated circuit(s) cards devices) classes provide all the resources necessary for USB communication between a terminal and a USB smart card with T = 0 and/or T = 0, APDUs, and TPDUs (including

<sup>5</sup> See section 6.5.2, ‘CRC checksums’, on page 126

extended APDUs). If TCP/IP communication is desired instead, the RNDIS (remote network device interface specification), CDC (communication device class), or EEM (Ethernet emulation model) device class is necessary. If a smart card also needs to be accessible from a computer via USB as a mass storage device, it must support the USB-MSC (USB mass storage class) device class.

#### 9.4.4 Summary and prospects

Two main application areas for USB smart cards can presently be foreseen. The first is pure USB smart cards that can be connected directly to the USB port of a computer, as illustrated in Figure 3.47 on page 59. This eliminates the need for a smart card terminal and thus perceptibly increases user acceptance.

The second foreseeable application area is in the mobile telephone sector. The data rate of the T = 0 protocol, which is used exclusively in this sector, has for some time been far from adequate for relatively large volumes of data, and it cannot be increased by further enhancements. This imposes distinct limits on the use of SIM and USIM cards as mass storage devices or HTTP servers, which could certainly be overcome by using the USB protocol for smart cards standardized by ETSI. The necessary hardware support in smart card microcontrollers is now available, although suitable extensions to mobile telephone sets would be necessary for USB. When these will actually become commercially available depends primarily on the business strategies of handset manufacturers and network operators.

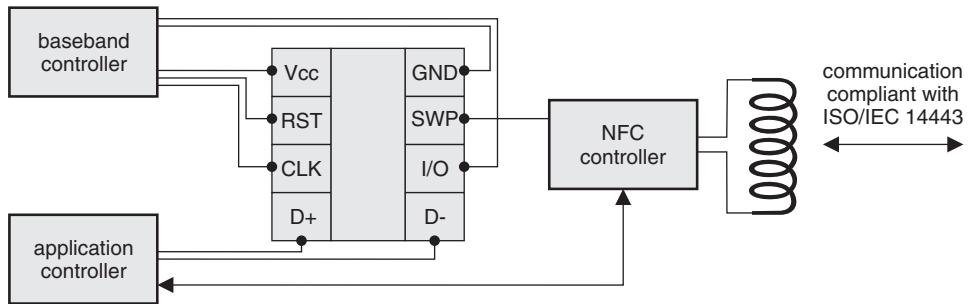
### 9.5 MMC TRANSMISSION PROTOCOL

The MultiMediaCard (MMC) interface is an inexpensive interface for use with memory cards containing NAND flash memory. It is designed for transferring data between a device and a memory card. The MultiMediaCard protocol specification (MultiMediaCard System Spec) is published by the MultiMediaCard Association [MMCA].

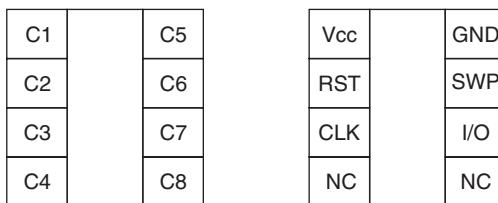
For compatibility with MMC memory cards used in mobile telephones, the MMC protocol is sometimes used for fast communication with smart cards. Now that the USB interface has been standardized as a smart card interface it can be assumed that the MMC protocol will not win any more ground in the smart card world. However, a few smart card microcontrollers<sup>6</sup> have hardware suitable for using the MMC protocol for communication with a terminal. This capability is in fact used in some regions for certain mobile telephone applications.

The MMC protocol requires at least three electrical connections: a clock line, a command line, and a 1-bit, 4-bit or 8-bit data path. This means that data can be transferred in parallel with a suitable configuration. The maximum possible transmission rate with this arrangement is 416 Mbit/s. Although the MMC protocol was originally designed for reading and writing data in large flash memories, this interface can also be used with smart cards to exchange commands in the usual APDU format.

<sup>6</sup> See section 5.4.3, ‘Communication with MMC’, on page 95



**Figure 9.29** The usual scheme for integrating a smart card in a typical mobile telephone with the  $T = 0$ , USB and SWP transmission protocols. The lines with arrowheads show communication paths and do not represent electrical connections



**Figure 9.30** Module contact assignments with the SWP interface (NC = not connected)

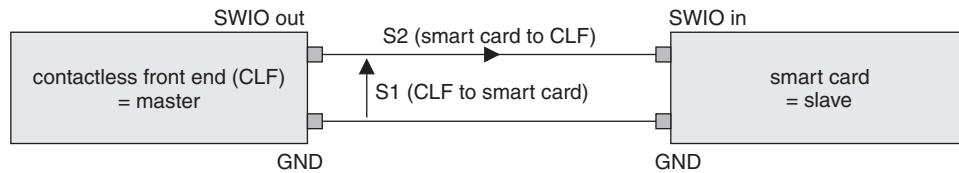
## 9.6 SINGLE-WIRE PROTOCOL (SWP)

Integration of NFC functionality in a mobile telephone along with a SIM or USIM requires direct communication between the NFC controller and the SIM or USIM, which can simultaneously exchange data with the mobile telephone via another channel ( $T = 0$  or USB). This led to the development of a new protocol called the Single-Wire protocol (SWP), which became established in the smart card world in 2007. This protocol was originally developed by Gemplus, and it managed to prevail over an alternative proposed by Philips (now NXP) called S<sup>2</sup>C (SigIn-SigOut Connection). The bus uses two lines (SigIn and SigOut), which allow a full-duplex connection to be implemented between two devices.

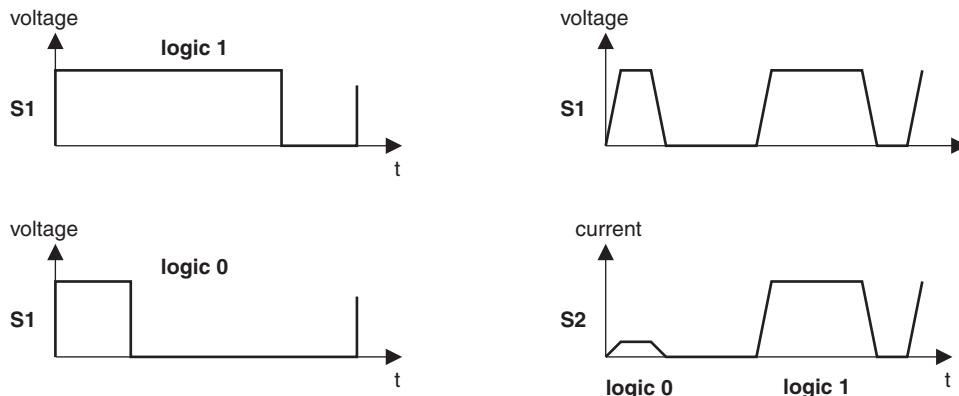
By contrast, the Single-Wire protocol needs only one connecting line, which has the major advantage that a module with eight contacts can provide enough connectivity for three protocols:  $T = 0$ , USB, and SWP (see Figure 9.29). The C6 contact, which was originally used for the external programming voltage for nonvolatile memory (EPROM or EEPROM) in the card, but became redundant a good while ago, is used for SWP. Figure 9.30 shows the scheme used with a standard eight-contact module for the SWP protocol in parallel with other protocols.

The Single-Wire protocol must be supported by a dedicated peripheral unit in the smart card microcontroller because it has a relatively complicated transmission method at the electrical interface level.<sup>7</sup> The protocol supports full-duplex operation, which means that data can be sent

<sup>7</sup> See also Section 5.4.4, ‘Communication with SWP’, on page 95



**Figure 9.31** Data communication between an NFC controller (contactless front end or CLF) and a UICC card with the Single-Wire protocol. Communication via the single wire in the CLF direction (S1) uses voltage sensing, while communication in the smart card direction (S2) uses current sensing

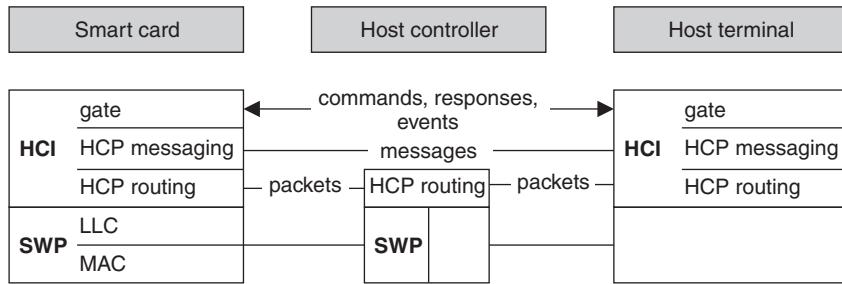


**Figure 9.32** Electrical (left) and logical (right) communication levels on the S1 interface. The S1 interface is depicted at the top right, while the two allowed current levels and corresponding logic levels on the S2 interface are shown below

and received concurrently. To establish a full-duplex connection between two communication parties with the Single-Wire protocol, the smart card uses current modulation to send its data in digital form via the single connecting wire and receives data via the same wire using voltage sensing.

The Single-Wire protocol is specified in ETSI specification TS 102 613, which defines an entity called the contactless front end (CLF) – typically the NFC component in a mobile telephone – that independently handles communication via the ISO/IEC 14443 protocol. Here ‘independently’ also means that it can operate without power from the mobile telephone if necessary. In the Single-Wire protocol, the CLF is always the master at the electrical level, which means that the UICC is the slave, as illustrated in Figure 9.31. The allowed voltage classes are B ( $3\text{ V} \pm 10\%$ ) and C ( $1.8\text{ V} \pm 10\%$ ). Class A ( $5\text{ V}$ ) is not supported because it is no longer used in the telecommunication sector.

Digital data is transferred in full-duplex mode using a combination of voltage and current sensing as depicted in Figure 9.32. If the CLF sends a high state to the UICC, the UICC can either draw current or not draw current. If the UICC draws current, the CLF sees a logic 1; otherwise it sees a logic 0. The Reset contact of the smart card module does not have any effect on smart card operation at the Single-Wire protocol level because protocol resets are triggered via the higher-level protocol layers.



**Figure 9.33** SWP protocol stack and associated HCI layer

Like all modern transmission protocols, SWP has a layered architecture. The task of the data link layer is to transport link protocol data units (LPDUs). As shown in Figure 9.33, this layer can be broken down into two sublayers: medium access control (MAC) and logical link control (LLC). The MAC layer is responsible for exchanging individual message frames, while the LLC layer is responsible for error management and flow control. For transmission error detection, the individual frames are protected by a 16-bit CRC. A net data transmission rate of 320 kbit/s to 1.6 Mbit/s in each direction can be achieved with this configuration, depending on the length of the bit interval.

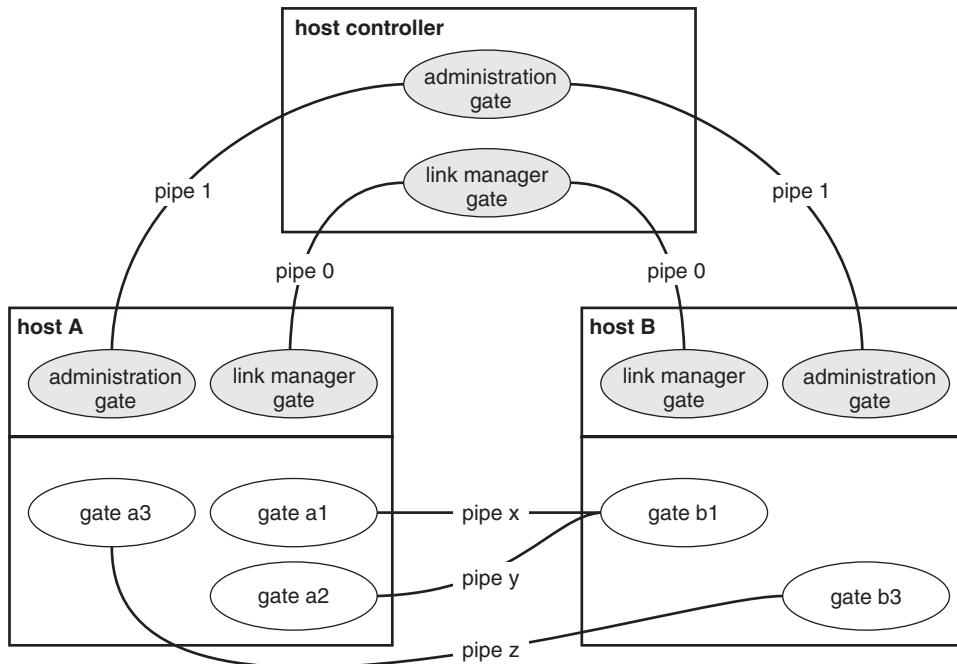
The host controller interface (HCI) layer is located above the data link layer of the Single-Wire interface. This layer defines the interface between the individual logical components. The HCI layer in turn consists of three sublayers. As seen from the SWP level, they are the HCP routing mechanism, the HCP messaging mechanism, and a set of gates. The gates are the entry points to the services of the applications built on top of this layer.

The HCI is specified in TS 102 622, which also specifies several important aspects of the overall interface, as illustrated in Figure 9.33. The highest level in the hierarchy consists of the hosts, which are logical entities that perform one or more services. A terminal or UICC is an example of a host. Hosts can be identified uniquely by a one-byte value.

In this scheme, the hosts contain gates, which are entry points for individual services. The HCI protocol enables message exchange between the gates of different hosts. Here a distinction is made between management gates and generic gates. Management gates are intended to be used for managing the connections, while generic gates are used for the application. Gates can also be identified uniquely by a one-byte value.

Each pair of gates is connected by a pipe to enable message exchange. There are two different types of pipes: static pipes and dynamic pipes. Static pipes are always available, which means they do not have to be created and they cannot be deleted, in contrast to dynamic pipes, which can be created and deleted. A pipe can have one of two states: open or closed. These states persist even if the host is powered down or up. Pipes can be identified uniquely by a one-byte value.

Figure 9.34 on the next page shows the logical relationships between hosts, gates and pipes with an HCI. The host controller uses two management gates for administration and link management. These gates are connected by static pipes to the corresponding gates in hosts A and B. This arrangement is used to establish and manage application-specific communication between the two hosts. At the application level, the two hosts communicate via gates a1, a2 and a3 and the associated pipes. The protocol allows a single gate to have several pipes connected to different gates.



**Figure 9.34** Logical relationships between the host controller, two hosts, management gates, generic gates, and several static and dynamic pipes with an HCI. The management gates are shaded grey, while the generic gates are white. The static pipes are numbered 0 and 1; all other pipes are dynamic. In a typical mobile telephone, the host controller and host B logical units are individual elements of the NFC controller. In this situation, host A would be the USIM

The SWP and the HCI built on top of it can be used to implement a wide variety of applications and services between SIM and USIM cards, NFC controllers, and mobile telephone units. The SWP protocol with its higher-level HCI layer has a clearly defined logical level, which means it corresponds to the current technical state of the art. This puts the SWP with the HCI in a good position to fulfill the market demands in the mobile telecommunication sector, which are often very dynamic.

# 10

## Contactless Data Transmission

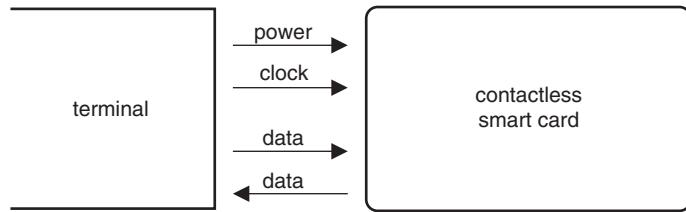
Contactless cards do not require any electrical connection between the smart card and the card terminal in order to transmit power and data over a short distance. The technical advantages of contactless technology and examples of new applications that are enabled by this technology are described in Section 2.3.2, ‘Contactless memory cards’, on page 20. This chapter describes the technology and operating principles of contactless smart cards.

The methods used for transmitting power and data with contactless cards are not new. They have been generally known for many years in the realm of radio-frequency identification (RFID) systems, which are used in a wide variety of applications, such as animal implants and transponders for electronic anti-start systems in vehicles.

A large variety of methods are used in radio systems, and especially in radar systems, to identify persons or objects at short or long distances. Only a few of these methods are suitable for use with smart cards in ID-1 format, which are the subject of this chapter, since all of the functional components must be housed in a flexible card with a thickness of only 0.76 mm. One of the difficult problems here is integrating flexible batteries in the bodies of mass-produced cards. Although flexible batteries with suitable thickness are now available, experience with using them in field trials or mass production is not yet available. Consequently, contactless cards are limited to passive techniques in which the operating power of the card must be extracted from the electromagnetic field of a card terminal, which limits the useful range to around one meter.

To facilitate understanding of the various methods, they can be classified according to diverse parameters. One approach is to classify them according to the method used to transmit power and data. The most commonly used methods are radio waves or microwaves, optical transmission, capacitive coupling, and inductive coupling. Capacitive and inductive coupling are best suited to the flat shape of a smart card without an internal power source. Presently available commercial systems utilize these methods exclusively, and they are the only ones considered in the relevant families of ISO/IEC standards (10536, 14443, and 15693). Consequently, in this book we limit ourselves to describing these methods.

Just as with contact smart cards, a system with contactless cards consists of at least two components: a card and a compatible terminal. The terminal can act as a reader or a reader/writer, according to the technology used. As a rule, the terminal includes an additional interface for communication with a background system.



**Figure 10.1** Necessary power and data transmission between a terminal and a contactless smart card. This link utilizes antennas in the card and the terminal

The following four functions are necessary to allow a contactless card to communicate with a terminal (see Figure 10.1):

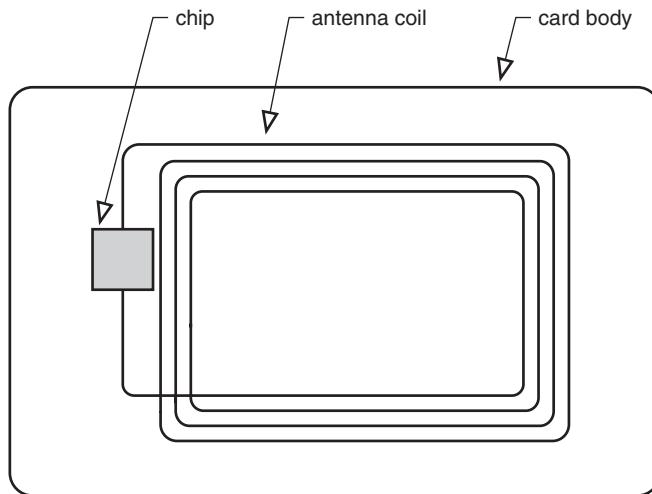
- power transmission to the card for powering the integrated circuit
- clock signal transmission
- data transmission to the card
- data transmission from the card

Many different concepts based on experience with RFID systems have been developed to satisfy these requirements. Most of them are specifically designed for particular applications. For instance, there is a considerable difference between systems where the distance between the card and the terminal is only a few millimeters in normal use and systems where the distance can be around one meter. Naturally, compatibility problems are inevitable when many different solutions designed and optimized for specific applications are developed. The committee responsible for generating the relevant standards (ISO/IEC JTC1/SC17/WG8) has attempted to restrict the number of technical variants in order to enable interoperability at reasonable effort and cost.

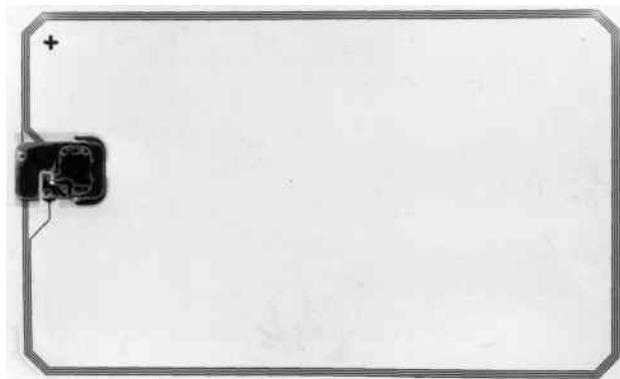
## 10.1 INDUCTIVE COUPLING

Inductive coupling is presently the most widely used technique for contactless smart cards. It can be used to transmit both power and data. All methods described in the ISO/IEC standards use inductive coupling. Differing requirements with regard to factors such as range, along with general constraints such as radio frequency licensing regulations, have resulted in a variety of implementations.

With some applications, such as access control, it is only necessary to be able to read the data stored in the cards, which means that technically simple solutions can be used. Due to their low power consumption (a few tens of microwatts), the working range of such cards extends to around one meter. Their memory capacity is usually only several hundred bits. If it is also necessary to write data, the power consumption rises to more than  $100 \mu\text{W}$ . As a consequence, the range is reduced to around 10 cm in write mode, since licensing restrictions prevent increasing the emitted power of the writing equipment to any desired level. The power consumption of contactless microprocessor cards is even greater and is typically 10 mW. The usable range relative to the terminal is thus even more restricted.



**Figure 10.2** Basic structure of a contactless smart card with inductive coupling

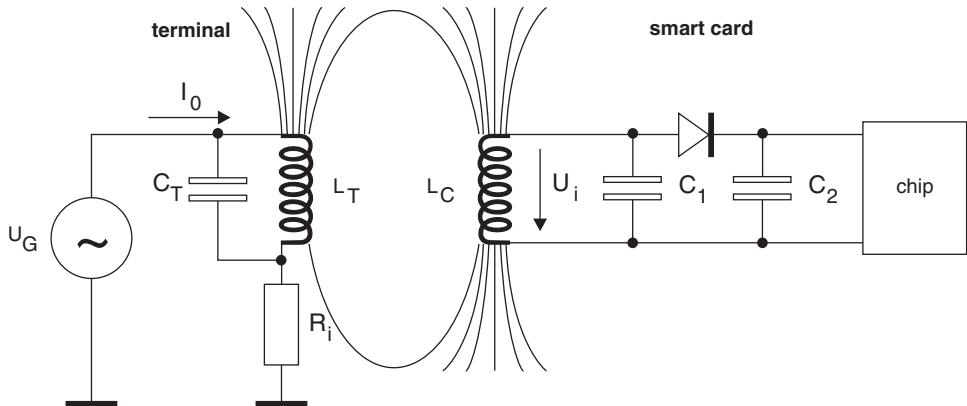


**Figure 10.3** An inlay foil for a contactless smart card with inductive coupling using an etched coil

All cards that employ inductive coupling work on the same principle, regardless of their range or power consumption. One or more coils (usually with large enclosed areas) are incorporated in the card body to act as coupling components for power and data transmission, along with one or more chips (see Figures 10.2 and 10.3). A variety of methods are commonly used to produce the coil and connect the coil to the chip. They are described in Section 14.4.5, ‘Card bodies with integrated antennas’, on page 589.

## 10.2 POWER TRANSMISSION

For the above-mentioned reasons, contactless smart cards are almost always operated passively. This means that all of the power necessary for the operation of the chip in the card must be transmitted from the terminal to the card.



**Figure 10.4** Using inductive coupling to supply power to a smart card. As the distance between the card and the terminal is small compared with the wavelength of the electromagnetic field, the loosely coupled transformer model can be used to describe this arrangement

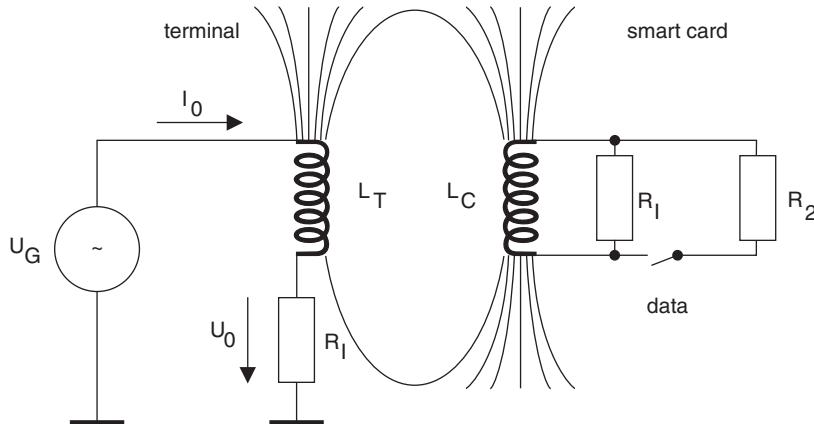
The is done using the principle of a loosely coupled transformer, as illustrated in Figure 10.4. The transformer coil belonging to the terminal is used to generate a strong high-frequency electromagnetic field. The frequencies most commonly used for this purpose are <135 kHz and 13.56 MHz, which correspond to wavelengths of 2400 and 22 m. This means that the wavelength of the electromagnetic field is several times larger than the distance between the card and the terminal, or in other words, that the card is located in the near-field region of the terminal. This allows the loosely coupled transformer model to be used.

If a contactless card is brought into the vicinity of the terminal, part of the magnetic field passes through the coil of the card, which causes an AC voltage  $U_i$  to be generated in the coil. This voltage is rectified and used to power the chip. The coupling between the terminal coil and the card coil is very weak, so the efficiency of this arrangement is very low. Consequently, a very strong current must flow in the terminal coil in order to generate the necessary field strength. This is achieved by connecting a capacitor  $C_T$  in parallel with the coil  $L_T$ , with the capacitance chosen such that the resonant frequency of the circuit formed by the coil and the capacitor matches the frequency of the signal used to transmit power to the card.

Coil  $L_C$  and capacitor  $C_1$  in the card also form a resonant circuit with the same resonant frequency. The voltage induced in the card is proportional to the signal frequency, the number of turns in coil  $L_C$ , and the area enclosed by the coil. This means that the number of turns needed for the coil drops with increasing signal frequency. At 125 kHz, it is 100 to 1000 turns, while at 13.56 MHz it is only 3 to 10.

### 10.3 DATA TRANSMISSION

All known digital modulation methods can be used for transmitting data from the terminal to the card. The most commonly used methods are amplitude shift keying (ASK), frequency shift keying (FSK), and phase shift keying (PSK). ASK and PSK are most often used, since they are especially easy to demodulate.



**Figure 10.5** Example circuit illustrating the principle of load modulation, which is used in contactless smart cards for data transmission. The smart card can modulate voltage  $U_0$  in the terminal by switching resistor  $R_2$  in and out of the circuit

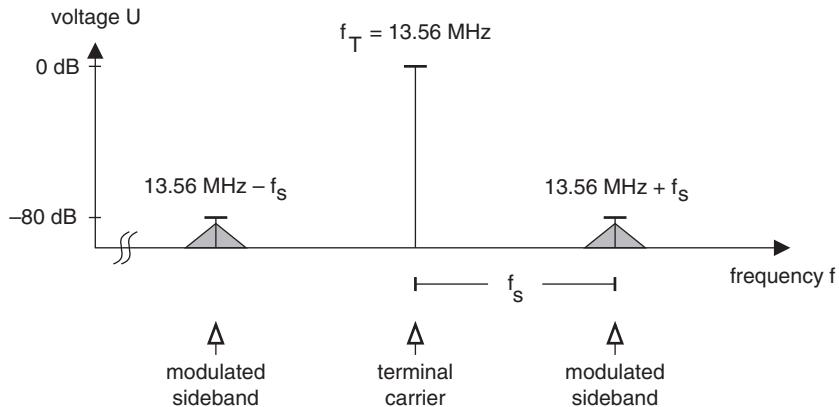
In the other direction, from the smart card to the terminal, a type of amplitude modulation called load modulation is used. It is generated by using the data signal to digitally switch a load in the card. If a smart card tuned to the resonant frequency of the terminal is brought into the near field of the terminal, it draws power from this field as previously described. This causes the current  $I_0$  in the coupling coil of the terminal to increase, which can be detected as an increased voltage drop across an internal resistor  $R_i$ . The smart card can thus vary (modulate) the voltage  $U_0$  in the terminal by varying the load on its coil, for example by switching load resistor  $R_2$  in and out of the circuit as shown in Figure 10.5. If the switching of resistor  $R_2$  is controlled by the data signal, the data can be detected and processed in the terminal.

Due to the weak coupling between the coils of the terminal and the card, the voltage variations induced in the terminal by load modulation are very small. In practice, the amplitude of the modulation signal is only a few millivolts. This can only be detected using sophisticated circuitry, since it is overshadowed by the significantly stronger transmit signal of the terminal (around 80 dB). However, if a subcarrier with frequency  $f_s$  is used, the received data signal appears in the terminal as two sidebands at frequencies  $f_c \pm f_s$ , as illustrated in Figure 10.6. The sidebands can be separated from the significantly stronger terminal signal by passing them through a bandpass filter and then amplified. Demodulation is straightforward after this.

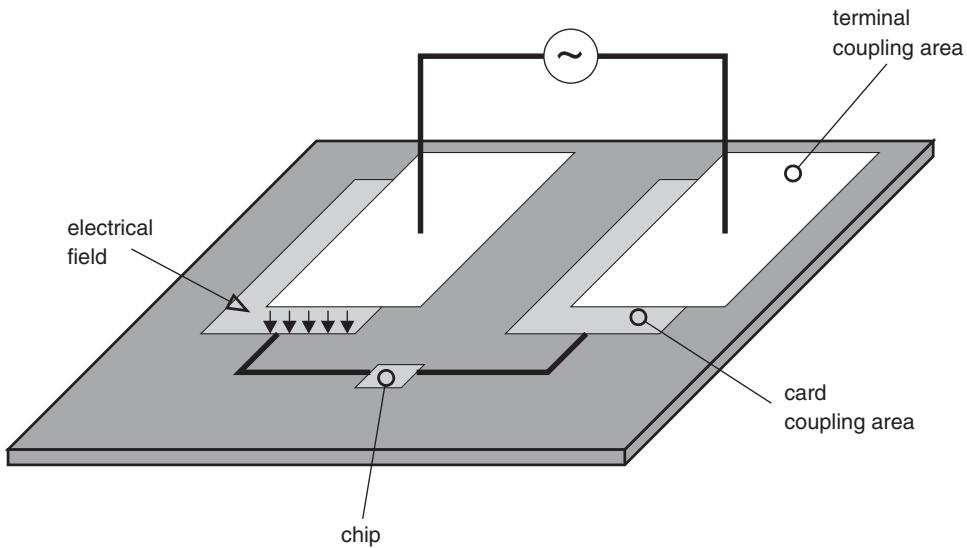
A drawback of modulation using a subcarrier is that it requires significantly more bandwidth than direct modulation. Consequently, this technique can only be used in a few frequency bands that provide adequate bandwidth.

## 10.4 CAPACITIVE COUPLING

If the distance between the card and the terminal is small, capacitive coupling can be used for data transmission as shown in Figure 10.7 on the following page. With this type of coupling, conductive surfaces are incorporated in the card body and the terminal to serve as the plates of a capacitor when the card is inserted in the terminal or placed on the terminal. The capacitance that can be obtained essentially depends on the sizes of the coupling surfaces



**Figure 10.6** Load modulation using a subcarrier produces two sidebands separated from the terminal carrier frequency by the value of the subcarrier frequency  $f_s$ . The information is contained in the sidebands of the two subcarrier sidebands, which result from the modulation of the subcarrier (figure based on Klaus Finkenzeller [Finkenzeller 06])



**Figure 10.7** Operating principle of capacitive coupling. The coupling arises from the alternating electrical field between two parallel, electrically conductive surfaces in the card and the terminal

and their separation. The maximum size is limited by the dimensions of the card, while the minimum separation is determined by the insulation required between the coupling surfaces. A usable capacitance of several tens of picofarads can be obtained at an acceptable level of cost and effort. This is insufficient for transmitting enough power to operate a microprocessor. Consequently, this method is used only for data transmission, with the operating power being transmitted inductively. This mixed method is standardized in ISO/IEC 10536 for close-coupling cards, and it is described in detail in Section 10.7, ‘Close-Coupling Cards’.

(ISO/IEC 10536)', on page 291. As the name suggests, this method is limited to short coupling distances.

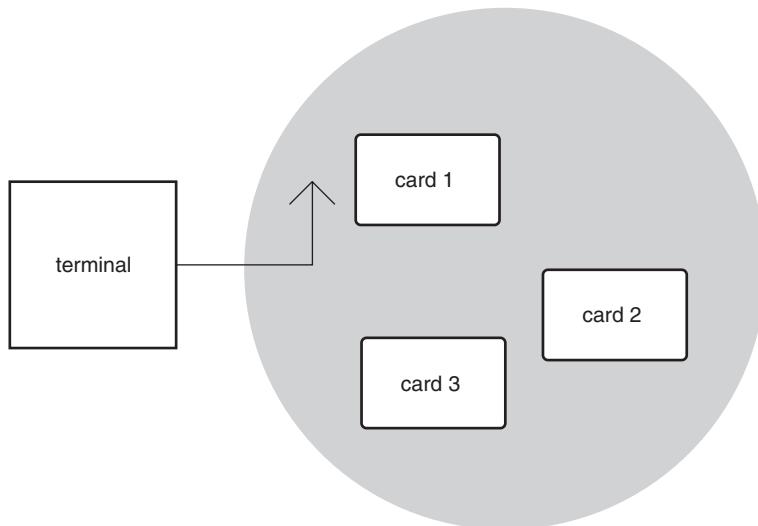
## 10.5 COLLISION AVOIDANCE

When contactless cards are used, there is always a possibility that two or more cards may be located in the working range of a terminal at the same time (see Figure 10.8). This is especially true for systems with large effective ranges, but it can also occur in systems with small ranges – for example, two cards can lie on top of each other and be activated at the same time by the terminal. All cards within range of a particular terminal will attempt to respond to commands from the terminal. These simultaneous data transmissions will unavoidably cause interference and loss of data if suitable countermeasures are not taken. The technical methods used to ensure interference-free data exchanges when several cards are within the working range of a terminal are called collision-avoidance methods or anticollision methods.

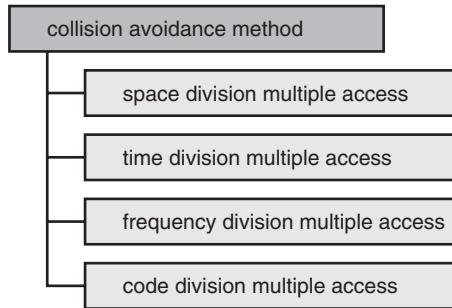
Exchanging data between many mobile stations and a base station is a frequently encountered situation in communication engineering, and it is called multiple access. A typical example is a mobile telephone network, in which all users located in a particular radio cell concurrently access a single base station. Numerous methods have been developed to allow the signals of the individual users to be distinguished from each other. These anticollision methods can be classified into four types, as illustrated in Figure 10.9.

Space division multiple access (SDMA) attempts to constrain or scan the working range of a terminal in such a way that only one card can be acquired at the same time. As this method requires very complex and expensive antennas, it is not used with contactless cards.

With time division multiple access (TDMA), mechanisms are used to ensure that the individual cards have different timing behavior so they can be identified separately and addressed



**Figure 10.8** Concurrent operation of several cards within the range of a terminal (multiple access) requires the use of a collision avoidance method to ensure interference-free data exchange



**Figure 10.9** The four basic types of anticollision methods

individually by the terminal. This is the most commonly used method, and it has many variants. Two of them, which are standardized in ISO/IEC 14433-3 for proximity cards, are described in detail in Section 10.9, ‘Proximity Cards (ISO/IEC 14443)’, on page 297.

With frequency division multiple access (FDMA), different carrier frequencies are used concurrently to support several transmission channels. However, this method is complex and thus expensive. Consequently, it is not used with contactless cards. The same considerations apply to code division multiple access (CDMA).

## 10.6 STATE OF STANDARDIZATION

ISO and IEC began the standardization process for contactless smart cards in 1988. The objective was to generate standards for contactless cards that would enable the integration of the technology for contactless power and data transmission with existing standardized smart cards compliant with ISO/IEC 7816. This means that a contactless card may also have other functional components, such as a magnetic stripe, embossing, or chip contacts. This allows contactless cards to be used in existing systems that employ other technologies.

As already described, the technical options for transmitting power and data without using contacts essentially depend on the desired distance between the card and the terminal for reading and writing data. Consequently, it is not possible to generate a single standard that provides a single technical solution to all of the requirements arising from various applications.

Presently, three different standards describing contactless cards for three different reading ranges have been completed (see Table 10.1). Each of these standards in turn allows various options, since the members of the standardization committee could not agree on a single

**Table 10.1** Completed ISO/IEC standards for contactless smart cards. Each standard consists of several parts, which are described in the text

Standard	Type of contactless smart card	Range
ISO/IEC 10536	Close coupling (CICC)	Up to $\approx 1$ cm
ISO/IEC 14443	Proximity coupling (PICC)	Up to $\approx 10$ cm
ISO/IEC 15693	Vicinity coupling (VICC)	Up to $\approx 1$ m

solution. In order to achieve interoperability between the various options, card terminals must support all of these options.

Standardization began with close-coupling cards (ISO/IEC 10536) because the microprocessors available at that time had relatively high current consumption, so it was not possible to transmit sufficient power over larger distances. In practice, this type of card offers only small advantages over normal contact cards, since it must be inserted in a terminal or placed on a marked area on the surface of a terminal. In the late 1980s, there was not yet much experience with large-scale use of contact smart cards, and it was feared that problems due to wear, dirt, or contact damage could occur in everyday use. However, experience during subsequent years has shown that these fears were unfounded and it is not worthwhile to use contactless technology if the only reason is to avoid contact problems. In addition, the structure of close-coupling cards is complex, which leads to high manufacturing costs. Consequently, up to now this technology has not achieved any market presence.

With the constant reduction of the current consumption of microcontrollers as a result of rapid progress in semiconductor technology, it became possible to use them in contactless cards with larger working ranges (up to  $\approx 10$  cm). For many applications, this has the significant advantage that the card does not have to be inserted in a terminal, but instead can be simply held close to a terminal or waved past a terminal. Consequently, work on the standardization of proximity cards and vicinity cards began in 1994. These two types of cards have been very successful in commercial terms, with hundreds of millions now in use. As a result of the rapid development of proximity card technology, the ISO/IEC 14433 standard has been amended repeatedly since its original issue in 2001, while the ISO/IEC 10693 standard for close-coupling cards has remained practically frozen. The ISO/IEC 14433 standard and its amendments are currently being revised to improve readability and ease of understanding. The revised version is now published.

## 10.7 CLOSE-COUPING CARDS (ISO/IEC 10536)

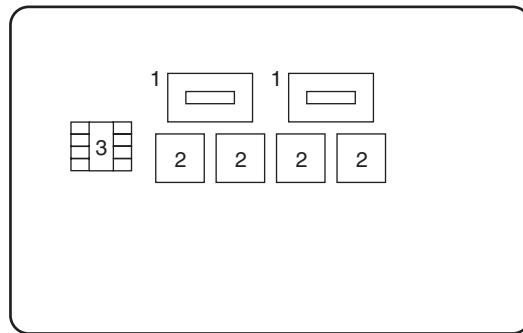
In the ISO/IEC 10536 standard for close-coupling cards, this application is designated ‘slot or surface operation’, which expresses the fact that in use the card must be inserted in a slot or laid on a marked area of the terminal.

The ISO/IEC 10536 standard, which bears the title ‘Identification Cards – Contactless Integrated Circuit(s) Cards’, consists of three parts:

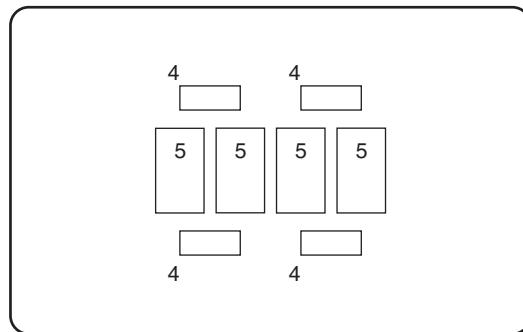
- Part 1: Physical characteristics (Issue 2, 2000)
- Part 2: Dimension and location of coupling areas (Issue 1, 1995)
- Part 3: Electronic signals and reset procedures (Issue 1, 1996)

The main requirements for this standard were broad compatibility with ISO/IEC 7816, operation with any desired orientation of the card relative to the terminal, a carrier frequency range of 3–5 MHz for power and data transmission, bidirectional data transmission with inductive or capacitive coupling, and a card power consumption of less than 150 mW, which is sufficient for using microcontrollers.

Part 1 of the standard defines the physical properties of the card. Essentially the same requirements are imposed as for contact smart cards, particularly with regard to bending and



**Figure 10.10** Arrangement of the coupling components of a contactless card: (1) coupling coils in the card body; (2) capacitive coupling areas in the card body; (3) chip contacts



**Figure 10.11** Arrangement of the coupling components in a terminal for contactless cards: (4) coupling coils in the terminal; (5) capacitive coupling areas on the terminal

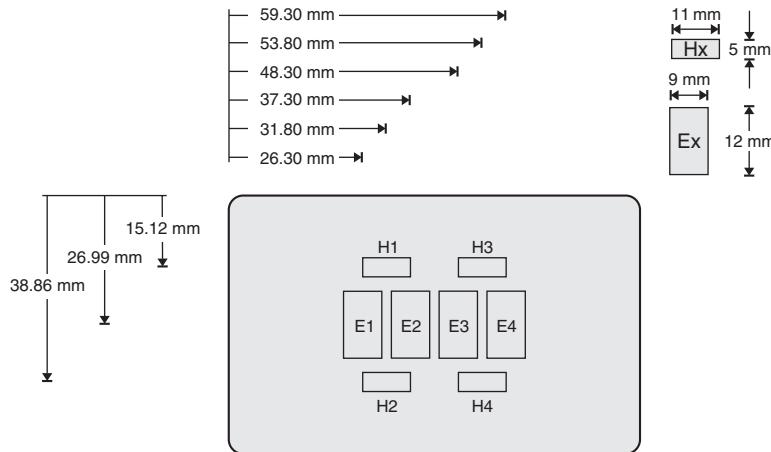
twisting. One difference is in the tolerance to electrostatic discharge (ESD). A contactless card does not require any conductive path between the card surface and the integrated circuit embedded in the card body, so it is largely insensitive to damage from ESD. A test voltage of 10 kV is thus specified in the standard, compared with 1.5 kV for contact cards.

Part 2 of the standard specifies the locations and dimensions of the coupling components. As it was not possible to agree on a single method, capacitive and inductive coupling components are defined in such a way that both types can be implemented in the same card or terminal, as illustrated in Figures 10.10, 10.11, and 10.12. These arrangements are intended to ensure orientation independence with suitable excitation components in the terminal.

Part 3 of the standard, published in 1996, is the most important part to date. It describes the modulation methods to be used for capacitive and inductive data transmission, since agreement on a single method could not be achieved. A terminal that complies with the standard must therefore support both methods, and both methods may be implemented in the same card.

### 10.7.1 Power transmission

Power is transmitted to the card by a sinusoidal alternating magnetic field with a frequency of 4.9152 MHz that passes through one or more inductive coupling areas, depending on how many coils are present in the card. The terminal must generate four different fields.



**Figure 10.12** Locations and dimensions of the coupling areas of the contactless card and the terminal

Alternating magnetic fields F1 and F2 pass through areas H1 and H2 and have opposite phases (phase difference  $180^\circ$ ), while fields F3 and F4 pass through areas H3 and H4 and also have opposite phases. The phase difference between fields F1 and F3 or F2 and F4 is  $90^\circ$ . Each magnetic field is strong enough to transmit at least 150 mW to the card. However, the card should not consume more than 200 mW. This complicated definition of the magnetic fields is necessary to achieve the same data transmission characteristics with four different card orientations, as explained below.

## 10.7.2 Inductive data transmission

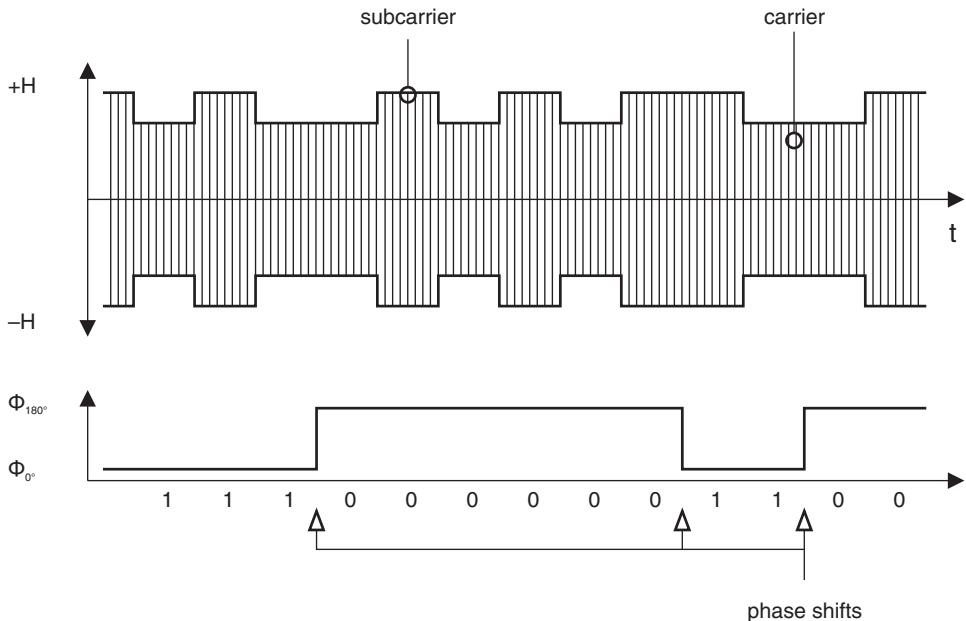
Different modulation methods are used for data transmission in the two directions.

### 10.7.2.1 Transmission from the card to the terminal

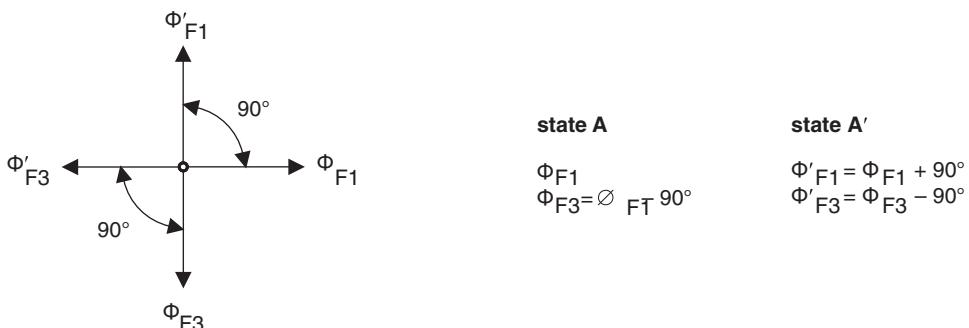
For data transmission from the card to the terminal, a subcarrier is first generated by load modulation at a frequency of 307.2 kHz with a load variation of at least 10 %. Data modulation is then performed by switching the phase of the subcarrier by  $180^\circ$ , producing two phase states that can be interpreted as logic 0 and logic 1 (see Figure 10.13 on the next page). The initial state after the magnetic field has been established is defined to be logic 1. This initial state (interval  $t_3$  in Figure 10.16 on page 295) remains stable for at least 2 ms. After this, every subcarrier phase shift represents a reversal of the logic state, yielding non return to zero (NRZ) coding. The transmission rate, at least for the ATR, is 9600 bit/s.

### 10.7.2.2 Transmission from the terminal to the card

To transmit data from the terminal to the card, the four alternating magnetic fields F1–F4, which pass through coupling areas H1–H4, are phase modulated using phase-shift keying



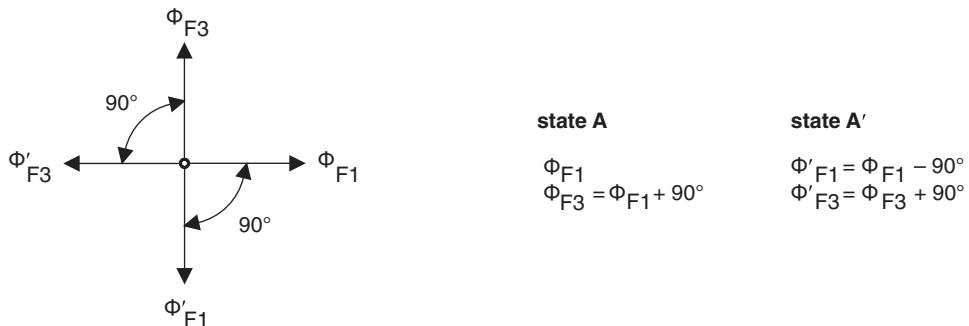
**Figure 10.13** Operating principle of phase modulation for data transmission with a contactless smart card. The upper diagram shows the alternating magnetic field, while lower diagram shows the associated phase states. The carrier frequency is 4.9152 MHz, and the subcarrier frequency is 307.2 kHz



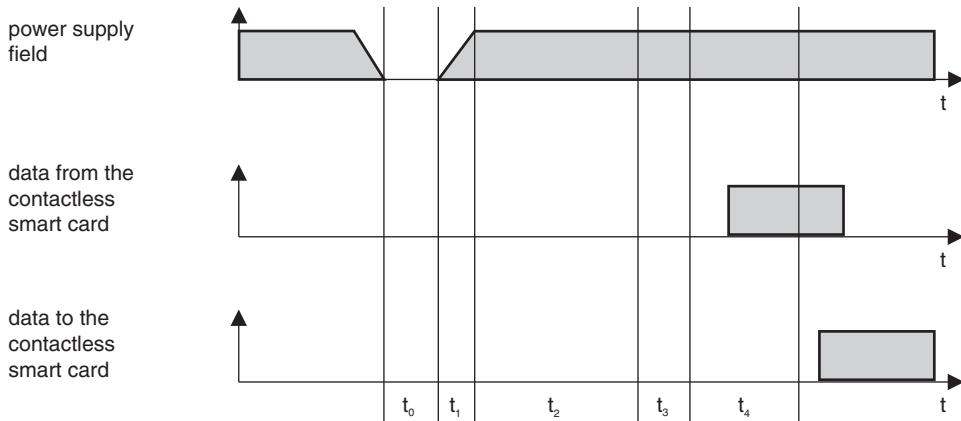
**Figure 10.14** First phase modulation variant for data transmission to contactless cards. The four arrows represent phase vectors

(PSK) such that the phases of all four fields shift simultaneously by 90°. This defines two phase states A and A'. Depending on the orientation of the card relative to the terminal, this yields two different phase state constellations, as shown in Figure 10.14 and Figure 10.15 on the facing page.

As the card must operate properly in all four possible orientations relative to the terminal, the initial state (intervals  $t_2$  and  $t_3$  in Figure 10.15 on the next page) is interpreted as a logic 1, regardless of which of the variants is present. After this, each phase change represents a reversal of the logic state, which again results in NRZ coding.



**Figure 10.15** Second phase modulation variant for data transmission to contactless cards. The four arrows represent phase vectors



**Figure 10.16** Timing diagram for data transmission with a contactless card according to ISO/IEC 10536 3. Here  $t_0 \geq 8$  ms,  $t_1 \leq 0.2$  ms,  $t_2 = 8$  ms,  $t_3 = 2$  ms, and  $t_4 \leq 30$  ms

### 10.7.3 Capacitive data transmission

One pair of coupling areas (either E1 and E2 or E3 and E4 in Figure 10.12 on page 293) is used for capacitive data transmission from the card to the terminal, depending on the orientation of the card relative to the terminal. The other pair of coupling areas can be used for data transmission in the opposite direction. The card uses a specific pair of coupling areas to send the ATR, which allows the terminal to recognize the orientation of the card. The maximum voltage between a pair of coupling areas is limited to 10 V, but it must be greater than the minimum differential voltage threshold of the receiver (300 mV). Differential NRZ encoding is used for data transmission. The transmitter generates the encoding by reversing the voltage between areas E1 and E2 or E3 and E4. Here again, the state representing a logic 1 is established in interval  $t_3$  (see Figure 10.16). After this, every polarity reversal represents a change in the logic state.

**Initial state and answer to reset**

To allow the terminal to unambiguously determine the type of data transmission and the card orientation at the start of a data exchange, time intervals must be defined for initiating power and data transmission. Figure 10.16 on the preceding page shows the constraints and values of the reset recovery time  $t_0$ , power rise time  $t_1$ , initialization time  $t_2$ , stable logic state time  $t_3$ , and answer to reset time  $t_4$ .

**Minimum reset recovery time ( $t_0$ )**

In order to generate a reset by switching the power transmission field off and back on, the time between switching the field off and on again, during which no power is transmitted, must be at least 8 ms.

**Maximum power rise time ( $t_1$ )**

The rise time of the power transmission field generated by the terminal must not exceed 0.2 ms.

**Initialization time ( $t_2$ )**

The initialization time, which is the time allowed for the card to attain a stable operating state, is 8 ms.

**Stable logic state time ( $t_3$ )**

The logic state is held at logic 1 for 2 ms before the answer to reset is sent. During this time, the card and the terminal states are set to logic 1 for inductive data transmission.

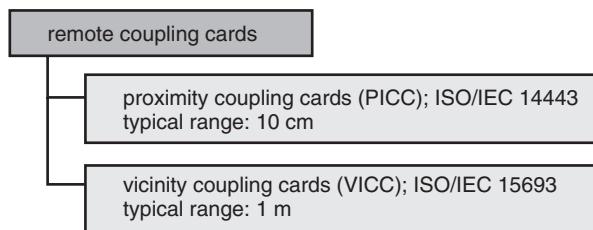
**Maximum ATR response time ( $t_4$ )**

The card must start sending the ATR before 30 ms have elapsed. The card can use the ATR to indicate that the power level, data rate, or field frequency parameters must be changed for subsequent operation. The leeway provided here can be utilized according to the requirements of the application. For example, a significantly higher data rate can be selected for a time-critical application.

## 10.8 REMOTE COUPLING CARDS

The term ‘remote coupling cards’ encompasses smart cards that can transmit data over a range extending from a few centimeters to approximately one meter from the terminal. This capability is very important for all applications in which data must be exchanged between the card and the terminal without requiring the card user to manually insert the card in a terminal. Examples of such applications are access control, vehicle identification, electronic tickets, local public transport, ski passes, airline tickets, electronic purses, and baggage identification.

As can be surmised from this large variety of applications, there are many different technical implementations. An attempt was made during the standardization process to restrict the number of technical variants, with mixed success. The ISO/IEC 14443 and ISO/IEC 15633 standards address coupling ranges up to 10 cm and 1 m, respectively (see Figure 10.17).



**Figure 10.17** The ISO/IEC standards distinguish two types of remote coupling cards: proximity cards and vicinity cards

## 10.9 PROXIMITY CARDS (ISO/IEC 14443)

The ISO/IEC 14443 standard, which is titled ‘Identification cards – Contactless integrated circuit(s) cards – Proximity cards’, describes the properties and operating principles of contactless smart cards with a range of approximately 10 cm. The amount of power that can be transmitted over this range is sufficient to operate a microprocessor. To enable these cards to be used with an existing infrastructure for contact cards, they often have contacts in addition to the coupling components for contactless operation, so they can be used with or without contacts as desired. Cards with this capability are called dual-interface cards.

The ISO/IEC 14443 standard is formulated such that it is compatible with ISO/IEC 7816 (the standard for contact smart cards) at the application level. Although the electrical interface uses contactless coupling instead of direct electrical contact, the formats for exchanging data between the card and the terminal are the same. Consequently, commands and data can be exchanged between the card and the terminal using either interface of a dual-interface card without any difficulties, as illustrated in Figure 2.11 on page 24.

This interoperability between the ISO/IEC 7816 and ISO/IEC 14443 standards allows contactless technology to be utilized in many applications already in widespread use, since it does not matter to the background system whether a contact card or a contactless card is presented at the user interface. If dual-interface cards are used, it is even possible to continue using existing terminals in parallel with the installation of new terminals for contactless operation.

Due to this interoperability, cards compliant with ISO/IEC 14443 can doubtlessly achieve widespread use. In the areas of payment transactions, public transport, travel documents and access control alone, the number of microcontroller cards compatible with ISO/IEC 14443 was more than 100 million in 2008.

Although the card’s range limitation of approximately 10 cm may appear to be inconvenient for some applications, it provides a certain amount of protection against undesired access to the card over relatively large distances. The large antennas in the immediate vicinity of a terminal necessary for eavesdropping on data transmissions could hardly be installed in an unobtrusive manner. Ultimate protection against attacks on the card data is provided by authentication and encryption methods such as those used for this purpose with contact smart cards.

The first issue of ISO/IEC 14443 was published in 2001. Experience with implementations in the following years created a need for corrections and additions. In particular, experience with applications gave rise to a demand for data rates greater than the value of 106 kbit/s specified in Part 2 of ISO/IEC 14443. High data rates and correspondingly short transaction

times are especially important in contactless applications. This is because the card must be within the working range of the terminal for the entire communication interval, which means that the user must hold the card close to the terminal for the duration of this interval. This becomes increasingly inconvenient as the length of this interval increases, with a correspondingly higher probability of unintentional removal of the card from the working range of the terminal. Consequently, a transaction time of a few tenths of a second is required for access control systems in particular.

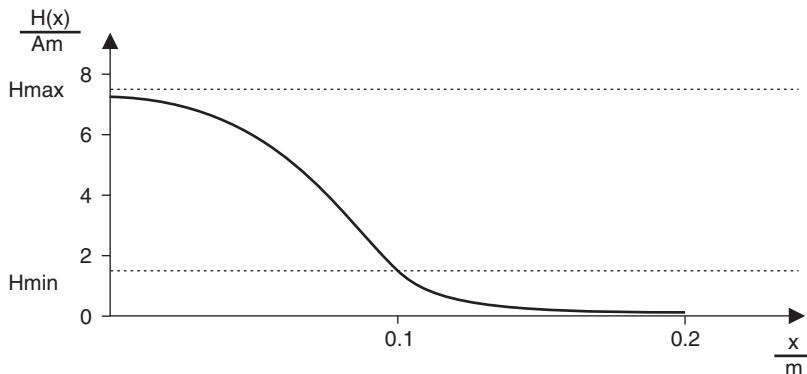
This requirement has been addressed by several amendments to the standard, with the result that ISO/IEC 14443 currently has the structure listed below, with additional amendments currently in the preparation stage:

- Part 1: Physical characteristics
- Part 2: Radio frequency power and signal interface
- Part 2 Amendment 1: Bitrates of fc/64, fc/32 and fc/16
- Part 2 COR1: ISO/IEC 14443-2 / AM1 Defect report and technical corrigendum 1
- Part 3: Initialization and anticollision
- Part 3 Amendment 1: Bitrates of fc/64, fc/32 and fc/16
- Part 3 COR1 ISO/IEC 14443-3 / AM1 Defect Report and Technical corrigendum 1
- Part 3 Amendment 3: Handling of reserved fields and values
- Part 4: Transmission protocol
- Part 4 Amendment 1: Handling of reserved fields and values

The clarity of the standard is reduced by the large number of amendments. Consequently, the responsible ISO/IEC working group is working on a revised version of the standard. As it is not yet available, here we can only take into account the current status of the amendments as listed above. Additional amendments may result from the revision of the standard. Accordingly, readers are strongly advised to consult the current version of the standard.

### 10.9.1 Physical properties

The physical properties of proximity integrated circuit cards (PICCs) specified in Part 1 of ISO/IEC 14443 essentially correspond to the requirements specified for smart cards with contacts. The dimensions match the dimensions of ID-1 cards specified in ISO/IEC 7810. This ensures that the cards will fit in terminals designed for magnetic-stripe cards or contact cards. It must be expected that in normal use, proximity cards will be exposed to electromagnetic fields corresponding to those intended for use with other types of cards, such as cards that comply with the ISO/IEC 10536 or ISO/IEC 15693 standard. The cards must not suffer permanent damage as the result of exposure to these fields or the effects of exposure to normal ambient electromagnetic fields. In order to ensure this, the standard specifies the maximum strengths of alternating electric and magnetic fields that the cards must withstand without damage. It is the responsibility of the semiconductor manufacturer to design chips that meet these requirements.



**Figure 10.18** Typical magnetic field strength characteristic of a terminal (PCD) for proximity cards

### 10.9.2 Power transmission and signal interface

Proximity cards operate on the principle of inductive coupling. Operating power and data are both transmitted using an alternating magnetic field generated by the card terminal. In the ISO/IEC 14443 standard, the card terminal is called a proximity coupling device (PCD). For the sake of readability, the general term ‘terminal’ is used along with the abbreviation ‘PCD’ in the following descriptions.

The transmission frequency  $f_c$  of the PCD is specified as  $13.56 \text{ MHz} \pm 7 \text{ kHz}$ , with a magnetic field strength  $H$  of at least  $1.5 \text{ A/m}$  and at most  $7.5 \text{ A/m}$  (effective value). A typical curve of field strength versus distance for a PCD is shown in Figure 10.18.

The range of the system can be estimated from the field strength characteristic of the PCD and the activation field strength of a proximity card (PICC). With the typical field strength curve shown in Figure 10.18 and an assumed PICC activation field strength of  $1.5 \text{ W/m}$ , the range is approximately 10 cm.

### 10.9.3 Signal and communication interface

The ISO/IEC 14443 standard defines two different communication interfaces, which are designated Type A and Type B. The reason for standardizing two different protocols does not arise from technical considerations, but instead from the fact that several technical concepts from various manufacturers already existed when the ISO/IEC 14443 standard was being generated. As is unfortunately often the case with standardization, the diverging interests of the involved parties prevented them from agreeing on a single protocol, although this would have been more sensible in the interest of easy interoperability. As a compromise solution, agreement was reached on the two protocols mentioned above, and they were published as an international standard in June 2001. Card terminals must support both protocols in order to achieve full interoperability with all cards compliant with ISO/IEC 14433, since most cards support only one protocol.

When the terminal is in the idle state while waiting to detect a proximity card, it must periodically switch between the two communication protocols. This enables it to recognize

both Type A and Type B cards. Once the PCD has recognized a card, it continues to use the corresponding communication protocol until the card is deactivated by the terminal or leaves the working range of the terminal.

Of course, it can happen that Type A cards receive a Type B command or Type B cards receive a Type A command during polling. To prevent this from blocking communication, Part 3 of the ISO/IEC 14443 standard and Amendment 1 to Part 3 specify rules for the card response in such situations. A basic rule is that the card must return to the idle state after receiving an unmodulated signal for 5 ms. The standard describes examples of the card response to unintelligible commands.

#### 10.9.4 Type A communication interface

During initialization and anticollision, a bit rate of  $f_c/128$  ( $\approx 106$  kbit/s) is used for data transmission from the terminal to the card with Type A cards. After initialization and anticollision, any of the following bit rates can be used for data transmission:  $f_c/128$  ( $\approx 106$  kbit/s),  $f_c/64$  ( $\approx 212$  kbit/s),  $f_c/32$  ( $\approx 424$  kbit/s), and  $f_c/16$  ( $\approx 847$  kbit/s).

##### Data transmission from the terminal to the card

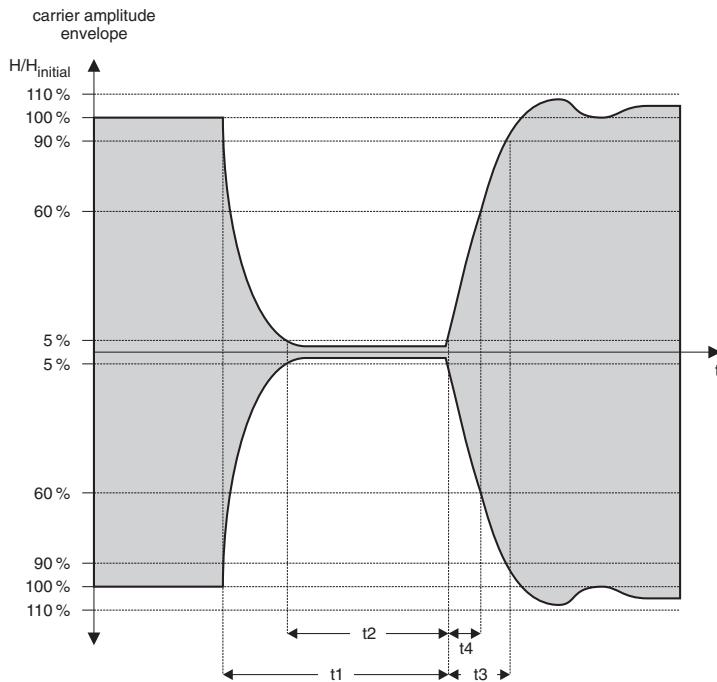
Digital amplitude modulation (100 % ASK) at a bit rate of  $f_c/128$  with modified Miller coding is used for data transmission from the PCD to the card, with a maximum pause interval of 3  $\mu$ s. This relatively short pause makes it easier to provide a continuous supply of power to the card. Figure 10.19 on the facing page shows the exact specification of the pause interval and the related fall and rise times. The card recognizes the end of the pause during time interval  $t_4$ , which means after the magnetic field has reached 5 % of  $H_{\text{initial}}$  and before it exceeds 60 % of  $H_{\text{initial}}$ . Overshoots must be limited to  $H_{\text{initial}} \pm 10\%$ .

##### Data transmission from the terminal to the card at bit rates of $f_c/64$ , $f_c/32$ , and $f_c/16$

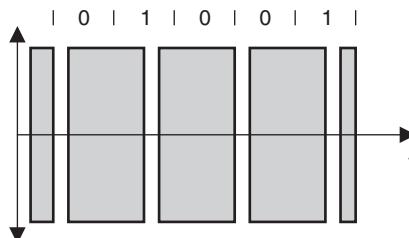
Amplitude modulation with a modulation index of at least 40 % is used at relatively high bit rates such as  $f_c/64$  ( $\approx 212$  kbit/s),  $f_c/32$  ( $\approx 424$  kbit/s) or  $f_c/16$  ( $\approx 847$  kbit/s) to avoid high-frequency interference. The detailed pause parameters are specified in Amendment 1 of the standard.

Figure 10.20 on the next page shows an example of the coding of a bit string using modified Miller coding. The following coding rules apply here:

- Logic 1: pause after half the bit interval.
- Logic 0: no pause, with the following exceptions: (a) if there are two or more logic 0 states in succession, there is a pause at the start of the bit interval; (b) if the first bit of a protocol frame is 0, it is represented by a pause at the start of the bit interval.
- Start of message: pause at the start of a bit interval.
- End of message: logic 0 followed by one bit with no pause.
- No data: no pause for the duration of at least two bits.



**Figure 10.19** Specification of the pause interval according to ISO/IEC 14443-2 Type A with a bit rate of  $f_c/128$ . The maximum pause interval is limited to  $3 \mu\text{s}$  to minimize the interruption of power transmission to the card. Here  $2.0 \mu\text{s} \leq t_1 \leq 3.0 \mu\text{s}$ ; if  $t_1 > 2.5 \mu\text{s}$ , then  $0.5 \mu\text{s} \leq t_2 \leq t_1$ , and if  $t_1 \leq 2.5 \mu\text{s}$ , then  $0.7 \mu\text{s} \leq t_2 \leq t_1$ ,  $0 \mu\text{s} \leq t_3 \leq t_1$ , and  $0 \mu\text{s} \leq t_4 \leq 0.4 \mu\text{s}$

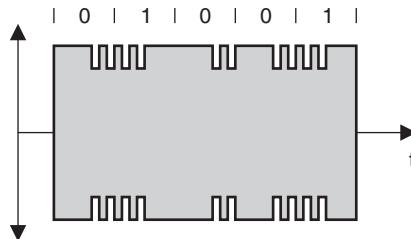


**Figure 10.20** Coding of a bit sequence from the terminal to the card using a Type A communication interface with 100 % ASK, modified Miller coding, and 106 kbit/s. The figure shows the voltage on the terminal antenna

### Data transmission from the card to the terminal

The bit rate for data transmission from the card to the terminal is also  $f_c/128$  ( $\approx 106 \text{ kbit/s}$ ). The following bit rates can be used for data transmission after initialization and anticollision:  $f_c/128$  ( $\approx 106 \text{ kbit/s}$ ),  $f_c/64$  ( $\approx 212 \text{ kbit/s}$ ),  $f_c/32$  ( $\approx 424 \text{ kbit/s}$ ), and  $f_c/16$  ( $\approx 847 \text{ kbit/s}$ ).

Load modulation with a subcarrier is used, which means that the subcarrier is generated by switching a load in the card. The frequency of the subcarrier is specified as  $f_s = f_c/16$  ( $\approx 847 \text{ kHz}$ ). At a bit rate of  $f_c/128$  ( $\approx 106 \text{ kbit/s}$ ), the subcarrier is modulated by on/off keying (OOK) of the subcarrier with Manchester coding.



**Figure 10.21** Load modulation for data transmission from the card to the terminal using a subcarrier with a frequency of  $f_c/16$  ( $\approx 847$  kHz) and Manchester coding with a bit rate of 106 kbit/s and OOK. The figure shows the voltage on the card coil

Figure 10.21 shows an example of the coding of a bit sequence. The coding is defined as follows:

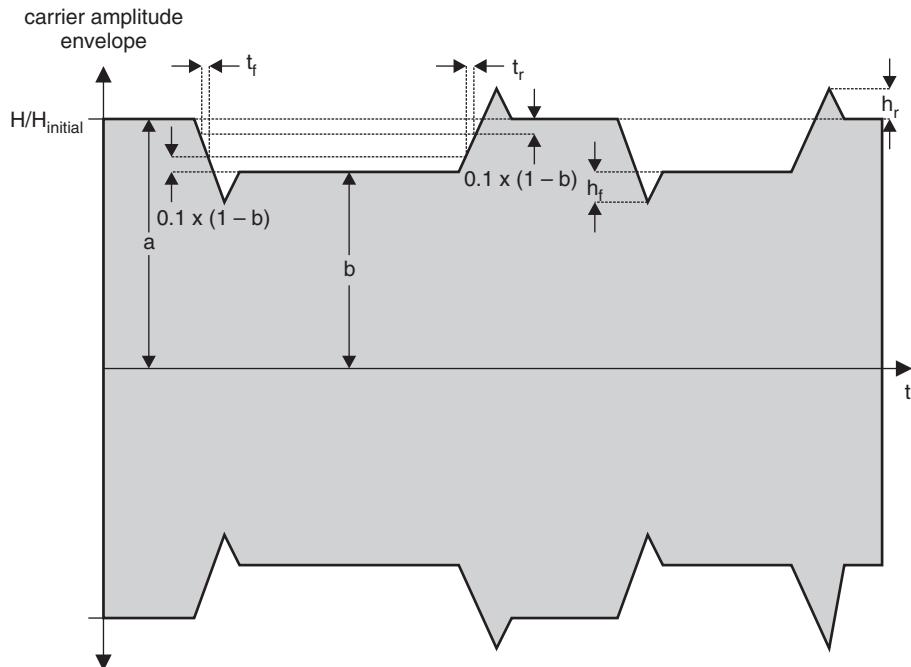
- Logic 1: the carrier is modulated by the subcarrier in the first half of the bit interval.
- Logic 0: the carrier is modulated by the subcarrier in the second half of the bit interval.
- Start of message: the carrier is modulated by the subcarrier in the first half of the bit interval.
- End of message: the carrier is not modulated for one bit interval.
- No data: no subcarrier modulation.

Modulation using on/off keying of the subcarrier has proven to be less suitable for higher bit rates. Consequently, binary phase shift keying (BPSK, subcarrier modulation with 180° phase shift) using NRZ coding is used for bit rates of  $f_c/64$ ,  $f_c/32$  or  $f_c/16$ , as with Type B data transmission.

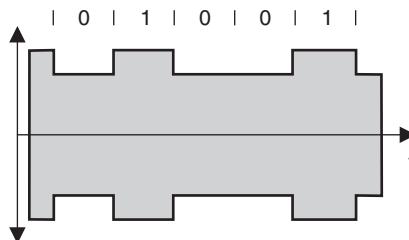
## 10.9.5 Type B communication interface

### 10.9.5.1 Data transmission from the terminal to the card

With Type B cards, ASK modulation with a modulation index of 10 % (-2 %, +4 %) is used for data transmission from the terminal to the card. Unlike Type A, where a continuous supply of power is achieved by using very short pauses, with Type B this is achieved by using a relatively low modulation index defined such that at least 86 % of the carrier field is always available. Here again, the bit rate during initialization and anticollision is  $f_c/128$  ( $\approx 106$  kbit/s). Any of the following bit rates can be used after initialization and anticollision:  $f_c/128$  ( $\approx 106$  kbit/s),  $f_c/64$  ( $\approx 212$  kbit/s),  $f_c/32$  ( $\approx 424$  kbit/s), and  $f_c/16$  ( $\approx 847$  kbit/s). Figure 10.22 on the next page shows the specific form of Type B modulation. As illustrated in Figures 10.23 and 10.24, simple non return to zero (NRZ) bit coding is used, and the coding rules specify that the greater carrier amplitude represents logic 1 and the lesser carrier amplitude represents logic 0.



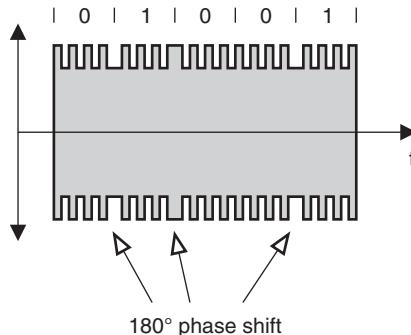
**Figure 10.22** Type B carrier modulation. The low modulation index enables a continuous supply of power. The values of the rise and fall times ( $t_r$  and  $t_f$ ) and the overshoots ( $h_r$  and  $h_f$ ) depend on the bit rate and are specified in the standard



**Figure 10.23** Coding of a bit sequence from the terminal to the card using a Type B communication interface with 10 % ASK, NRZ coding, and a bit rate of 106 kbit/s. The figure shows the voltage on the terminal antenna

#### 10.9.5.2 Data transmission from the card to the terminal

Load modulation with a subcarrier is also used for data transmission from the card to the terminal with Type B cards. Here as well, the subcarrier frequency  $f_s$  is  $f_c/16$  ( $\approx 847$  kHz). In contrast to Type A, the subcarrier is modulated using binary phase shift keying (BPSK) and NRZ coding, with a bit rate of  $f_c/128$  ( $\approx 106$  kbit/s) during initialization and anticollision. Any of the following bit rates can be used after initialization and anticollision:  $f_c/128$  ( $\approx 106$  kbit/s),  $f_c/64$  ( $\approx 212$  kbit/s),  $f_c/32$  ( $\approx 424$  kbit/s), and  $f_c/16$  ( $\approx 847$  kbit/s).



**Figure 10.24** Coding of a bit sequence from the card to the terminal with a Type B communication interface and a subcarrier frequency  $f_c/16$  ( $\approx 847$  kHz), BPSK, NRZ coding, and a bit rate of 106 kbit/s. The figure shows the voltage on the card coil

To achieve an unambiguous initial state, the sequence described here must be followed at the start of each protocol frame. No subcarrier is generated during a guard time interval  $T_{RO} > 64/f_s$  after receipt of data from the terminal. After the guard time, the card generates the subcarrier without phase shift keying for a synchronization time  $TR_1 > 80/f_s$ . The phase during this time is defined as the reference phase  $\varphi_0$ . The initial phase  $\varphi_0$  is defined to be logic 1, so the first phase shift represents a change from logic 1 to logic 0. For the duration of the subsequent data transmission,  $\varphi_0$  represents logic 1 and  $\varphi_0 + 180^\circ$  represents logic 0.

### 10.9.6 Initialization and anticollision (ISO/IEC 14443-3)

When a proximity card enters the working range of a terminal, communication between the card and the terminal must first be established. The terminal may already be communicating with another card, or several cards may be present in the working range of the terminal at the same time. In such situations, it is necessary to have some way to select a specific card or to communicate with a card without interference.

Before the terminal can initiate communication with a card, it must determine whether a card is present in its working range and what type of card is present. To this end, it periodically alternates between Type A and Type B communication until it detects a card. After this, the terminal continues to use the communication method appropriate to the card until it deactivates the card or the card leaves the working range of the terminal. Part 3 of ISO/IEC 14443 specifies a number of rules for this polling process, in particular the timing characteristics. It also describes the process of establishing communication between the card and the terminal and the anticollision methods to be used for selecting a specific card. Due to the different modulation methods, Type A and Type B cards have different protocol frames and anticollision methods.

Part 3 of the ISO/IEC 14443 standard was originally published in 2001 and subsequently amended in 2005 and 2006. Amendment 1 specifies higher data rates (up to 847 kbit/s) in order to satisfy application requirements for shorter transaction times. However, this significantly increases the complexity of the implementation of the standard in the terminals. Amendment 3 restricts the number of options, particularly with regard to parameter coding, which improves compatibility between different implementations. As all parts of the ISO/IEC 14443 standard

**Table 10.2** Allowed bit rates for communication between the card and the terminal.  
The bit rate during initialization and anticolision is always  $f_c/128$

Divisor $D$	etu ( $128/(D \times f_c)$ )	Bit rate
1	$128/f_c (\approx 9.4 \mu s)$	$f_c/128 (\approx 106 \text{ kbit/s})$
2 (optional)	$128/(2 f_c) (\approx 4.7 \mu s)$	$f_c/64 (\approx 212 \text{ kbit/s})$
4 (optional)	$128/(4 f_c) (\approx 2.4 \mu s)$	$f_c/32 (\approx 424 \text{ kbit/s})$
8 (optional)	$128/(8 f_c) (\approx 1.2 \mu s)$	$f_c/16 (\approx 847 \text{ kbit/s})$

are currently under revision in order to incorporate the various amendments in the main body of the standard and to improve the clarity and comprehensibility of the standard, readers are strongly urged to consult the current version of the standard in order to be sure that the latest amendments and corrections are taken into account.

Table 10.2 lists typical data rates for communication between the card and the terminal.

#### 10.9.6.1 Type A initialization and anticolision

A dynamic binary search algorithm is used for the initialization and selection of Type A cards. With this method, the terminal must be able to detect data collisions at the bit level. As explained below, collisions at the bit level can be detected with the Manchester coding scheme used here (see Figure 10.29 on page 309), although this requires all cards in the working range of the terminal to transmit their data synchronously.

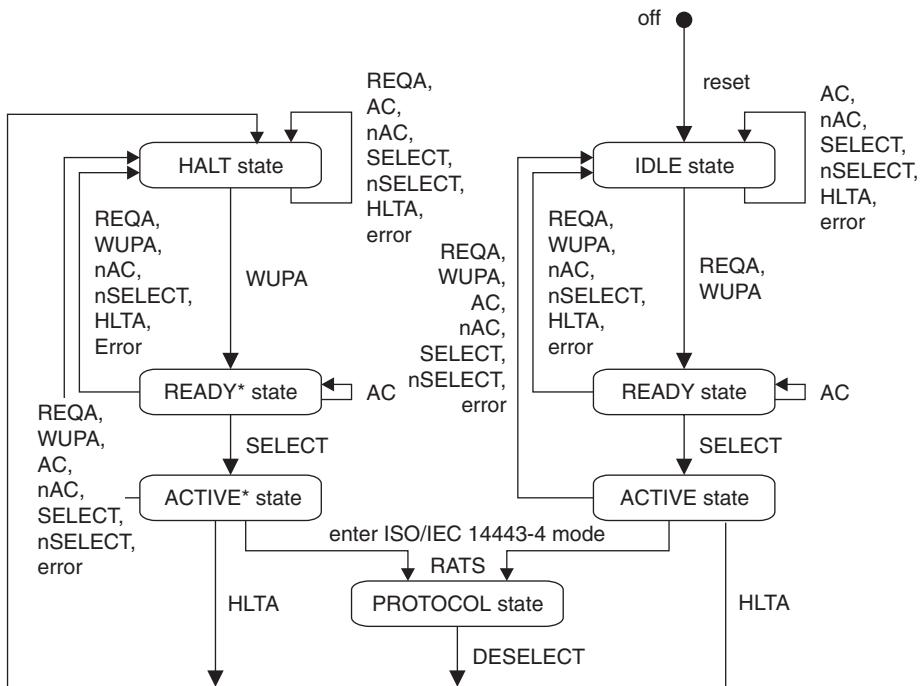
If a proximity card enters the field of a terminal, the microprocessor in the card is supplied with power and the card enters the Idle state after the power-on reset. In this state, the card is only allowed to respond to a Type A Request command (REQA) or a Type A Wake-Up command (WUPA). All other commands sent by the terminal for communication with any other Type A or Type B card already present in the working range of the terminal must be ignored in order to avoid interfering with this communication. Figure 10.25 on the next page shows a state diagram with all possible states that can be assumed by a Type A card during the initialization and anticolision phase.

As already mentioned, the card enters the Idle state after the power-on reset. The standard requires the card to enter the Idle state within 5 ms after it receives adequate operating power from the terminal field. This means that the terminal must wait at least 5 ms after detecting the presence of a card before sending any commands to the card. If the terminal supports both Type A and Type B cards, it can start polling after this 5-ms interval. In the Idle state, the card awaits further commands. It changes to the Ready state when it recognizes a REQA or WUPA command, but it ignores all other commands.

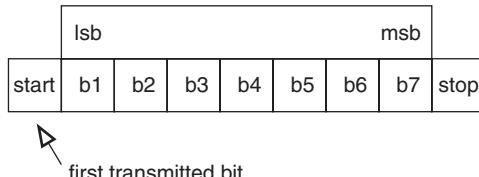
To ensure that REQA and WUPA commands are recognized with a high degree of reliability, they are sent in special short frames. All other commands, except anticolision commands, are sent in standard frames. A special type of frame called a bit-oriented anticolision frame is defined for the anticolision commands.

#### Short frame format

As already mentioned, short frames are used only for initialization commands. A short frame consists of nine bits in the following sequence: a start bit, seven data bits with the least



**Figure 10.25** State diagram of a Type A PICC during initialization and anticollision. The abbreviations are explained in the text



**Figure 10.26** Short frame format

significant bit first, and a stop bit (see Figure 10.26). The coding rules for the start and stop bits and the data bits are described in Section 10.9.4 on page 300.

Table 10.3 on the next page shows the coding of the **REQA** and **WUPA** commands, which are the only commands sent using short frames.

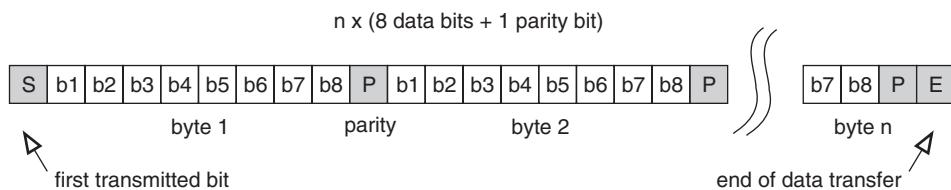
The terminal sends **REQA** and **WUPA** commands to determine whether any cards are present in the working range of the terminal (see Figure 10.25).

### Standard frame

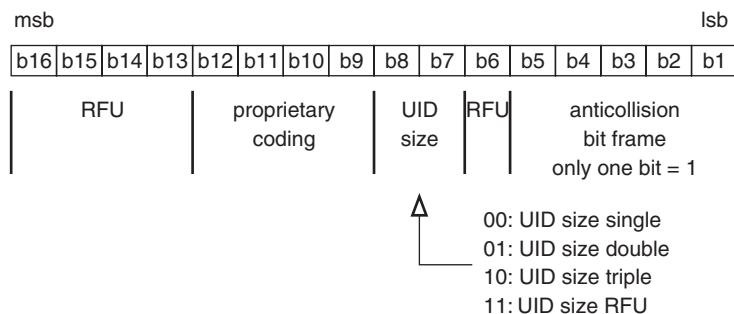
Standard frames are used for regular data exchange. For communication at the bit rate specified for initialization and anticollision ( $f_c/128$ ), a standard frame consists of a start of message bit, a series of  $n$  data bytes (each with an odd parity bit) where  $n \geq 1$ , and an end of message bit (see Figure 10.27).

**Table 10.3** Coding of the REQA and WUPA commands, which use the short frame format with seven data bits

b7	b6	b5	b4	b3	b2	b1	Meaning
0	1	0	0	1	1	0	'26' = REQA
1	0	1	0	0	1	0	'52' = WUPA
0	1	1	0	1	0	1	'35' = optional time slot method
1	0	0	x	x	x	x	'40' ... '4F' = proprietary
1	1	1	1	x	x	x	'78' ... '7F' = proprietary
all other values							RFU



**Figure 10.27** Standard frame format. All parity bits are odd with a bit rate of  $f_c/128$ . The last parity bit of each frame is even with a bit rate of  $f_c/64$ ,  $f_c/32$ , or  $f_c/16$



**Figure 10.28** Coding of ATQA. All RFU bits must be set to 0. Bits 9–12 can be used to indicate other nonstandardized methods. One of the bits b1–b5 must be set to 1. Bits b7 and b8 indicate the size of the UID

When a standard frame is transmitted at a bit rate of  $f_c/64$ ,  $f_c/32$  or  $f_c/16$ , which is only possible after completion of initialization and anticollision, the final parity bit in the standard frame is inverted, which means that the final byte is transmitted with even parity.

After recognizing a Type A request (REQA) or Type A Wake-Up (WUPA) command, the card changes to the Ready state and sends an Answer to Request, Type A (ATQA) after a precisely defined frame delay time (see Figure 10.30 on page 310). An ATQA consists of two bytes, and all ATQA messages are sent synchronously by all addressed cards due to the uniquely specified frame delay time. Figure 10.28 show the coding of ATQA.

When the terminal receives an ATQA, it recognizes that at least one card is present in its working range. It then initiates the anticollision process, which also allows it to read the Type A unique identifier (UID) by sending a SELECT command. After the terminal has determined the complete identifier, it sends a SELECT command containing this identifier. The card with this identifier acknowledges this command by sending a Select Acknowledge (SAK) message and changes to the Active state. In the Active state, the card can communicate using higher-level protocols, such as the protocol defined in ISO/IEC 14443-4.

The card can be put into the Halt state by sending it a Type A Halt command (HLTA). It can also be put into the Halt state by special commands belonging to higher-level protocols. In the Halt state, the card only responds to a Type A Wake-Up (WUPA) command, in which case it sends a Type A Answer to Request (ATQA) and changes to the Ready\* state, which is similar to the Ready state. The conditions for changing to the Active\* state are shown in Figure 10.25 on page 306.

The process for collision avoidance and determining the identifier operates as follows. If two or more cards are concurrently in the Ready state and located in the working range of a terminal, they respond simultaneously to a SELECT command from the terminal by sending part of their individual identifiers. This is done using a special bit-oriented frame that allows the direction of data transmission between the terminal and the card to be reversed after an arbitrary number of data bits have been sent. If several cards sending different data are present, the terminal will receive the data bits superimposed on each other, and it can detect a collision by the fact that this superimposition causes the carrier to be modulated by the subcarrier for the full duration of one or more bit intervals. This is an irregular state, since Manchester coding requires a transition to occur within each bit interval. Figure 10.29 on the next page illustrates how this irregular state arises when there is a collision.

In order for the terminal to detect a collision at the bit level, all cards in the working range of the terminal that are in the Ready state must respond to an ANTICOLLISION command at exactly the same time. To ensure this, the timing of the terminal and the cards for exchanging frames is specified precisely in ISO/IEC 14443-3.

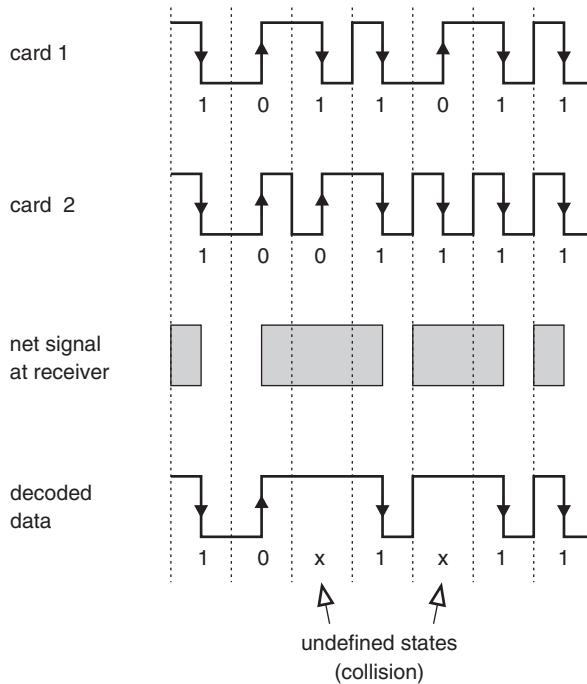
### Frame delay time (FDT)

The time between the end of the last pause sent by the terminal at the end of a message and the leading edge of start bit modulation by the card is called the frame delay time PCD to PICC, which is abbreviated as FDT. This interval is defined in Figure 10.30 on page 310. There are two different cases, depending on whether the final data bit sent by the terminal is a logic 1 or a logic 0.

For the REQ, WUPA, ANTICOLLISION and SELECT commands, the factor  $n$  that determines the value of FDT is set to 9, which means that FDT is  $1236/f_c$  or  $1172/f_c$ . As a result, all cards within the working range of the terminal respond synchronously to these commands used in the anticollision process. This makes it possible to detect collisions at the bit level.

The time between the final modulation sent by the card and the first pause in the signal sent by the terminal is called the frame delay time PICC to PCD. This time is not significant for collision detection at the bit level. Consequently, it is not specified precisely in the standard. The only requirement is that it must be at least  $1172/f_c$ . In addition, the minimum time between two successive REQ commands is specified as  $7000/f_c$ . This is called the request guard time.

As can be seen from the following description, if several cards are present in the working field of the terminal, the terminal may have to make several passes through the anticollision



**Figure 10.29** Collision of two bit sequences with Manchester coding (Type A). With interference-free transmission, the carrier is always modulated by the subcarrier during only one half of each bit interval. If different bit values are superimposed, modulation is present for the entire duration of the bit interval, enabling the terminal to detect a collision

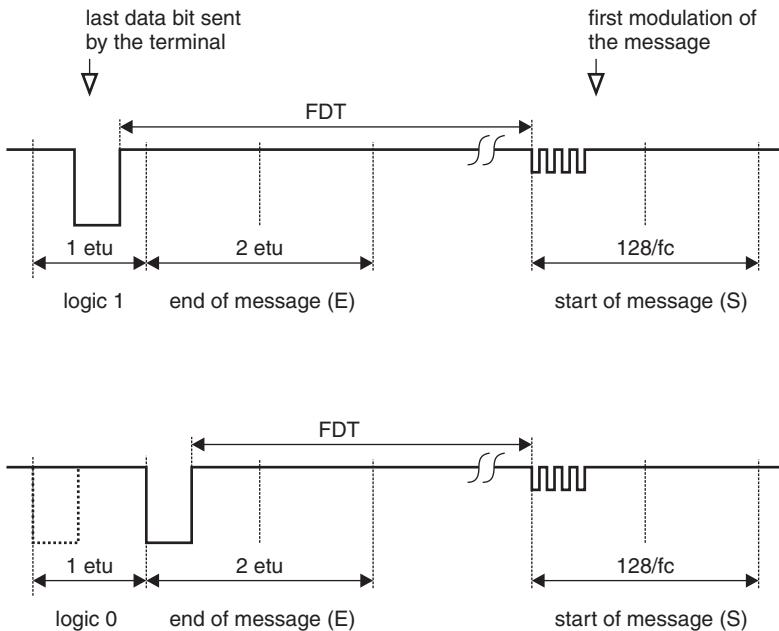
loop before it can communicate with a specific card. To avoid spending too much time in this loop, the anticollision (AC) command (which is a special form of the SELECT command) uses a protocol frame called a bit-oriented anticollision frame as described below.

### Bit-oriented anticollision frame

If the terminal (PCD) detects a collision (see Figure 10.29), it responds by sending one or more bit-oriented anticollision frames, which have the same structure as standard seven-byte frames but are divided into two parts. The first part is used to transmit data from the PCD to the PICC, while the second part is used to transmit data in the opposite direction, from the PICC to the PCD. The relative lengths of the two parts differ in successive passes through the anticollision loop, but the total number of data bits in the two parts is always 56. The following rules apply to the lengths of the two parts:

- The total number of data bits is 56.
- The minimum length of part 1 is 16 data bits.
- The maximum length of part 1 is 55 data bits.

Consequently, part 2 has a minimum length of one data bit and a maximum length of 40 data bits. The frame can be divided into two parts at any desired position, which means that



**Figure 10.30** Frame delay time PCD to PICC (FDT)

the boundary may be located within a data byte. In this case, no parity bit is appended to the first part of the divided byte, and the parity bit of the second part of the divided byte is ignored by the PCD. Figure 10.31 on the facing page shows two examples of anticollision frames.

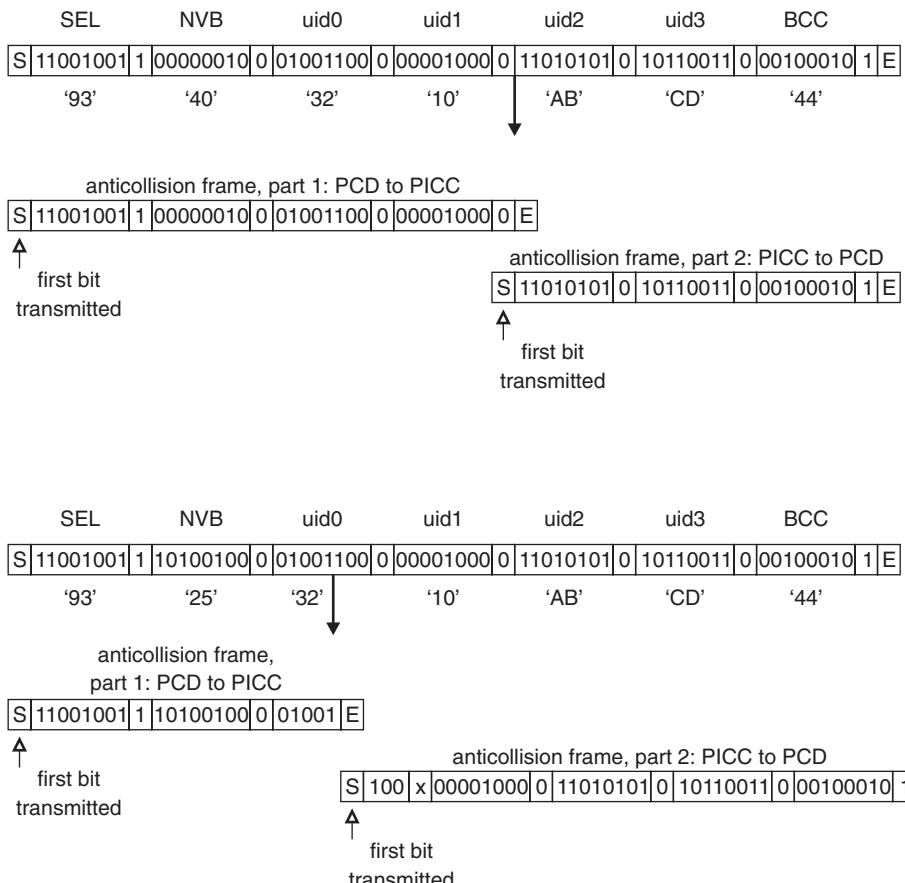
### Commands used in the anticollision loop

As already indicated by the state diagram in Figure 10.25 on page 306, the following commands can be used during initialization and in the anticollision loop: REQA (Type A Request), WUPA (Type A Wake-up), ANTICOLLISION, SELECT, and HLTA (Type A Halt). The REQA and WUPA commands are described in Section 10.9.6.1, ‘Type A initialization and anticollision’, on page 305.

### The ANTICOLLISION and SELECT commands

The ANTICOLLISION and SELECT commands are used in the anticollision loop. As indicated in Figure 10.31 on the facing page, these commands consist of the following data fields: SEL (select code; 1 byte; see Table 10.5), NVB (number of valid bits; 1 byte), and 0 to 40 bits of the unique identifier (UID), depending on the value of NVB. The ANTICOLLISION command is transmitted in a bit-oriented anticollision frame, while the SELECT command is transmitted in a standard frame. The unique identifier (UID) may consist of four bytes (single size), seven bytes (double size), or ten bytes (triple size) (see Table 10.4). The size of the UID is indicated by bits b7 and b8 of ATQA (see Figure 10.28 on page 307).

The UID may be a fixed number or a random number generated by the card. With double- and triple-size identifiers, a cascade tag is sent as the first byte for cascade levels 1 and 2. The cascade tag code is ‘88’. The first byte (uid0) of a single-size UID is not allowed to have the value ‘88’.



**Figure 10.31** Two examples of bit-oriented anticollision frames. The upper diagram shows an anticollision frame split after the fourth complete byte, while the lower diagram shows a frame split after the fifth data byte plus four data bits. The following abbreviations are used here: SEL = Type A select code; NVB = number of valid Type A bits; BCC = UID CL check byte; uidn = byte n of the unique identifier

**Table 10.4** Possible unique identifier (UID) sizes

Cascade level	UID size	Number of UID bytes
1	Single	4
2	Double	7
3	Triple	10

Up to four data bytes can be sent to the PCD in an ANTICOLLISION or SELECT command transaction (see Figure 10.31). If the card has a double- or triple-size unique identifier, it indicates this to the terminal by setting the cascade bit in SAK (Select Acknowledge), and it remains in the Ready state. In response, the terminal starts the anticollision process again to determine bytes 5 to 7 of the UID. With a triple-size identifier, the anticollision process must

**Table 10.5** Coding of the select code byte (SEL), which indicates the current cascade level of the terminal

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	0	0	1	0	0	1	1	'93': cascade level 1 selected
1	0	0	1	0	1	0	1	'95': cascade level 2 selected
1	0	0	1	0	1	1	1	'97': cascade level 3 selected
1	0	0	1	all values other than those listed above				RFU

be performed a third time to determine bytes 8 to 10. To clearly indicate which part of the UID is being requested, the terminal sends the SELECT command to advise the card of its current cascade level. The entire process is illustrated in Figure 10.32 on the next page in the form of a flow chart.

If the NVB byte does not specify 40 bits, the command is an ANTICOLLISION command and the card remains in the Ready or Ready\* state. Once the terminal has determined the full UID, it sends a SELECT command with NVB = '70', which means that 40 data bits are specified. A CRC\_A checksum calculated as described in ISO/IEC 13239 is appended to this command. An example of this calculation is explained in the informational Annex A of ISO/IEC 14433-3. If the card is addressed by a SELECT command containing its full identifier, it changes from the Ready state to the Active state or from the Ready\* state to the Active\* state (see Figure 10.25 on page 306) and sends SAK (Select Acknowledge) to indicate that the UID is complete. Table 10.6 and Table 10.7 on page 314 show the coding of NVB (number of valid bits) and SAK.

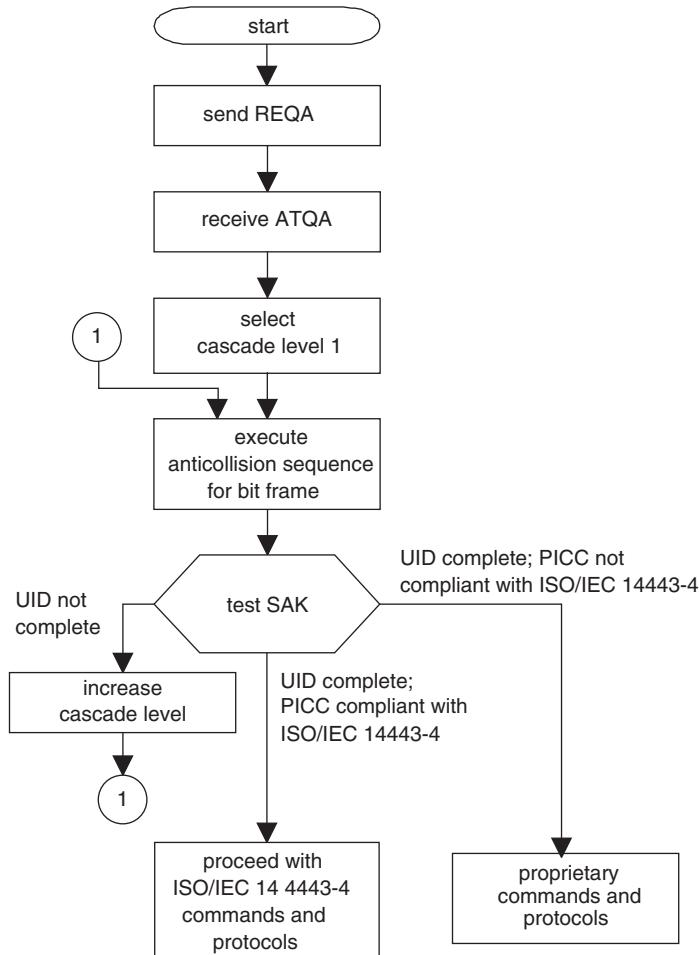
The anticollision loop process as seen by the terminal is shown in Figure 10.34 on page 316 in the form of a flow chart, whose individual steps are described in Table 10.8. This loop must be iterated at each successive cascade level until the terminal knows the full UID. The individual steps of the anticollision algorithm are described in Table 10.8 on page 315.

To illustrate the selection process, Figure 10.35 on page 317 shows an example in which two cards are within range of the terminal. In this example, card 1 (PICC 1) has a single-size UID with a value of '10' for uid0, while card 2 (PICC 2) has a double-size UID.

The terminal initiates the selection process by sending a REQA command. All cards within range of the terminal respond to this command. In this example, PICC 1 responds with an ATQA in which bit b1 is set to indicate an anticollision bit frame and bits b7 and b8 are cleared to indicate a single-size UID. PICC 2 also indicates an anticollision bit frame by setting bit b1 in its ATQA and a double-size UID by setting bit b7.

In the next step, the actual anticollision loop starts at cascade level 1. The terminal sends an ANTICOLLISION command with a Select code of '93', which indicates an anticollision frame for cascade level 1. The value of '20' for NVB (number of valid bits) means that the terminal is not sending any portion of the cascade level 1 UID. Each card within range of the terminal thus responds with its full cascade level 1 UID. The first bit collision occurs at bit b4. The terminal recognizes this bit collision and sends a new ANTICOLLISION command, this time containing the first three bits of the CL1 UID, which were received without errors, followed by a 1, and consequently with a value of '24' assigned to NVB.

The first four bits now match the first four bits of the CL1 UID of card 2, but not the first four bits of the CL1 UID of card 1. Consequently, only card 2 responds with the remaining 36



**Figure 10.32** Flow chart of the process for selecting a specific card. After the terminal sends a REQA command, all cards in the Idle state respond simultaneously with ATQA

bits of its CL1 UID. As the terminal now knows the complete CL1 UID of card 2, it sends a SELECT command for card 2, and card 2 responds with an SAK (Select Acknowledge) with the cascade bit (b3) set. From this, the terminal recognizes that the UID is not yet complete, so it increments the cascade level.

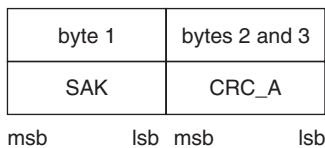
The terminal sends a new ANTICOLLISION command with a select code (SEL) indicating an anticollision bit frame and cascade level 2. NVB is set to '20' in order to request the complete CL2 UID from the card. Card 2 responds to this command by sending all 40 bits of its CL2 UID. The terminal can now send a SELECT command containing all bits of the CL2 UID, to which the card responds with an SAK in which cascade bit b3 is not set, indicating that its UID is complete. Card 2 changes from the Ready state to the Active state, in which it can receive commands for higher-level protocols.

**Table 10.6** Coding of NVB (number of valid bits). The upper four bits are called the byte count and indicate the number of complete data bytes (including SEL and NVB) sent by the terminal. The lower four bits are called the bit count and indicate the number of bits of any incomplete byte sent by the terminal (see Figure 10.31 on page 311)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	1	0	x	x	x	x	Byte count = 2
0	0	1	1	x	x	x	x	Byte count = 3
0	1	0	0	x	x	x	x	Byte count = 4
0	1	0	1	x	x	x	x	Byte count = 5
0	1	1	0	x	x	x	x	Byte count = 6
0	1	1	1	x	x	x	x	Byte count = 7
x	x	x	x	0	0	0	0	Bit count = 0
x	x	x	x	0	0	0	1	Bit count = 1
x	x	x	x	0	0	1	0	Bit count = 2
x	x	x	x	0	0	1	1	Bit count = 3
x	x	x	x	0	1	0	0	Bit count = 4
x	x	x	x	0	1	0	1	Bit count = 5
x	x	x	x	0	1	1	0	Bit count = 6
x	x	x	x	0	1	1	1	Bit count = 7

**Table 10.7** Coding of Select Acknowledge (SAK)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
x	x	x	x	x	1	x	x	Cascade bit set: UID incomplete
x	x	1	x	x	0	x	x	UID complete; PICC compliant with ISO/IEC 14443-4
x	x	0	x	x	0	x	x	UID complete; PICC not compliant with ISO/IEC 14443-4



**Figure 10.33** Format of SAK (Select Acknowledge). SAK is sent by the card when NVB indicates 40 data bits and the data bits match the UID of the card

#### 10.9.6.2 Type B initialization and anticollision

Type B proximity cards use a dynamic slotted Aloha algorithm for selection. With this algorithm, cards within the working range of a terminal send their data to the terminal in predefined time slots. The time slots are selected randomly, and the number of slots can be determined by the terminal. If only a few slots are provided and significantly more cards are present in

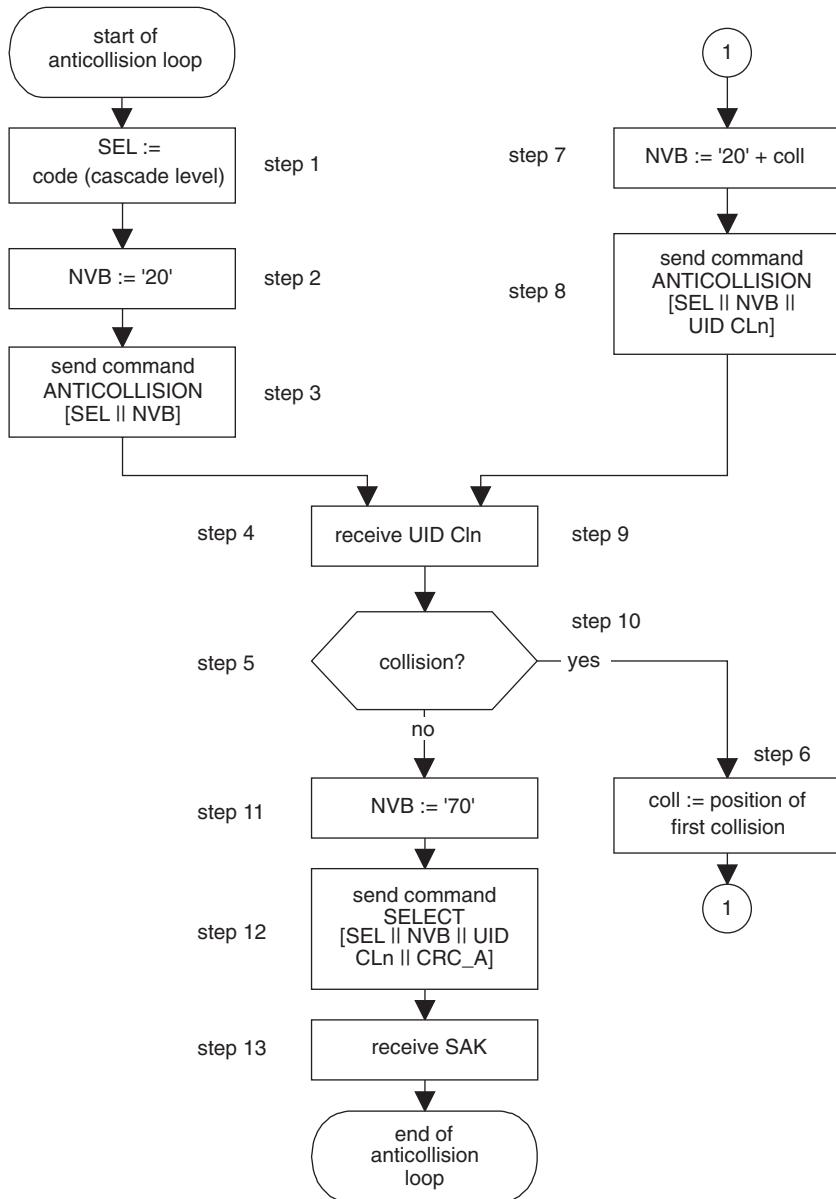
**Table 10.8** The steps of the anticollision algorithm. The numbering corresponds to Figure 10.34 on the next page

Step	Action
1	The terminal sets the select code (SEL) for the cascade level.
2	The terminal sets the NVB byte (number of valid bits) to ‘20’. This indicates that the terminal will not send any part of the UID, which means that every card within the working range of the terminal will send its full $CL_n$ UID (the portion of the UID for cascade level $n$ ).
3	The terminal sends an ANTICOLLISION command containing SEL and NVB.
4	All cards within range of the terminal respond with their $CL_n$ UIDs.
5	If several cards with different identifiers are within range of the terminal, a collision will occur. If no collision occurs, steps 6 through 10 are skipped.
6	The terminal determines the bit position of the first collision.
7	The terminal sets NVB to a value that indicates the number of valid bits of the $CL_n$ UID. The valid bits consist of the portion of the $CL_n$ UID that was received before the collision, followed by a 0 or 1.
8	The terminal sends an ANTICOLLISION command containing SEL, NVB, and the valid bits.
9	Each card with an exact match between the valid bits sent by the terminal and the corresponding bits of its $CL_n$ UID sends the remaining bits of its $CL_n$ UID.
10	If a collision occurs again, steps 6 to 9 are repeated.
11	If no collision occurs, the terminal sets NVB to ‘70’. This means that the terminal will send the full $CL_n$ UID.
12	The terminal sends a SELECT command containing SEL, NVB, and the full $CL_n$ UID, followed by the CRC_A checksum.
13	The card whose identifier matches the $CL_n$ UID responds with SAK.
14	If the UID is complete, the card sends an SAK with the cascade bit cleared and changes from the Ready state to the Active state, or from the Ready* state to the Active* state.
15	The terminal checks whether the cascade bit is set, in order to decide whether another anticollision loop must be executed at a higher cascade level.

the working range of the terminal than the number of available slots, the probability that a card can send its data without interference is small. In this situation, interference-free data transmission is usually only possible if each time slot is used by only one card, or in case a slot is used by more than one card, if one of these cards is significantly closer to the terminal than the others and its data predominates due to its higher signal level. Although increasing the number of slots increases the probability of interference-free data transmission, it has the disadvantage of increasing the amount of time required to execute the polling loop because all of the time slots must be checked each time, even if only one card is present in the working range of the terminal.

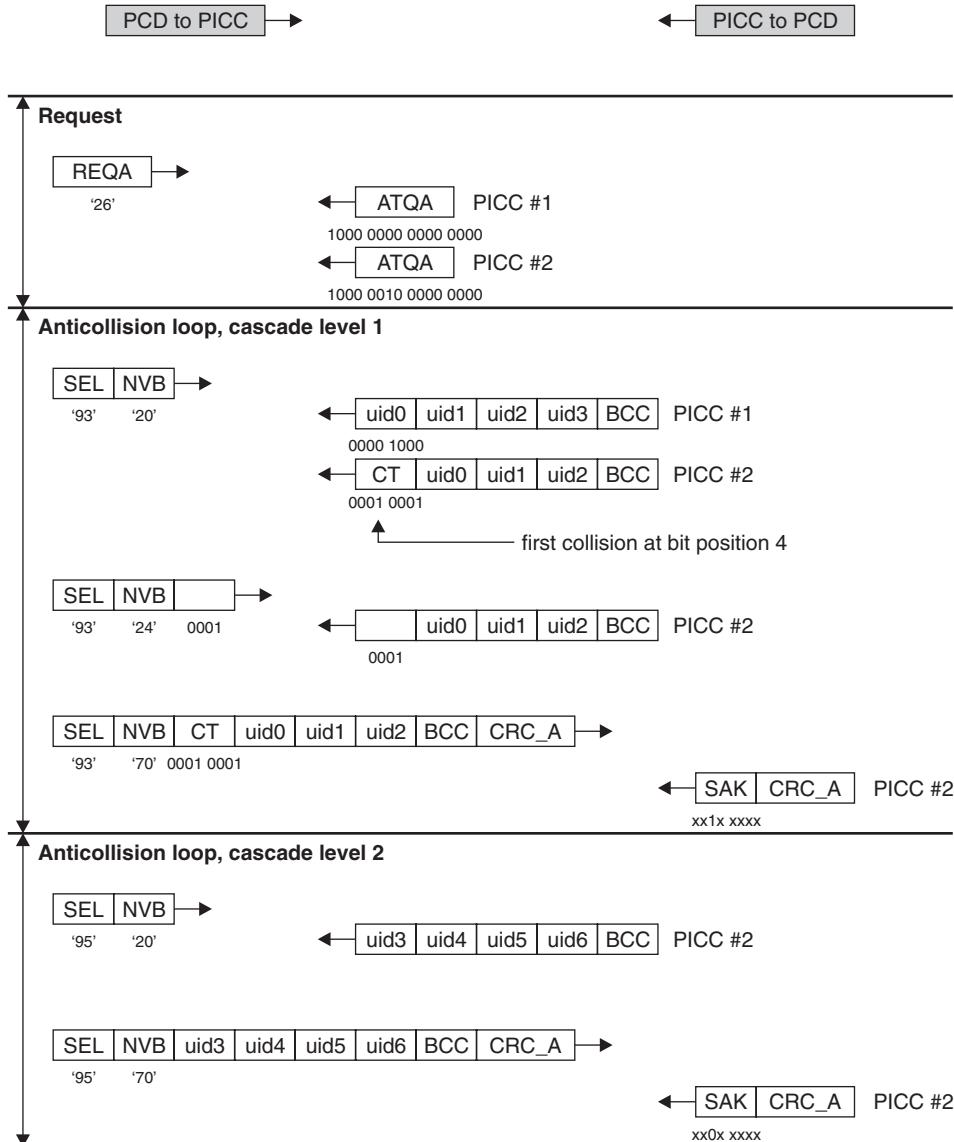
To facilitate understanding of this algorithm, which is specified in ISO/IEC 14443-3, we first illustrate it with a simplified example before describing it in detail.

The terminal needs two commands to execute the algorithm. The first is the REQUEST command, which causes all cards within the working range of the terminal to transmit their unique identifiers in the subsequent time slots. In our example, four time slots are available. The second command is the SELECT command, which sends a previously determined identification number to the cards within range of the terminal. This activates the card with the matching identification number. All cards with other identification numbers remain passive and respond only to REQUEST or SELECT commands with matching identification numbers.



**Figure 10.34** Flow chart of the anticollision loop as seen by the terminal

When the terminal is in standby mode, it periodically sends REQUEST commands. Here we assume that six cards with identification numbers 1 to 6 simultaneously enter the working range of the terminal. All six cards recognize the next REQUEST command at the same time, and they select time slots at random for sending their identifiers to the terminal (see Figure 10.36 on page 318). In this example, collisions occur in time slots 1 and 3, while identification numbers 2 and 3 are sent without interference in time slots 2 and 4. The terminal



**Figure 10.35** Example of a Type A initialization and anticollision sequence. For the sake of simplicity, only the essential elements of the communication process are shown

can now select one of the latter two cards using the SELECT command and then communicate with the selected card without any collision.

When communication with the selected card is completed, the terminal can search for other cards by sending a new REQUEST command. If it does not receive any error-free identification numbers on the first attempt, it sends more REQUEST commands until it receives a valid identification number.



**Figure 10.36** Example anticollision situation with Type B contactless cards. Collisions occur in time slots 1 and 3, so only identification numbers 2 and 3 are sent without interference. This example is described in detail in the text

If significantly more cards are present in the working range of the terminal than the available time slots, the probability of error-free recognition of an identification number is very small because more than one card will send its identification number in each time slot. The number of time slots must be increased in this situation, even though this increases the time required to execute the anticollision algorithm.

Now that the basic principle of this anticollision algorithm has been explained, we can provide a detailed description of the commands and timing characteristics of the cards and the terminal as specified in ISO/IEC 14443-3 for Type B cards. This standard defines a set of commands that allow various types of anticollision strategies to be implemented. This gives users a certain amount of scope for system optimization.

### Formats and timing of characters and frames

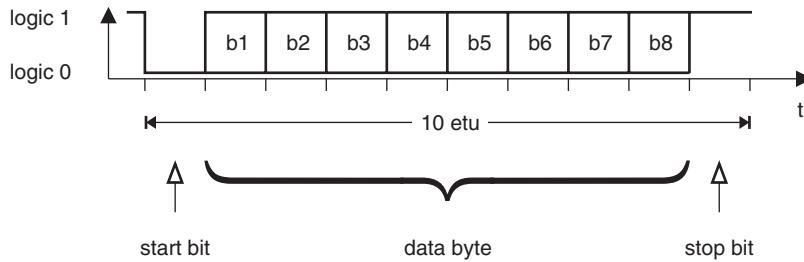
The terminal and the cards send data bytes as characters, with sets of characters grouped into frames. For error detection, a two-byte CRC checksum (CRC\_B) computed according to ISOIEC 13239 is appended to the data bytes in each frame. An example calculation of CRC\_B is provided in Annex B of ISOIEC 14443-3.

Each character consists of one start bit (logic 0) followed by eight data bits (with the least significant bit first) and one stop bit (logic 1), as illustrated in Figure 10.37.

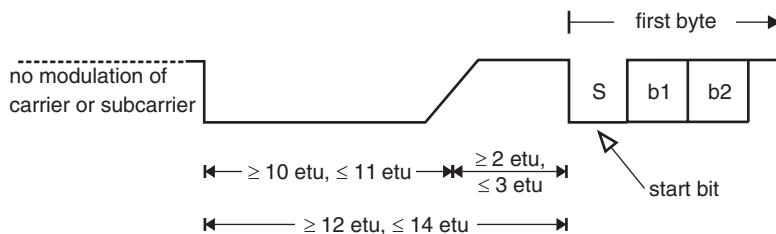
Characters are separated by an interval called the extra guard time (EGT), which gives the transmitter and receiver time to prepare for the next character. For data transmission to the card, the extra guard time ranges from 0 to 6 etu, and in the opposite direction it ranges from 0 to 2 etu.

Sets of characters are grouped into frames for transmission in each direction. A frame begins with a start of frame (SOF) character, followed by the characters to be transmitted and ending with an end of frame (EOF) character. The SOF character consists of a falling edge followed by 10 etu of logic 0. A rising edge occurs within the 11th etu, followed by logic 1 for at least 2 etu and at most 3 etu (see Figure 10.38).

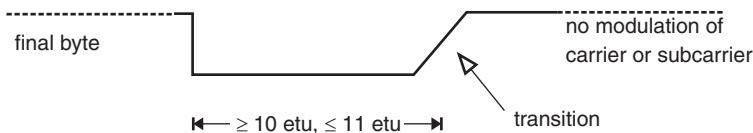
The EOF character also begins with a falling edge, followed by 10 etu of logic 0 and a rising edge within the 11th etu (see Figure 10.39).



**Figure 10.37** Character format. The required transmission time is 10 etu



**Figure 10.38** Timing of the start of frame (SOF) character



**Figure 10.39** Timing of the end of frame (EOF) character

To prevent the transmission protocol from hanging if an error occurs and to provide the card with defined minimum and maximum times for its internal activities, the interval between two frames sent in opposite directions is specified in the standard.

After the card recognizes the EOF of a frame sent by the terminal, it waits for the duration of guard time TR0 before generating the unmodulated subcarrier. After an additional waiting time TR1, which is called the synchronization time, the card starts to modulate the subcarrier. The minimum value of TR0 is  $64/f_s$ , and the maximum value during an anticollision loop (for ATQB) is  $256/f_s$ . For all other frames, the maximum value is calculated using the formula

$$TR0_{\max} = (256/f_s) \times 2^{FWI} - TR1$$

The value of FWI ranges from 0 to 14 and is supplied to the terminal in the ATQB (Answer to Request Type B). The minimum value of the synchronization time TR1 is  $80/f_s$ , and the maximum value is  $200/f_s$ .

When the terminal recognizes the end of a frame sent by the card, it waits for at least the TR2 interval before starting to send a new frame. During this interval, the card switches off the subcarrier within 2 etu of the end of the transmitted frame. The minimum value of TR2 is supplied to the terminal in the Protocol Interval field of ATQB.

**Table 10.9** Format of REQB and WUPB commands

APf	AFI	PARAM	CRC.B
1 byte	1 byte	1 byte	2 bytes

### Card selection process

If a Type B card enters the working range of a terminal, the microprocessor in the card is supplied with power and enters the Idle state after the power-on reset. Like Type A cards, Type B cards must enter the Idle state within 5 ms of receiving sufficient operating power from the terminal field. In the Idle state, the card is not allowed to send any frames. The REQB or WUPB command includes a parameter that indicates the number of time slots provided by the terminal, along with an application family identifier (AFI) that can be used to specify a particular application group. This amounts to a sort of preselection because only those cards whose applications belong to the indicated application family will respond.

### REQB and WUPB commands

The terminal sends REQB or WUPB commands to determine whether any Type B cards are present within its working range. The WUPB command is specifically used to wake up any cards that may be in the Halt state. As shown in Table 10.9, an REQB or WUPB command consists of an anticollision prefix (APf) with a value of ‘05’, followed by the AFI byte, a parameter byte (PARAM) indicating the number of available time slots, and a two-byte CRC.B.

The AFI (application family identifier) byte identifies the type of application supported by the terminal and enables PICC preselection. If the AFI is not ‘00’, only cards with an application corresponding to the AFI are allowed to respond to an REQB or WUPB command. Table 10.10 on the next page shows the coding of AFI, while Table 10.11 on the facing page shows the coding of PARAM. If b4 is 0, the command is a REQB command and will be processed by all cards in the Idle or Ready state. If b4 is 1, the command is a WUPB command and will be processed by all cards in the Idle, Ready, or Halt state. If b5 is 0, the terminal does not support the extended ATQB format. If b5 is 1, the terminal supports the extended ATQB format. The extended ATQB format is described in Section 10.9.6.2 on page 314. The number of time slots is coded as shown in Table 10.12 on page 322.

After a card receives a valid REQB command, it checks whether it supports any of the applications indicated by the AFI. If it does, it examines the PARAM byte to obtain the value  $N$ , which indicates the number of available slots. The card is now in the Ready Requested state.

- If  $N = 1$ , the card sends an ATQB (Type B answer to request) and switches to the Ready Declared state.
- If  $N > 1$ , the card generates a random number  $R$  with a uniform distribution over the range of 1 to  $N$ . If  $R = 1$ , the card sends an ATQB (Type B answer to request) and changes to the Ready Declared state.
- If  $R > 1$ , the card waits until it receives a Slot Marker command with a matching time slot number, at which time it sends an ATQB and changes to the Ready Declared state.

**Table 10.10** Coding of the application family identifier (AFI)

AFI upper nibble	AFI lower nibble	Meaning	Remarks and typical uses
'0'	'0'	All families and subfamilies	No application preselection
X	'0'	All subfamilies of family X	Application preselection
X	Y	Only subfamily Y of family X	
'0'	Y	Only proprietary subfamily Y	
'1'	'0', Y	Transportation	Local public transport, buses, airlines, etc.
'2'	'0', Y	Financial sector	Electronic purses, banks, retail trade, etc.
'3'	'0', Y	Identification	Access control, etc.
'4'	'0', Y	Telecommunication	Public telephone network, mobile telephones, etc.
'5'	'0', Y	Medical	
'6'	'0', Y	Multimedia	Internet services, etc.
'7'	'0', Y	Games	
'8'	'0', Y	Data storage	Portable media, etc.
'9' - 'D'	'0', Y	RFU	
'E'	'0', Y	Machine-readable travel documents	$Y = 1$ : passports; $Y = 2$ : visas; other values of Y: RFU
'F'	'0', Y	RFU	

**Table 10.11** Coding of the PARAM byte. Bit b4 = 0 identifies an REQB command (all cards in the Idle or Ready state respond to this command). Bit b4 = 1 identifies a WUPB command (all cards in the Idle, Ready, or Halt state respond to this command)

b8	b7	b6		b5		b4		b3	b2	b1
RFU				Extended ATQB support		REQB/WUPB		N (number of slots)		

### Slot Marker command

The terminal sends a Slot Marker command at the start of each time slot. Table 10.13 on the next page shows the format of the Slot Marker command.

The terminal can send up to N Slot Marker commands, and it can send each command after it receives an ATQB message or earlier if it does not receive any ATQB message. This means that the terminal does not have to wait until the end of a time slot if it recognizes that the time slot is empty.

The anticollision prefix byte is coded as APn = 'n5', where n is the number of the next slot (see Table 10.14). Slot Marker commands may be sent in any desired order; they do not have to be sent in increasing order of slot number.

After sending a Slot Number command, the terminal waits for an interval  $TR0_{max} = 256/f_s$  before attempting to detect the start of an ATQB response from a card. If the terminal does not detect an ATQB, it can immediately transmit the next Slot Marker command.

**Table 10.12** Coding of N (number of slots). The probability that a card responds in the first time slot is  $1/N$ . If a terminal uses only one time slot,  $N$  indicates the probability that a card will respond in this slot

b3	b2	b1	$N$
0	0	0	$2^0 = 1$
0	0	1	$2^1 = 2$
0	1	0	$2^2 = 4$
0	1	1	$2^3 = 8$
1	0	0	$2^4 = 16$
1	0	1	RFU
1	1	X	RFU

**Table 10.13** Format of the Slot Marker command

APn	CRC_B
1 byte	2 bytes

**Table 10.14** Coding of the slot number in the anticollision prefix byte APn

nnnn	Slot number
0001	2
0010	3
0011	4
...	...
1110	15
1111	16

**Table 10.15** Format of ATQB (Type B answer to request). The Protocol Info field of the extended ATQB format has a length of four bytes instead of three bytes

APa	PUPI	Application data	Protocol Info	CRC_B
'50'	4 bytes	4 bytes	3 or 4 bytes	2 bytes

### Type B answer to request

The ATQB message received from the card in response to a REQB, WUPB, or Slot Marker command is formatted as shown in Table 10.15 (two options).

ATQB contains information about parameters of the smart card that the terminal needs in order to select the card. The extended ATQB format is optional. The card can only use this

**Table 10.16** Format of the Application Data field. The coding of AFI is shown in Table 10.10 on page 321. It indicates the application family in case of a multiapplication card. CRC\_B(AID) is the ISO/IEC 7816-4 CRC\_B checksum of the application identifier (AID) of an application in the card corresponding to the AFI sent in an REQB or WUPB command. The Number of Applications field indicates the number of additional applications in the card. The upper nibble of this byte indicates the number of applications corresponding to the AFI, where ‘0’ means ‘no applications’ and ‘F’ means ‘15 or more applications’. The lower nibble indicates the total number of applications in the card, with the same meaning (‘0’ = ‘no applications’; ‘F’ = ‘15 or more applications’)

AFI	CRC_B(AID)	Number of applications
1 byte	2 bytes	1 byte

**Table 10.17** Format of the Protocol Info field

Byte 1	Byte 2	Byte 3
Bit_Rate_Capability 8 bits	Max_Frame_Size 4 bits	Protocol_Type 4 bits

**Table 10.18** Coding of the Frame Option (FO) parameter

b2	b1	Meaning
1	x	PICC supports NAD
x	1	PICC supports CID

format if the terminal has indicated in the PARAM byte of the REQB or WUPB command that it supports the extended ATQB format.

The pseudo-unique PICC identifier (PUPID) is the identification number of the PICC for the anticollision loop. This may be a number that is permanently assigned to the card, or it may be a random number generated by the card after each power-on reset.

The Application Data field contains information about the applications installed in the card. This information allows the terminal to select the desired card if several cards are present in its working range. The meaning of the application data depends on the Application Data Coding (ADC) field in the Protocol Info. It indicates whether the CRC\_B compression method (described below) or proprietary coding is used. If the CRC\_B compression method is used, the Application Data field is formatted as shown in Table 10.16.

The Protocol Info field indicates significant parameters that supported by the card (see Table 10.17). These parameters allow the terminal to adjust its configuration to best match the capabilities of the card for the subsequent application protocol, or to adapt its configuration in case of cards that do not fulfill all the requirements of the standard.

Tables 10.18 and 10.19 on the next page show the coding of the individual fields of the Protocol Info field.

**Table 10.19** Coding of the Application Data Coding (ADC) parameter

b4	b3	Meaning
0	0	Proprietary application
0	1	Application coded as described in the text

**Table 10.20** Coding of the Protocol Type field in the Protocol Info field

b4	b3	b2	b1	Meaning
0	X	X	1	PICC supports ISO/IEC 14443-4
0	X	X	0	PICC does not support ISO/IEC 14443-4
0	0	0	X	Minimum value of TR2 = 10 etu + 32/f <sub>s</sub>
0	0	1	X	Minimum value of TR2 = 10 etu + 128/f <sub>s</sub>
0	1	0	X	Minimum value of TR2 = 10 etu + 256/f <sub>s</sub>
0	1	1	X	Minimum value of TR2 = 10 etu + 512/f <sub>s</sub>

**Table 10.21** Maximum frame size capacity of the card

Max_Frame_Size code in ATQB	0	1	2	3	4	5	6	7	8	9 - 'F'
Maximum frame size (bytes)	16	24	32	40	48	64	96	128	256	>256 RFU

The Frame Waiting Time Integer (FWI) parameter specifies the maximum amount of time needed by the card to start sending a response after it has fully received a command from the terminal. If the card does not respond within this interval, the terminal can assume that communication with the card has been interrupted.

The frame waiting time (FWT) is calculated using the formula

$$FWT = (256 \times 16/f_c) \times 2^{FWI}$$

FWI has a value of 0 to 14, with 15 reserved for future use (RFU). The minimum and maximum frame waiting time can be calculated using this formula. FWI = 0 yields the minimum value:  $FWT_{min} \approx 302 \mu s$ , while FWI = 14 yields the maximum value:  $FWT_{max} \approx 4.949 \text{ ms}$ .

If the terminal and the card support the extended ATQB format, the FWI parameter can be included in the response to the ATTRIB command. A fixed value  $(256 \times 16/f_c \times 2^4 \approx 4.8 \text{ ms})$  is specified for the response to the ATTRIB command. The shortest possible frame waiting time must be selected to achieve high-speed communication.

The Protocol Type field consists of four bits, in which bit b1 indicates whether the card supports the ISO/IEC 14434 transmission protocol (see Table 10.20). Bits b3 and b2 determine the minimum value of the frame delay time (TR2).

The Max\_Frame\_Size field indicates the maximum frame size the card can receive (see Table 10.21). This is limited by the size of the receive buffer in the card's RAM. Inexpensive chips typically have only a small amount of RAM, so they can only receive small frames.

**Table 10.22** Coding of the Bit\_Rate\_Capability field, which indicates the bit rates supported by the card. All other values are reserved for future use (RFU)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	0	0	0	0	0	0	PICC supports only 106 kbit/s in both directions
1	X	X	X	0	X	X	X	The same bit rate in both directions
X	X	X	1	0	X	X	X	PICC to PCD: 1 etu = $64/f_c$ ,212 kbit/s supported
X	X	1	X	0	X	X	X	PICC to PCD: 1 etu = $32/f_c$ ,424 kbit/s supported
X	1	X	X	0	X	X	X	PICC to PCD: 1 etu = $16/f_c$ ,847 kbit/s supported
X	X	X	X	0	X	X	1	PCD to PICC: 1 etu = $64/f_c$ ,212 kbit/s supported
X	X	X	X	0	X	1	X	PCD to PICC: 1 etu = $32/f_c$ ,424 kbit/s supported
X	X	X	X	0	1	X	X	PCD to PICC: 1 etu = $16/f_c$ ,847 kbit/s supported

The Bit\_Rate\_Capability field indicates the data rates supported by the card, as listed in Table 10.22. This enables the terminal to adjust the data rate to make best use of the card's capability.

The card changes to the Ready Declared state after sending its ATQB (see Figure 10.42 on page 330). In this state, the card only responds to REQB, WUPB, ATTRIB, and HLTB commands. It responds to a received REQB or WUPB command in the same way as when it is in the Idle state.

If the card recognizes a valid ATTRIB command with a PUPI that matches the PUPI of the card, it sends an Answer to ATTRIB frame and changes to the Active state. If the PUPI parameters do not match, the card remains in the Ready Declared state and waits for an ATTRIB command with the right PUPI.

The card responds to a relevant HLTB command (containing a matching PUPI) by sending an Answer to HLTB and changing to the Halt state.

In the Active state, the card has a card identifier (CID) that is assigned to it by the ATTRIB command. This means that it is in a higher protocol layer and responds to suitable application commands having the proper CID and correct CRC\_B checksum.

Specific commands belonging to this higher protocol layer can place the card in the Idle or Halt state. When it is in the Active state, the card is not allowed to respond to REQB, WUPB, Slot Marker, or ATTRIB commands.

In the Halt state, the card is passive and can only be reset to the Idle state by a valid WUPB command with the proper PUPI.

### Format and coding of the ATTRIB command

The ATTRIB command is sent by the terminal to the card and contains information needed to select a card. It also contains information about the parameters supported by the terminal for subsequent communication and those required by the card for error-free communication. This includes the minimum value of the guard time TR0, the minimum value of the synchronization time TR1, whether the card can suppress SOF and/or EOF to accelerate communication, the maximum frame size, and selection of the optimum bit rate.

The parameters specified in the ATTRIB command apply to communication processes after the response to the ATTRIB command. These parameters are defined in Tables 10.23 to 10.30.

The Minimum TR0 parameter defines the minimum time that the card must wait before responding to a command received from the terminal. This is the time needed by the terminal to switch from transmit mode to receive mode, which depends on the capability of the terminal.

The Minimum TR1 parameter defines the minimum delay between the activation of the subcarrier and the start of data transmission (see Figure 10.40 on page 328). The terminal needs this time for synchronization with the card.

Bits b3 and b4 of Param 1 indicate to the terminal whether the card supports EOF and/or SOF suppression from the card to the terminal in order to reduce communication overhead. This capability is optional for the card. SOF or EOF suppression is only possible at the  $f_c/128$  bit rate ( $\approx 106$  kbit/s). It is not possible at higher bit rates.

The lower nibble of Parameter 2 (bits b4–b1) indicates the maximum frame size that can be received from the terminal. The upper nibble is used to select the bit rates in both directions. The terminal can select the bit rates because it already knows the bit rates supported by the card from the ATQB.

**Table 10.23** Format of the ATTRIB command. The Identifier contains the PUPI value sent by the card in the ATQB. The format of Param 1 is shown in Table 10.24

'1D'	Identifier (PUPI)	Param 1	Param 2	Param 3	Param 4	Higher Layer Inf	CRC_B
1 byte	4 bytes	1 byte	1 byte	1 byte	1 byte	0 or more bytes	2 bytes

**Table 10.24** Coding of Param 1

b8	b7	b6	b5	b4	b3	b2	b1
minimum TR0		minimum TR1		EOF	SOF	RFU	RFU

**Table 10.25** Coding of Minimum TR0

Minimum TR0 for bit rates from PCD to PICC					
b8	b7	$f_c/128$	$f_c/64$	$f_c/32$	$f_c/16$
0	0	$64/f_s$	$64/f_s$	$64/f_s$	$64/f_s$
0	1	$48/f_s$	$32/f_s$	$16/f_s$	$16/f_s$
1	0	$16/f_s$	$8/f_s$	$4/f_s$	$4/f_s$
1	1	RFU	RFU	RFU	RFU

**Table 10.26** Coding of Minimum TR1

Minimum TR1 for bit rates from PCD to PICC					
b6	b5	$f_c/128$	$f_c/64$	$f_c/32$	$f_c/16$
0	0	$80/f_s$	$80/f_s$	$80/f_s$	$80/f_s$
0	1	$64/f_s$	$32/f_s$	$32/f_s$	$32/f_s$
1	0	$16/f_s$	$8/f_s$	$8/f_s$	$8/f_s$
1	1	RFU	RFU	RFU	RFU

**Table 10.27** SOF use

b3	SOF required
0	yes
1	no

**Table 10.28** EOF use

b4	EOF required
0	yes
1	no

**Table 10.29** Coding of bits b4–b1 in Parameter 2, which specify the maximum frame size

Max_Frame_Size code in ATTRIB	0	1	2	3	4	5	6	7	8	9 – ‘F’
Maximum frame size (bytes)	16	24	32	40	48	64	96	128	256	>256 RFU

**Table 10.30** Coding of bits b8–b5 in Parameter 2, which select the transmission bit rate

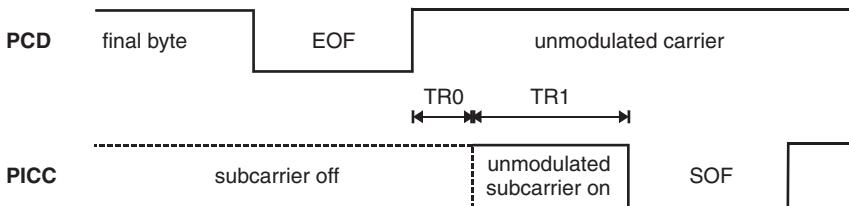
b8	b7	b6	b5	Meaning
0	0	x	x	PICC to PCD; 1 etu = $128/f_c$ , bit rate = 106 kbit/s
0	1	x	x	PICC to PCD; 1 etu = $64/f_c$ , bit rate = 212 kbit/s
1	0	x	x	PICC to PCD; 1 etu = $32/f_c$ , bit rate = 424 kbit/s
1	1	x	x	PICC to PCD; 1 etu = $16/f_c$ , bit rate = 847 kbit/s
x	x	0	0	PCD to PICC; 1 etu = $128/f_c$ , bit rate = 106 kbit/s
x	x	0	1	PCD to PICC; 1 etu = $64/f_c$ , bit rate = 212 kbit/s
x	x	1	0	PCD to PICC; 1 etu = $32/f_c$ , bit rate = 424 kbit/s
x	x	1	1	PCD to PICC; 1 etu = $16/f_c$ , bit rate = 847 kbit/s

The lower nibble of Parameter 3 is used to confirm the protocol type and the minimum frame delay time TR2. The coding is the same as in Table 10.20 on page 324. The upper nibble is set to ‘0’. All other values are reserved for future use.

Parameter 4 also consists of two parts. The lower nibble is called the Card Identifier (CID) and defines the logical number of the addressed card, with a range of 0 to 14; 15 is reserved for future use. The card identifier is specified by the terminal and is unique for each active card. If the card does not support the CID function, a value of ‘0’ is used. The upper nibble is set to ‘0’. All other values are reserved for future use. The Higher Layer Inf field can be used to transmit any desired higher-level command. Processing such commands is optional for the card.

### Response to the ATTRIB command

The card responds to every valid ATTRIB command (with the proper PUPI and correct CRC\_B checksum) as shown in Table 10.31 on the following page.



**Figure 10.40** Definition of the guard time TR0 and synchronization time TR1

**Table 10.31** Format of the response to an ATTRIB command

Byte 1	Bytes 2–n		
MBLI 1 byte	CID 1 byte	Higher Layer Response optional; 0 or more bytes	CRC_B 2 bytes

**Table 10.32** Format of the HLTB command

'50'	Identifier	CRC_B
1 byte	4 bytes	2 bytes

If the terminal receives a valid response to an ATTRIB command (with the same CID and a correct CRC\_B checksum), it knows that card selection was successful.

The lower nibble of the first byte in the response (bits b4–b1) contains the CID. The upper nibble of the first byte (bits b8–b5) is called the Maximum Buffer Length Index (MBLI). The card uses the MBLI to tell the terminal the maximum size of its input buffer. This enables the terminal to avoid causing an input buffer overflow in the card by sending too many chained frames. If MBLI is set to 0, the card does not provide any information about the size of its internal buffer. If MBLI is greater than 0, the maximum internal buffer length (MBL) can be calculated using the formula

$$MBL = (\text{PICCmaximumframesize}) \times 2^{(\text{MBLI}-1)}$$

The card previously sent its maximum frame size to the terminal in the ATQB. If the terminal sends chained frames, it must ensure that the cumulative length does not exceed the value of MBL. If the ATTRIB command does not include a Higher Layer Inf field, the response from the card does not include a Higher Layer Response field.

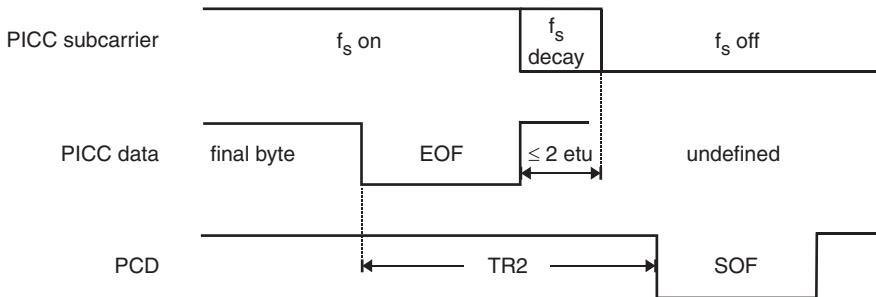
### HLTB command

The HLTB command is used to place a card in the Halt state so that it no longer responds to REQB commands. After responding to this command, the card ignores all subsequent commands except WUPB. Table 10.32 shows the format of the HLTB command.

The Identifier holds the PUPI of the card to be placed in the Halt state. The format of the card's response to a valid HLTB command is shown in Table 10.33 on the next page.

**Table 10.33** Format of the response to a HLTB command

'00'	CRC_B
1 byte	2 bytes



**Figure 10.41** Definition of the waiting time between a frame sent by the card and the following frame sent by the terminal. The card switches off the subcarrier only after the end of the EOF character

### Example anticollision sequence with three Type B cards

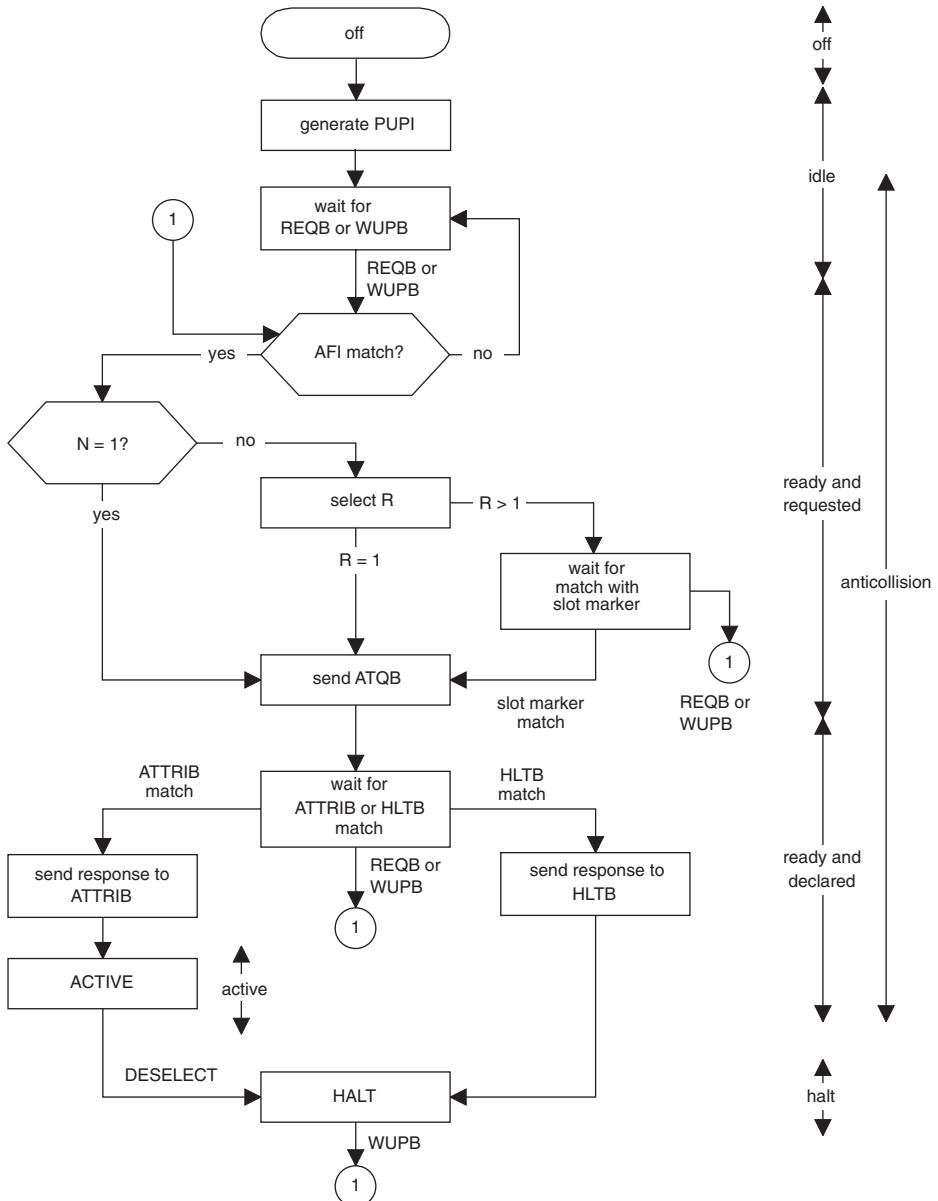
The standard lets developers implement various anticollision strategies. This fits with the basic concept of a standard, which is to facilitate interoperability while allowing as much implementation latitude as possible in order to avoid hampering technological progress. Figure 10.43 on page 331 and Figure 10.44 on page 332, which are included in an annex to the standard, are intended to illustrate the previously described commands and processes. They do not necessarily represent a technically superior implementation.

### 10.9.7 Transmission protocol (ISO/IEC 14433-4)

A half-duplex block-oriented transmission protocol tailored to the specific requirements of contactless systems is defined in Part 4 of ISO/IEC 14443. It is largely based on the T = 1 protocol defined in the ISO/IEC 7816-3 standard, which is widely used throughout the world. Both protocols enable the transmission of application protocol data units (APDUs) as defined in ISO/IEC 7816 Part 4 and described in Section 8.3, ‘Message Structure: APDUS’, on page 221. This simplifies the implementation of dual-interface cards, since they anyhow must support a transmission protocol for contact cards.

For Type A cards, the standard defines an activation sequence that must be executed before the start of the actual protocol. During this sequence, parameters for subsequent data transmission are exchanged between the terminal and the card and set to defined values. This includes parameters such as the bit rate in each direction and the required waiting times between frames.

Type B cards do not need a special activation sequence. They can immediately initiate the actual transmission protocol after being selected. With such cards, the necessary parameters for data transmission are specified and exchanged during the initialization and selection process as previously described.

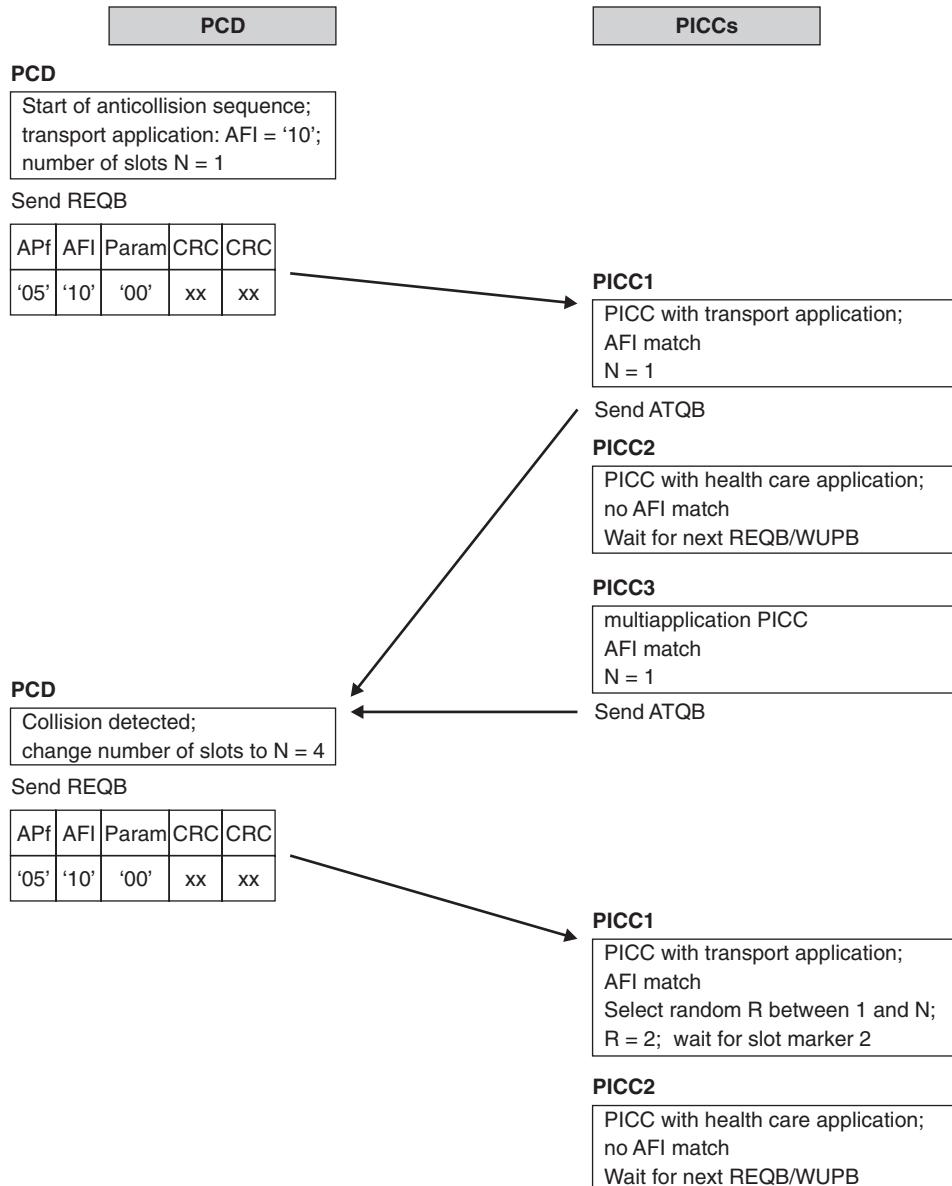


**Figure 10.42** Type B card initialization and anticollision process according to ISO/IEC 14443

#### 10.9.7.1 Protocol activation with Type A cards

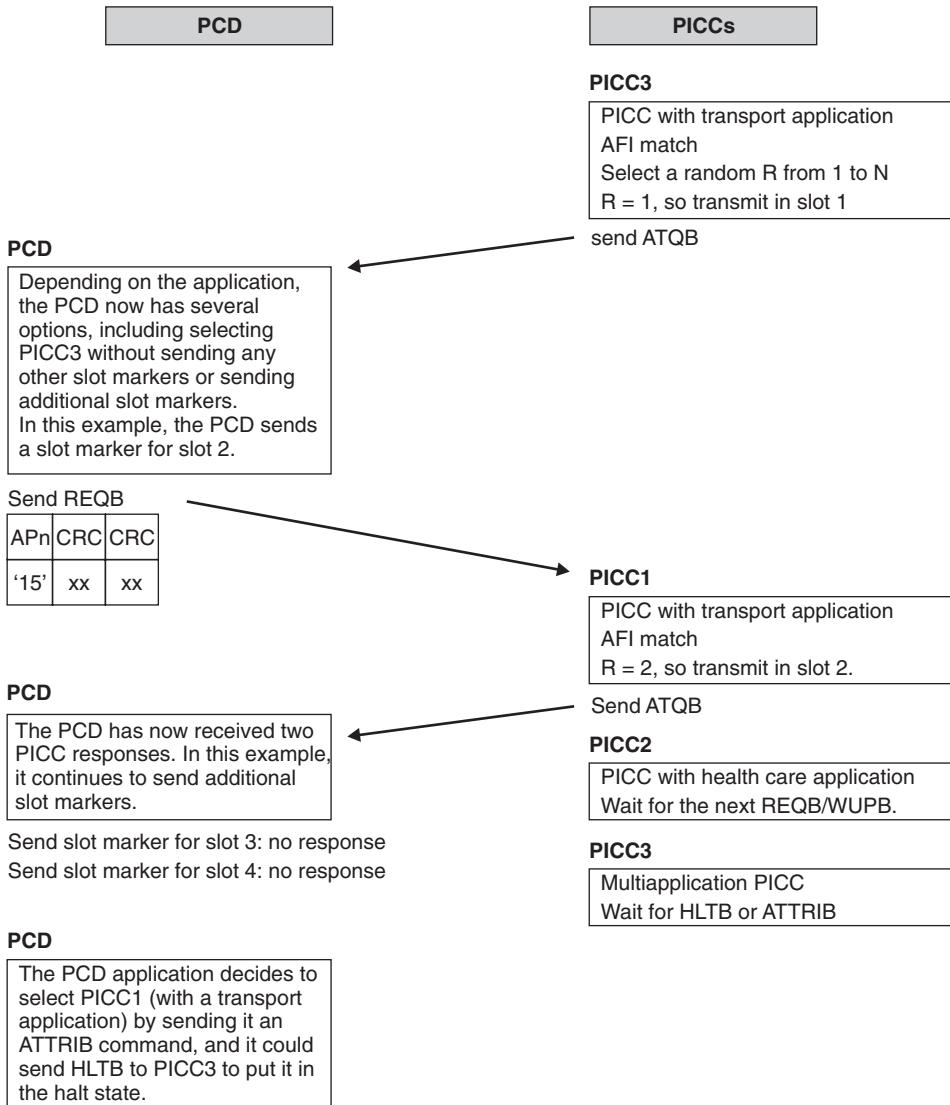
After a Type A PICC has been selected, the terminal executes an activation sequence illustrated by the flow chart in Figure 10.45 on page 333.

From the SAK (Select Acknowledge) sent by the card at the end of the anticollision loop, the terminal can see whether the card supports the standard transmission protocol. If it does



**Figure 10.43** Example anticolision sequence with three Type B cards (part 1 of 2)

not, the terminal sends an HLTA command to the card to place it in the Halt state. If the card supports the ISO/IEC 1443-4 protocol, the terminal sends a RATS (Request for Answer to Select) command to the card. The RATS command and the ATS (Answer to Select) returned by the card are used to exchange data and parameters in order to determine which data transmission options are supported by the card and the terminal. After this, PPS (Protocol and Parameter Selection) can be used to configure the modifiable parameters to make the best use of the capabilities of the card and the terminal. In order to make inexpensive, technically

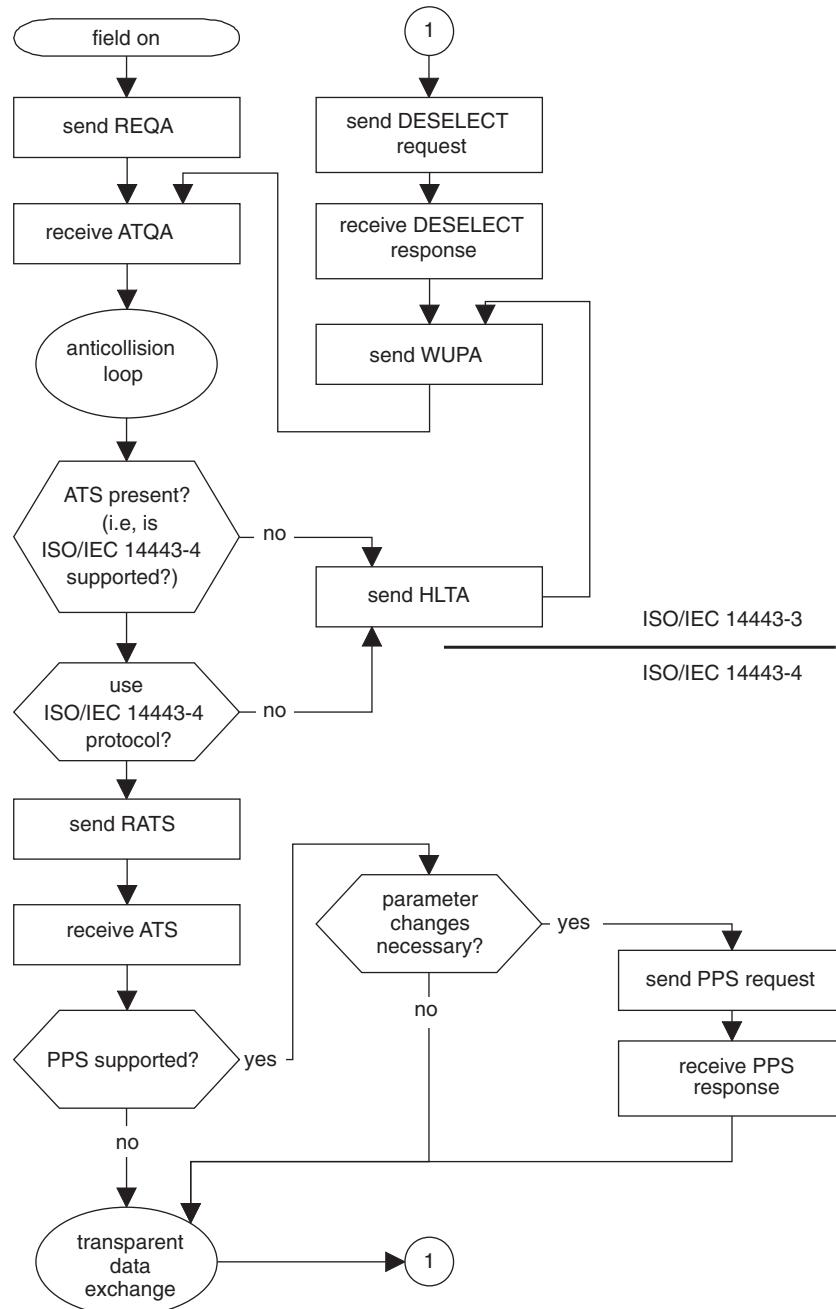


**Figure 10.44** Example anticollision sequence with three Type B cards (part 2 of 2)

simple implementations possible, default values are defined for the modifiable parameters. In the simplest case, the card supports only these default values. The PPS sequence is then unnecessary, and the terminal can immediately start exchanging data using the block protocol after receiving the ATS.

#### Request for Answer to Select (RATS)

The RATS command (see Figure 10.34) contains a parameter byte that specifies the maximum frame size that the terminal can receive (FSDI – Frame Size for Proximity Coupling Device) and the card identifier (CID) assigned to the card for the duration of its active state (see Figures



**Figure 10.45** Activation of a Type A card by a terminal

10.35 and 10.36). Starting with the reception of the RATS command, the card uses this CID as its logical identifier.

### Answer to Select (ATS)

A Type A card responds to a RATS command with an Answer to Select (ATS), which identifies the set of parameters supported by the card. These parameters, as shown in Table 10.37 and Figure 10.46, include:

- maximum frame size
- bit rates in both directions supported by the card
- waiting time between frames
- specific frame guard time
- support for NAD and CID

Default values are specified for cards that do not support parameter value selection. In the simplest case, when only the default values are supported, the ATS consists of only the length byte and the CRC bytes.

### Length byte

The length byte (TL) indicates the number of bytes transmitted in the ATS, including the TL byte but excluding the two CRC bytes. The length of the ATS may not exceed the maximum frame length (FSD) indicated in the RATS command. This means that the maximum value of TL cannot be larger than FSD – 2.

### Format byte

If the length indicated in TL is greater than 1, the format byte (T0) is transmitted next. T0

**Table 10.34** Format of the RATS command

'E0'	Parameter	CRC_A
1 byte	1 byte	2 bytes

**Table 10.35** Format of the RATS parameter byte. CID defines the logical number of the addressed card and has a range of 0 to 14; 15 is reserved for future use. FSDI codes the maximum frame size that the terminal can receive (FSD)

b8	b7	b6	b5	b4	b3	b2	b1
FSDI							CID

**Table 10.36** Coding of bits b8–b5 of the RATS parameter byte

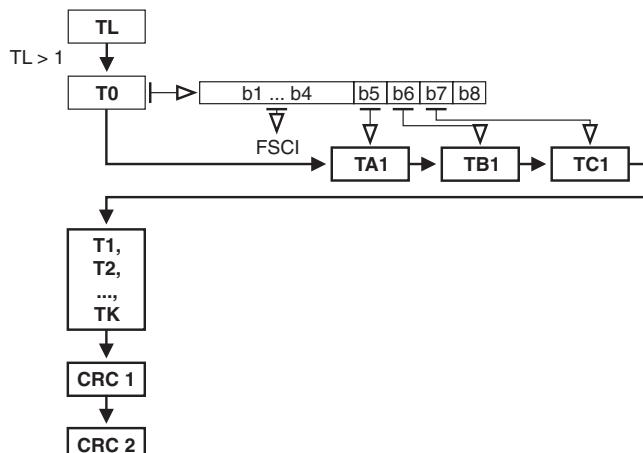
FSDI	0	1	2	3	4	5	6	7	8	9–‘F’
FSD in bytes	16	24	32	40	48	64	96	128	256	>256 RFU

consists of three parts. The most significant bit (b8) has a value of 0 (1 is reserved for future use). Bits b7–b5 indicate the presence of the subsequent interface bytes TA<sub>1</sub>, TB<sub>1</sub>, and TC<sub>1</sub> (see Table 10.38).

The lower nibble (b4–b1) is called the Frame Size for Proximity Card Integer (FSCI). As shown in Table 10.39, it codes FSC (Frame Size for Proximity Card), which is the maximum frame size that can be received by the card. The default value of FSCI is 2, corresponding to a maximum frame size of 32 bytes.

**Table 10.37** Data elements of the ATS and their designations according to ISO/IEC 14443-4

Data element	Designation
TL	Length byte
T0	Format byte
TA <sub>1</sub> , TB <sub>1</sub> , TC <sub>1</sub>	Interface bytes
T <sub>1</sub> , T <sub>2</sub> , … T <sub>k</sub>	Historical bytes
CRC <sub>1</sub> , CRC <sub>2</sub>	Cyclic redundancy check



**Figure 10.46** Format of Answer to Select

**Table 10.38** Coding of the T0 format byte

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	...	...	...	...	...	...	...	Set to 0 (1 is RFU)
0	1	X	X	...	...	...	...	TC1 transmitted
0	X	1	X	...	...	...	...	TB1 transmitted
0	X	X	1	...	...	...	...	TA1 transmitted
0	...	...	...	X	X	X	X	Maximum frame size integer (FSCI)

**Table 10.39** Coding of bits b4–b1 of the FSCI parameter

FSCI	0	1	2	3	4	5	6	7	8	9–‘F’
FSC in bytes	16	24	32	40	48	64	96	128	256	>256 RFU

**Table 10.40** Coding of the TA1 interface byte

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	...	...	...	0	...	...	...	Different values of D are supported for data transmission in both directions
1	...	...	...	0	...	...	...	Only one value of D is supported for data transmission in both directions
...	1	...	...	0	...	...	...	DS = 8 is supported
...	...	1	...	0	...	...	...	DS = 4 is supported
...	...	...	1	0	...	...	...	DS = 2 is supported
...	...	...	...	0	1	...	...	DR = 8 is supported
...	...	...	...	0	...	1	...	DR = 4 is supported
...	...	...	...	0	...	...	1	DR = 2 is supported

### TA1 interface byte

The TA1 interface byte consists of four parts, as indicated in Table 10.40. The most significant bit (b8) indicates whether different divisor values can be used for the two transmission directions. The value of the etu (elementary time unit, equal to one bit interval) is determined by the divisor  $D$  according to the formula

$$1 \text{ etu} = 128/(D \times f_c)$$

The initial value of  $D$  is 1, which yields an etu value of  $128/f_c$ . Bits b7–b5, which are called Divisor Send (DS), indicate the bit rates supported by the card for data transmission from the card to the terminal. Bit b4 is set to 0; 1 is RFU. Bits b3–b1, which are called Divisor Receive (DR), indicate the bit rates supported by the card for data transmission from the terminal to the card. The divisor values are selected by the terminal in the subsequent PPS command.

### TB1 interface byte

The TB1 interface byte transfers parameters that define the frame waiting time and the initial frame guard time. It accordingly consists of two parts, as shown in Table 10.41. The upper nibble (b8–b5) is called the Frame Waiting Time Integer (FWI) and determines the frame waiting time (FWT). The meaning and calculation of the frame waiting time are described in Section 10.9.7.2 on page 339. The lower nibble (b4–b1) is called the Start-up Frame Guard Time Integer (SFGI) and is used to calculate the initial frame guard time (SFGT). The SFGT is the time needed by the card after it sends the ATS before it is ready to receive the next frame. The SFGT value 15 is RFU. The value 0 means that the card does not need any specific SFGT.

**Table 10.41** Format of the TB1 interface byte

b8	b7	b6	b5	b4	b3	b2	b1
FWI						SFGI	

**Table 10.42** Coding of the TC1 interface byte

b8	b7	b6	b5	b4	b3	b2	b1
0	0	0	0	0	0	b2 = 1: card identifier (CID) is supported	b1 = 1: node address (NAD) is supported

SFGI can thus have a value of 1 to 14, and SFGT is calculated using the formula

$$\text{SFGT} = (256 \times 16/f_c) \times 2^{\text{SFGI}}$$

The default value of SFGI is 0, and the maximum value SFGT<sub>MAX</sub> is approximately 4 949 ms. From this, it can be seen that SFGT should be kept as small as possible to achieve a high transmission rate.

### TC1 interface byte

The TC1 interface byte indicates specific protocol options. It consists of two parts. The six most significant bits (b8–b3) are set to 0, as shown in Table 10.42. All other values are RFU. Bits b2 and b1 indicate which optional fields in the Prologue field are supported by the card (see below). The terminal is not obliged to send all fields that are supported by the card, so it may omit one or more of them. However, the terminal is never allowed to send any fields to the card that are not supported by the card. The default value of bit b2 is 1, and the default value of b1 is 0. This means that CID is supported and NAD is not supported.

### Historical bytes

The historical bytes (T1 to Tk) are optional. Their contents are defined in ISO/IEC 7816-4 (see also Section 8.1, ‘Answer to Reset’, on page 203). The maximum possible number of historical bytes can be determined from the maximum length of the ATS.

### Protocol and parameter selection

If the card indicates in the ATS that it supports modifiable parameters, the terminal can use the PPS (Protocol and Parameter Selection) command to change the parameters for the subsequent protocol. If the card does not support any modifiable parameters, it is not required to support the PPS command (see Figure 10.45 on page 333). In this case, the protocol continues with unaltered parameters.

If the card has indicated in the ATS that it supports modifiable parameters, the terminal can now determine whether it wants to utilize the modification options indicated by the card. If it does, it sends a PPS Request command formatted as shown in Table 10.43 on the next page.

**Table 10.43** Format of the PPS Request command

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
PPSS Start byte	PPS0 Parameter 0 indicates whether PPS1 is present	PPS1 Parameter 1 codes DRI and DS1	CRC1	CRC2

**Table 10.44** Coding of Parameter 0. All other values of b8–b6 and b4–b1 are RFU

b8	b7	b6	b5		b4	b3	b2	b1
0	0	0	PPS1 is transmitted if b5 = 1		0	0	0	1

**Table 10.45** Coding of Parameter 1 PPS1

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	0	0	DSI		DRI		DRI and DS1 code the divisor $D$ .

**Table 10.46** Coding of  $D$  by DRI and DS1

DRI/DSI	0	0	0	1	1	0	1	1
$D$		1		2		4		8

### Start byte

The start byte (PPSS) consists of two parts. The upper nibble (b8–b5) is set to ‘D’ to identify the PPS. All other values are RFU. The lower nibble (b4–b1), which is called the Card Identifier (CID), defines the logical number of the addressed card.

### Parameter 0

Parameter 0 indicates whether the optional Parameter 1 byte is present (see Table 10.44).

### Parameter 1

Parameter 1 indicates which of the possible values of DS and DR indicated in the TA1 interface byte have been selected (see Tables 10.45 and 10.46). This determines the transmission rate for subsequent data transmissions.

### Protocol and parameter selection response

The card confirms correct reception of a protocol and parameter selection request by sending a protocol and parameter selection response, which consists of only the PPSS start byte and the two checksum bytes (CRC1 and CRC2). After this, the terminal and the card use only the selected parameters for data transmission.

### Activation frame waiting time

To avoid unnecessarily long waiting times in case of transmission problems, ISO/IEC 14443-4 specifies the maximum allowable interval between when a Type A card receives the end of a frame and when it sends a response. This time is called the activation frame waiting time, and it is set to  $65\,536/f_c$  ( $\approx 4\,833\,\mu s$ ). If a card does not respond within this interval, the terminal knows that there is a communication problem with the card.

### Error detection and correction

In a system using contactless cards, it must be expected that errors will occur more often during data transmission than is usual with contact cards. For example, with contact cards it is relatively uncommon for a card to be removed from a terminal while it is communicating with the terminal. With contactless cards, interruption of communication can occur more often because the cards are free to move within the working range of the terminal during use and can accidentally leave the working range. It is thus important to have methods available that allow such interruptions to be recognized as quickly as possible and enable communication to be resumed in a state that is as well defined as possible.

To avoid having every error result in termination of communication, with the consequent need to start over, ISO/IEC 14443-4 defines rules for error detection and correction during the protocol activation phase of Type A cards. Rather than describe these rules in detail, we refer you to the appropriate section of the standard, where they are presented in a clearly understandable manner.

#### 10.9.7.2 Half-duplex block protocol (ISO/IEC 14443-4)

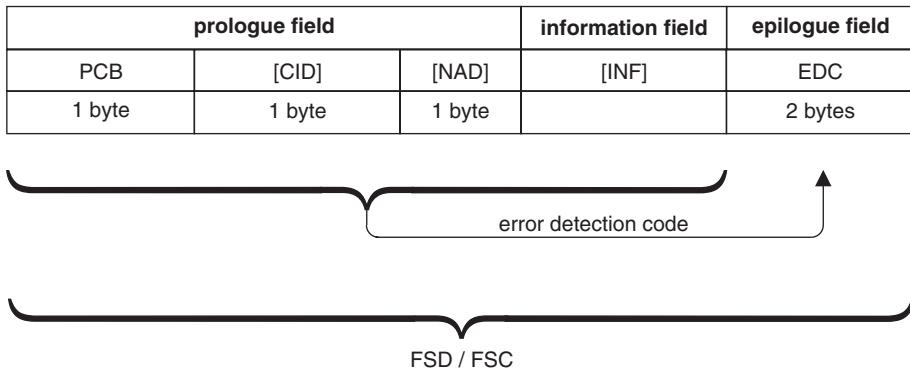
The transmission protocol defined in Part 4 of ISO/IEC 14443-4 addresses the specific requirements of applications that use contactless cards. For instance, it allows a terminal to communicate with several cards in parallel.

Like the T = 1 protocol for contact cards, this protocol supports clean layer separation in accordance with the OSI reference model. Layer separation means that data belonging to higher-level layers is transported fully transparently with respect to lower-level layers. The protocol is based on the frames defined for both types of cards (Type A and Type B) in Part 3 of the standard.

Four layers are distinguished: the physical layer (byte transmission in accordance with ISO/IEC 14443), the data link layer, the session layer (combined with the data link layer to minimize overhead), and the application layer where the commands are processed.

The block-oriented protocol begins after the activation sequence has been completed. The terminal has the initial transmit right. This means that after the activation sequence, the card must wait until it receives a block from the terminal. The card responds to each received block by sending a response block within the defined frame waiting time. After the response block has been sent, the transmit right returns to the terminal and the card changes back to reception mode. Communication continues in this manner, with the transmit right held alternately by the terminal and the card.

The protocol allows several cards to be activated simultaneously by a terminal. In this case, the cards are distinguished by their card identifiers (CIDs). The terminal can switch back and forth between several cards without having to spend time deactivating a card and activating another card each time.



**Figure 10.47** Block structure. The fields in parentheses are optional

As already mentioned, the probability of errors during data transmission is higher in systems with contactless cards than in comparable systems with contact cards. It is thus especially important for communication between the terminal and the card to take place in the shortest possible time. The data rate, which has a default value of 106 kbit/s, is relatively high compared with the default value of 9.6 kbit/s with the T = 0 and T = 1 transmission protocols. Data rates up to 847 kbit/s are possible if suitable parameter values are selected.

### Block structure

Each transmission block consists of a leading prologue field, an optional information field, and a trailing epilogue field (see Figure 10.47). The information field, if present, contains data for the application layer.

There are three different block types: information blocks (I blocks), which are used for the transparent exchange of application-layer data; reception acknowledgment blocks (R blocks), which are used for positive or negative acknowledgment of reception and do not include information fields; and system blocks (S blocks), which are used to exchange protocol control data. Two types of S blocks are defined: an S block for extending the frame waiting time, which has a single-byte information field, and the DESELECT block, which is used to place the card in the Halt state and does not have an information field.

### Prologue field

The Prologue field consists of the Protocol Control byte (PCB), an optional card identifier (CID), and an optional node address (NAD). The coding of the Protocol Control byte is shown in Table 10.47 on the facing page.

### Card identifier (CID)

The card Identifier field (see Table 10.48) is used to identify a particular card. It also contains information about the power available to the card (see Table 10.49). The two most significant bits (b8 and b7) indicate the power level in the card. Bits b6 and b5 are set to 0 and reserved for future use. Bits b4–b1 code the card identifier.

**Table 10.47** Coding of the Protocol Control byte (PCB)

Bit	I block PCB	R block PCB	S block PCB	
b8	0	1	DESELECT	WTX
b7	0	0	1	1
b6	0 (1 is RFU)	1	0	1
b5	1 = chaining	0 = ACK, 1 = NAK	0	1
b4	1 = CID follows	1 = CID follows	1 = CID follows	
b3	1 = NAD follows	0 (no NAD)	0 (no NAD)	
b2	1	1 (0 is RFU)	1 (0 is RFU)	
b1	Block number	Block number	0 (1 is RFU)	

**Table 10.48** Format of the card identifier (CID) field

b8	b7	b6	b5	b4	b3	b2	b1
Power level indication		0; 1 = RFU	0; 1 = RFU		CID		

**Table 10.49** Coding of the power level indication

b8	b7	Meaning
0	0	The card does not support power level indication
0	1	Insufficient power for full functionality
1	0	Sufficient power for full functionality
1	1	More than sufficient power for full functionality

The coding of the CID field is defined in the PPS command for Type A cards and in the ATTRIB command for Type B cards. The following rules apply to evaluating the CID field:

- A card that does not support CID ignores all blocks containing a CID.
- A card that supports CID responds to blocks containing its CID by returning its CID. It ignores blocks that contain other CIDs, and it responds to any block containing a CID of 0 by returning a block with no CID.

These rules enable the terminal to communicate concurrently with several active cards without having to deactivate cards that are not being addressed. Bits b8 and b7 code the power level as shown in Table 10.49.

### Node address (NAD)

The third byte of the Prologue field is called the node address. It is used to establish and use certain types of logical connections between the card and the terminal. Node addresses are used in the same manner as for contact cards. They are defined in ISO/IEC 7816-3 and

**Table 10.50** Format of the INF field of an S(WTX) request

b8	b7	b6	b5	b4	b3	b2	b1
Power level indication		WXTM (1–59; 0 and 60–63 are RFU)					

**Table 10.51** Format of the INF field of an S(WTX) response. The value of WTXM is the same as in the previously received S(WTX) request block

b8	b7	b6	b5	b4	b3	b2	b1
0	0	WTXM					

described in Section 9.3.2, ‘The T = 1 transmission protocol’, on page 260. If a card does not support the optional NAD function, it must ignore all blocks that contain an NAD.

### Information field (INF)

In I blocks, the information field acts as a container for application layer data (see Tables 10.50 and 10.51). The content of this field is transmitted entirely transparently. In S blocks, the information field is used to control frame waiting time extension.

### Epilogue field (EDC)

The epilogue field contains the error detection code of the block. This CRC is calculated as described in the Annex of ISO/IEC 14443 Part 3.

### Frame waiting time (FWT)

In order to achieve defined termination of communication with a nonresponding card in the shortest possible time, a frame waiting time (FWT) is defined. It corresponds to the block waiting time of the T = 1 protocol for contact cards. The frame waiting time is the maximum interval between the end of a frame sent by the terminal and the start of the response frame from the card. If this time expires without a response from the card, the terminal can assume that there is a card malfunction, and the terminal reacquires the transmit right in order to initiate error detection mechanisms. As previously described, the ATS for Type A cards holds the frame waiting time integer (FWI) in the TB1 interface byte. The frame waiting time is calculated from the FWI using the formula

$$\text{FWT} = (256 \times 16/f_c) \times 2^{\text{FWI}}$$

If the optional TB1 interface byte is not sent, the default FWI of 4 applies, which yields an FWT of approximately 4.8 ms.

For Type B cards, FWI is specified in the ATQB. The minimum value of FWT results from an FWI value of 0 and is approximately 302 µs. The maximum value FWT<sub>max</sub>, which results from an FWI value of 14, is approximately 4 949 ms. The selected frame waiting time must be based on the normal command processing time of the card. If the selected value is overly large, the terminal will take longer than necessary to recognize that a card has stopped responding. In practice, cards can respond relatively quickly to most commands, but a few commands, such as

commands involving the computation of a cryptographic algorithm, require significantly more time. To avoid having to use a long frame waiting time for all commands in such cases, there is a mechanism for extending the waiting time. This allows the card to request an extension of the frame waiting time for individual commands.

### Frame waiting time extension

To request an extension of the frame waiting time, the card sends a special S block called S(STX). It receives a corresponding S(WTX) block from the terminal in acknowledgment of this request. The terminal is not allowed to refuse the request.

The frame waiting time extension factor is sent to the terminal in one byte of the information field of the S(WTX) block. The temporary frame waiting time for processing the current command is obtained by multiplying the normal frame waiting time by this factor.

The temporary extended frame waiting time starts at the end of the S(WTX) response sent by the terminal. It is calculated using the formula

$$\text{FWT}_{\text{temp}} = \text{WTXM} \times \text{FWT}$$

If the formula yields a result greater than the maximum frame waiting time  $\text{FWT}_{\text{MAX}}$  ( $\approx 4\,949$  ms), the maximum value is used instead of the calculated value.

### Block numbering

In I and R blocks, the block number is indicated in bit b1 of the Protocol byte. The purpose of the block number is to enable the recipient to request retransmission of blocks received with errors, since successive I blocks will have different numbers. As the block number is only one bit, it alternates between 0 and 1. The numbering procedure for the block number is as follows.

The terminal first sets the block number to 0 for each activated card. When the terminal receives an I block or an R(ACK) block with a block number that matches the current block number, it changes the block number for the associated card before sending the next block to the card.

The card sets the block number to 1 when it is activated. When it has received a block, it changes the block number before sending a block. If the card receives an R(ACK) block with a block number that does not match its current block number, it changes the block number before sending the next block. The block number is not changed when an R(NAK) block is received.

### Block chaining

As with the T = 1 block-oriented protocol (ISO/IEC 7816-3) for contact cards, the block profile specified in Part 3 of ISO/IEC 14443 allows block chaining. The block chaining function allows either of the communicating parties to transmit large data blocks that are too big to fit in a single frame by distributing the data over several I blocks and sending them in succession. Each of these chained I blocks has a length that is less than or equal to the frame length specified by FSC or FSD.

When block chaining is used, the sender sets the chaining bit in the protocol control byte (PCD) of the first chained block. This indicates to the recipient that block chaining is being used and the subsequent block contains chained data.

If the recipient receives the first chained I block correctly, it indicates this, as well as its readiness to receive the next block, by returning an R block with the same block number as the received block. The next block can then be sent.

This back-and-forth exchange of I and R blocks continues until the sender transmits an I block in which the chaining bit is not set. After receiving this block, the recipient knows that all of the application-layer data has been received, and it can process this data block and send the corresponding response.

#### **10.9.7.3 Deactivating a card**

When data transmission between the terminal and the card is completed, the terminal places the card in the Halt state by sending it a DESELECT command in an S block. The card responds to this command with an S(DESELECT) response block and enters the Halt state. The card identifier assigned to the card and sent to it in the RATS command when the card was activated is no longer valid after the card enters the Halt state. If the card is activated again by a wake-up command, it is assigned a new card identifier.

#### **10.9.7.4 Error handling**

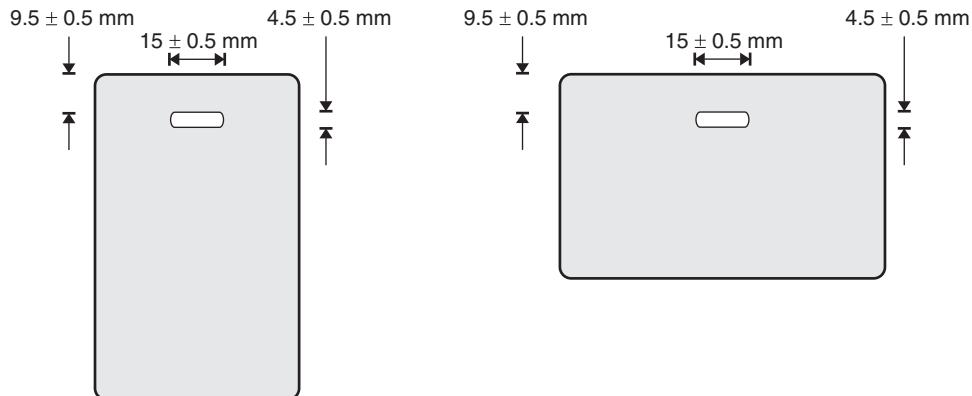
The block transmission protocol has error detection mechanisms similar to those of the T = 1 protocol, which enable resynchronization in several stages in case of transmission errors. The exact rules for the protocol sequence can be found in Part 4 of ISO/IEC 14443. Extensive examples of error-free protocol sequences and error handling are also provided in the Annex to the standard.

### **10.10 VICINITY INTEGRATED CIRCUIT CARDS (ISO/IEC 15693)**

ISO/IEC 15693, ‘Identification cards – Contactless integrated circuit(s) cards – Vicinity cards’, describes the properties and operating modes of contactless smart cards with a range up to 1 meter. This type of card is preferred for applications such as access control, since its operational range is large enough that it is not necessary for the user to present the card to the terminal manually. Instead, it can remain in the user’s pocket, purse, or other location.

As the maximum allowable magnetic field strength is restricted, the large range compared with proximity integrated circuit cards (PICCs) compliant with the ISO/IEC 14443 standard can only be achieved by severely reducing the minimum activation field strength of vicinity cards (VICCs). However, this also reduces the power available to the VICC, which means that it is not possible to use a microcontroller. Instead, only simple memory ICs with relatively simple security logic (a state machine) can be used.

There is also a security issue associated with the larger working range: it is difficult for users to recognize whether they (and their cards) are within the working range of a terminal. It is thus possible for a terminal to establish a connection to a card without the explicit desire or knowledge of the cardholder. Whether such undesired and unnoticed card accesses pose security risks depends on the application. In certain cases, any security problems that arise may be remedied at the application level, although this affects how the card is used (such as PIN entry by the user, which distinctly increases the transaction time).



**Figure 10.48** Locations and dimensions of slots for attaching a strap to a contactless card

Due to their restricted functionality, vicinity cards are normally not used together with other types of cards (such as proximity cards and contact cards) in open applications, for which reason we describe the most important properties and operating principles of vicinity cards only in broad terms here. Readers interested in more details are advised to consult the ISO/IEC 15693 standard and the associated test standard, ISO/IEC 15373 Part 7.

ISO/IEC 15693 consists three parts: ‘Physical characteristics’, ‘Air interface and initialization’, and ‘Anticollision and transmission protocol’. Part 1 of the ISO/IEC 15693 standard defines the physical properties of vicinity cards to the extent that they are not already specified in ISO/IEC 7810. Among other things, it defines maximum tolerable levels of ultraviolet radiation, X-rays, bending and twisting stresses, and electric and magnetic fields. These values are defined such that a VICC will not suffer any damage in normal use under the ambient conditions of every day life. They are essentially the same as the values specified for proximity cards in Part 1 of ISO/IEC 14443.

As already mentioned, access control to secure areas is one of the main applications for vicinity cards. In many cases, these cards are also used as identification cards (such as company ID cards) and must be worn in a visible location. To simplify this, the regions where slots for a strap can be punched in the card are specified in an annex to the ISO/IEC 15693-1 standard, as shown in Figure 10.48. It is advisable to keep ICs and antenna outside these regions in order to avoid damage to these components when the cards are punched.

### Contactless interface for power and data transmission

The properties and values of the electromagnetic fields used to supply power to vicinity cards and support bidirectional communication with the cards are described in Part 2 of ISO/IEC 15693. As with proximity cards, power is provided by an alternating magnetic field generated by the terminal (which is called a proximity coupling device (PCD) in the standard) with a frequency  $f_c$  of 13.56 MHz. For this purpose, the card has an antenna coil with several turns (around 3 to 6). The minimum and maximum magnetic field strengths for card operation are specified as  $H_{min} = 150 \text{ mA/m}$  and  $H_{max} = 5 \text{ A/m}$

Compared with the values for proximity cards, the minimum magnetic field strength (which is called the activation field strength) is a factor of ten lower. This enables the increased range of approximately 1 m to be obtained.

### Data transmission from the terminal to the card

The standard defines several modulation and coding methods. This range of options makes it easier to fulfill the requirements of differing national and international regulations and various applications, although this comes at the price of more complicated integrated circuits in the cards because they must support all of the modulation and coding methods. As with proximity cards, amplitude shift keying (ASK) is used with a modulation index of 10 % or 100 %. The timing characteristics of the pause are specified precisely in the standard and are similar to those of Type A and Type B proximity cards.

Two modulation options are available, independent of the modulation index: a 1 of 256 scheme and a 1 of 4 scheme. Pulse position modulation (PPM) is used in both cases, which means that the value of the transmitted character is determined by the relative timing of the modulation pulse. Not all possible combinations of modulation and coding methods are equally sensible. For example, 10 % ASK with 1 of 256 coding is preferably used for applications with a large working range because this combination achieves optimum utilization of the allowable magnetic field strength for supplying power to the card. The dialog between the terminal (VCD) and the card (VICC) begins with the following steps:

- the VICC is activated by the alternating magnetic field of the VCD;
- the VICC waits for a command from the VCD;
- the VCD sends a command;
- the VICC sends a response.

To simplify synchronization, data is transmitted in frames that are bounded by specific start of frame (SOF) and end of frame (EOF) markers. The terminal indicates the selected coding method to the card in the start of frame marker of the first frame. The transmission time of each byte is approximately 75.72 µs. The bit rate is 26.48 kbit/s ( $f_c/512$ ).

### Data transmission from the card to the terminal

Load modulation with a subcarrier is used for data transmission from the card to the terminal. The subcarrier is generated by load switching in the card. The data is Manchester coded, with a choice of ASK or FSK modulation. Two data rates are specified for each modulation method, resulting in four options for data transmission, all of which must be supported by the vicinity card. This range of data transmission options facilitates adaptation to different environmental requirements or application requirements. For example, the low data rate can be selected to achieve a larger working range. The selected option is specified by the terminal in the header of the transmission protocol.

### Subcarrier frequency and data rate

The parameters of the subcarrier for ASK or FSK modulation are listed in Table 10.52 on the facing page.

### Frame

For synchronization, frames bounded by start of frame (SOF) and end of frame (EOF) markers are also used for data transmission from the vicinity card to the terminal. The SOF and EOF parameters violate the coding rules for data transmission, which allows the start and end of each frame to be recognized without decoding the data stream.

**Table 10.52** Subcarrier parameters for ASK or FSK modulation

	ASK	FSK
Subcarrier frequency $f_{s1}$	423.75 kHz ( $f_c/32$ )	423.75 kHz ( $f_c/32$ )
Subcarrier frequency $f_{s2}$	—	484.28 kHz ( $f_c/28$ )
Low data rate	6.62 kbit/s ( $f_c/2\ 048$ )	6.67 kbit/s ( $f_c/2\ 032$ )
High data rate	28.48 kbit/s ( $f_c/512$ )	26.69 kbit/s ( $f_c/508$ )

**Table 10.53** UID format. The IC manufacturer code is specified in Amendment 1 of ISO/IEC 7816-6. The unique IC serial number is programmed in the chip by the manufacturer

64	57	56	49	48	1
'EO'		IC manufacturer code		IC serial number	

### Transmission protocol

After the vicinity card is activated by the alternating magnetic field of the terminal, it waits for a command from the terminal. The transmission protocol, which defines the mechanisms for exchanging commands and data between the terminal and the card, is described in detail in Part 3 of ISO/IEC 15693. A brief summary should suffice here.

As vicinity cards cannot contain microcontrollers due to the low available power, the protocol used with vicinity cards is significantly simpler than the protocol for contact cards or proximity cards. The standard specifies only two commands as mandatory; they are necessary for the anticollision sequence. All other commands are optional.

The protocol imposes a strict master–slave relationship between the terminal and the card. This means that the card cannot start to send data until it has correctly received a command from the terminal. Each command from the terminal and each response from the card is transmitted in a frame bounded by SOF and EOF markers as previously described. Each command consists of the following fields: SOF, flags, parameters, data, CRC, and EOF.

The terminal uses the Flags field to inform the card of the modulation method (ASK or FSK) and data rate that are used, among other things. In the response, the card indicates in the Flags field whether an error occurred in the processing of the received command.

The terminal can use an anticollision and selection mechanism to select one of several vicinity cards and mute the other cards in order to prevent collisions during data transmission. To differentiate the individual cards, each card is assigned a unique identifier (UID), which contains 64 bits and is described in Table 10.53.

Integrated circuits in vicinity cards usually have hardwired security logic, so their functionality is determined by the manufacturer and is not freely programmable. The standard allows manufacturers to define their own commands. If proprietary commands are used, the command frame has an additional field that holds the manufacturer code so that the manufacturer command can be uniquely specified. The standard and/or the IC manufacturer's data sheet should be consulted for details of the optional commands and manufacturer-specific commands.

## 10.11 NEAR FIELD COMMUNICATION (NFC)

Sony and NXP (formerly Philips) have been attempting to establish their Near Field Communication (NFC) wireless standard since 2002 – Sony with its FeliCa technology and NXP with its Mifare technology. As the name suggests, this is a form of near field technology consisting of a combination of contactless identification according to the ISO/IEC 14443 standard and wireless connection technology. The aim of NFC is to utilize ISO/IEC 14443 contactless technology operating at 13.56 MHz to simplify networking using Bluetooth, WLAN, or mobile telecommunication equipment by having NFC-capable devices located up to a few centimeters apart establish data links and identify themselves automatically without requiring any action on the part of the user.

NFC is a further extension of widespread networking. As the operational range is limited to a few centimeters, the technology largely prevents unintentional data transfer between equipment and is thus suitable for use in many applications with relatively simple security mechanisms.

Contactless proximity cards are only one component of NFC technology. Consequently, a full description of NFC technology is far beyond the scope of this book and would vastly increase the size of the book. Accordingly, readers interested in this technology are referred to the current specifications published by the NFC Forum [NFC Forum]. However, there is an overlap between the use of proximity cards and NFC devices in applications in the payment sector, so we present a brief summary of NFC technology here to give the reader an impression of the operating principles and potential applications of this technology.

It can be expected that NFC devices will partially or fully supplant contactless cards in some applications. In these cases, an NFC device can assume all of the functions of a contactless payment card. NFC devices such as mobile telephones can also be regarded as contactless smart cards with a different form factor. This is possible because contactless terminals (unlike terminals for contact cards or magnetic-stripe cards) are not dependent on the format of ID-1 cards, but instead can exchange data with any device that supports the ISO/IEC 14443 interface and is located within range of the terminal. This conformity with the most important standard for contactless smart cards makes NFC devices interoperable with millions of contactless smart cards and terminals already installed worldwide.

### 10.11.1 State of standardization

The nonprofit NFC Forum [NFC Forum] was founded by Philips, Nokia and Sony in 2004. Its objective is to promote the worldwide use of NFC technology. NFC Forum now has more than 130 members, who represent manufacturing companies, application developers, merchants, financial service providers, transport companies, and nonprofit organizations.

To increase the use of NFC technology in consumer electronics, mobile equipment and PCs, NFC Forum has generated a system architecture for NFC applications and solutions that includes security aspects. For this purpose, in a relatively short time NFC Forum generated specifications, in part based on standards such as ISO/IEC 14443 as well as ECMA and ETSI standards, that encompass the NFC device architecture and data exchange protocols. Some of these standards were submitted to ISO/IEC as proposed standards for the fast track procedure and were published in 2004 with the title ‘Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol’.

In 2006, NFC Forum published specifications for the technology architecture, including data formats (NFC Data Exchange Format – NDEF), and in 2007 it published specifications for four different tag types supported by all NFC devices. The four tag types are based on ISO/IEC 14443 (Type A and Type B) and the FeliCa specification. Millions of tags compliant with these formats, in the form of Mifare products from NXP and FeliCa products from Sony, are used as contactless cards for applications such as ticketing in local public transport systems and access control. Due to the interoperability of the proximity card standard and the NFC standard, these applications were immediately open to NFC, which means that the cards used in these applications can be read by NFC devices and the installed terminals can communicate with NFC tags.

Type 1 and Type 2 tags correspond to the ISO/IEC 14443 Type A standard. They have only a small amount of memory (1–2 kbit) and a low data rate (106 kbit/s). These tags are very inexpensive and are especially suitable for large-scale applications with relatively low security requirements (such as read and/or write protection).

Type 3 corresponds to the FeliCa specification and currently has 2 KB of memory and a data rate of 212 kbit/s. It has more security functions and is suitable for more complex applications.

Type 4 is fully compatible with ISO/IEC 14443 (Type A and Type B) and thus fully corresponds to contactless proximity cards. Integrated circuits for this type are available from several manufacturers. The memory capacity is presently as much as 64 KB, and the data rate extends from 106 to 424 kbit/s. As these ICs incorporate a microprocessor and sophisticated security logic, they are well suited to multipurpose applications with the highest security requirements. However, these excellent properties require a relatively large chip area, which is reflected in the component price.

## 10.11.2 NFC protocol

The NFC protocol is similar to the protocol for proximity cards, with the difference that both NFC parties can assume the role of master. Communication is half-duplex, which means that the communicating devices can either send or receive at any given time, but not both. To avoid collisions, each device must first verify that no other device is already transmitting on the carrier frequency before it starts to transmit (listen before talk). The NFC protocol differentiates between the initiator and the recipient (target) of communication, with each device fundamentally being able to assume the role of initiator or target. However, the roles cannot be reversed after a link has been established. As the name suggests, the initiator is the device that initiates and controls the data exchange (the master). The target is the device that responds to commands from the initiator (the slave). For comparison, in the proximity card protocol the terminal assumes the role of master and the card assumes the role of slave.

The application (the initiator) selects the initial data rate from a choice of 106, 212, or 424 kbit/s. During the subsequent communication session, the data rate can be adjusted to suit the requirements of the application and the capabilities of the devices. NFC supports the various modulation and coding methods of proximity cards, with the target determining the actual speed and protocol parameters.

The NFC protocol distinguishes between two communication modes: active and passive. In the active communication mode, each of the NFC devices generates its own high-frequency field at the carrier frequency for transmitting data. In the passive communication mode, only the initiator generates a high-frequency field at the carrier frequency, while the target uses load

modulation for data transfer. The passive mode is especially important for battery-powered equipment, such as mobile telephones and PDAs.

In active mode, NFC devices can exchange all types of data at distances up to 20 cm. Two use cases are distinguished in passive mode. In case 1, the NFC device assumes the role of a terminal (PCD) for proximity cards compliant with ISO/IEC 14443. In case 2, the NFC device assumes the passive role (it does not generate a high-frequency field) and acts like a proximity card.

### **10.11.3 NFC applications**

The potential applications of NFC technology can be classified into three categories, which impose different requirements on the NFC devices concerned.

#### ***10.11.3.1 Rapid access to information regarding services***

In this application scenario, the user holds a personal NFC device (such as a mobile telephone) next to an NFC tag and receives the information stored in the tag in a fraction of a second. This information may be a short message, a web address, a telephone number, or other brief information that facilitates access to the offered service.

Smart posters are one example of this application. A smart poster has an embedded NFC tag that provides information on the product concerned. For example, the user may obtain the URL of a website where additional information on the product is available or where tickets can be purchased.

#### ***10.11.3.2 Peer-to-peer information exchange***

In this application scenario, NFC is used to establish communication between two devices so they can exchange data. If the volume of data is relatively small (a few kilobytes), the NFC link can also be used for the data transfer. However, with larger data volumes it is better to use NFC technology to establish a different sort of wireless connection, such as Bluetooth or WiFi. Anyone who has ever had problems with setting up a Bluetooth or WiFi connection will appreciate the convenience of simply holding the two devices close to each other and having the connection established automatically.

#### ***10.11.3.3 Mobile payment***

Mobile NFC devices such as mobile telephones can support payment transactions that are compatible with millions of installed contactless terminals for ticketing or making purchases. In this scenario, the NFC device assumes the role of the contactless payment card. In addition, the user has a list of all payment transactions constantly available on the mobile device. The NFC device can also be used as a reader for other contactless cards. Of course, payment applications are subject to strict security requirements that require an approved security component in the device.

### 10.11.3.4 Secure NFC

From the very start, one of the main target applications for NFC technology has been contactless payment using mobile devices. With the increasingly widespread use of contactless credit card terminals compliant with ISO/IEC 14443, especially in Asia and the USA, a rapidly growing market for NFC-capable mobile equipment has opened up. As already mentioned, a security component in the NFC-capable mobile device is a prerequisite for mobile payment. Three preferred options for integrating a security component in a mobile telephone or PDA are currently under discussion.

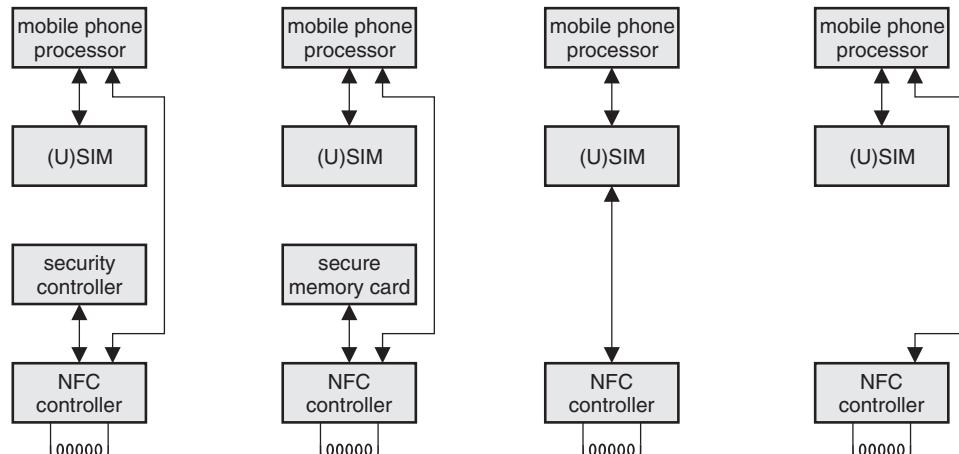
The first option is to use a security chip permanently integrated in the mobile equipment as the security component. One solution here is to use a smart card microcontroller already approved for use in payment applications to host the payment application. This achieves the same security level as with a corresponding smart card. One problem with this solution is maintaining the interchangeability of the mobile equipment, since the application is permanently tied to the equipment.

The second option is to use an external secure memory card (such as a secure microSD card). In this case, the security chip is housed in the memory card along with the flash memory. With this solution, the user can easily transfer the payment application to a different mobile device by swapping the memory card.

In the third option, the SIM or USIM card contains the payment application. With this approach, the application can be hosted directly in the SIM microcontroller or in a supplementary security chip in the SIM or USIM module. A supplementary security chip has the advantage that only this chip needs to be certified.

The question of which approach will prevail essentially depends on the business model ultimately agreed on by the parties involved, including the network operator, the banks, the card issuer, the equipment manufacturer, and the merchants. The diverse interests of the parties involved must be reflected in a system and security architecture that is acceptable to all parties.

The Global Platform consortium [GP] is developing a specification for the initialization and secure management of several types of security components and the SIM, under the condition



**Figure 10.49** Several options for integrating a security component for NFC in a mobile device

that this solution must be independent of the integration of the security component in the individual equipment and should support several mobile telecommunication architectures (GSM, UMTS, and CDMA).

In any case, it is necessary to devise a solution for secure loading and personalization of the payment application in the security component. Unlike the activation of ordinary applications, this process cannot be performed by the network operator. As with the personalization of bank cards, this process must be performed by a trustworthy entity that is certified for payment systems. The first service providers that can offer personalization and activation over the air (OTA) and enjoy the trust of the financial service providers as well as the network operators have already appeared.

## 10.12 FELICA

FeliCa is a contactless card system developed by Sony that is widely used in Japan and Asia in electronic ticketing and electronic purse systems. Several types of memory chips with integrated security logic are available as standard products, along with corresponding terminals. The technology is similar to that of the ISO/IEC 14443 standard. FeliCa technology was proposed to ISO/IEC for inclusion in the ISO/IEC 14443 standard as Type C, but no consensus could be reached on this due to the desire to avoid yet another variant in this standard after types A and B had already been accepted.

FeliCa cards operate at a frequency of 13.56 MHz with a data rate of 212 kbit/s using Manchester coding. FeliCa is compatible with the ISO/IEC 18092 standard (Near Field Communication).

## 10.13 MIFARE

Mifare is a contactless card system developed by NXP (formerly Philips) that is used worldwide. The Mifare system is compatible with ISO/IEC 14443 Type A (Proximity Integrated Circuit(s) Cards) and ISO/IEC 18092 (Near Field Communication). Several types of memory chips with integrated security logic and microcontroller chips (including dual interface chips) are available as standard products, along with corresponding terminals.

# 11

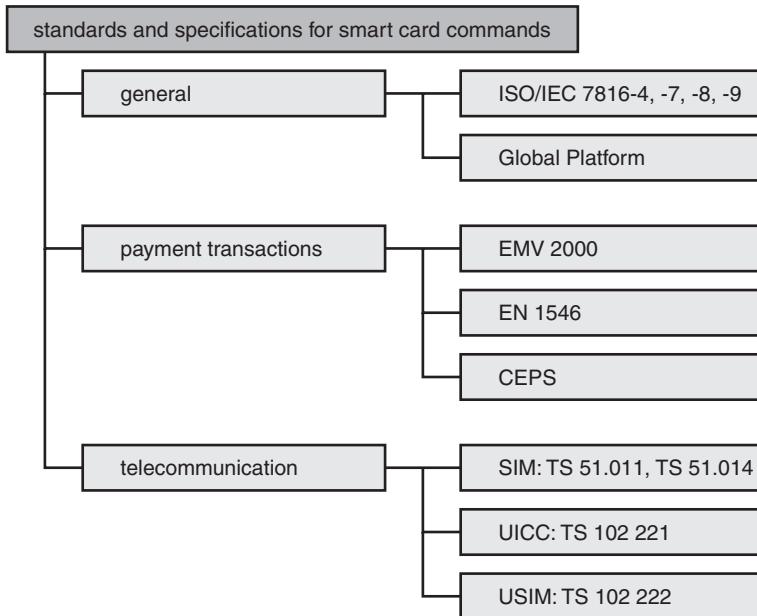
## Smart Card Commands

Communication between a terminal and a smart card is always based on the master–slave principle. This means that the terminal (as the master) sends a command to the card, which as the slave immediately processes the command, generates a response, and returns the response to the terminal. The card never sends data without first having received a corresponding command from the terminal. Even the ATR is no exception to this rule, since it is a response to the reset signal, which is also a sort of command sent to the card.

Communication always employs a transmission protocol, such as  $T = 0$  or  $T = 1$ . These relatively uncomplicated protocols meet the specific requirements of smart card applications and are optimized for this purpose. Deviations from these precisely specified protocols are not permitted inside application processes. The transmission protocols allow data to be sent to the card and received from the card in a manner that is completely transparent to the transport layer. The data is embedded in a sort of container called an application protocol data unit (APDU). APDUs sent by the terminal to the card are the commands to the card. The terminal also receives the responses to its commands in APDUs embedded in the transmission protocol. There are a large number of commands based on this mechanism, and they initiate specific activities in the card. The simplest examples are read and write commands for smart card files.

In smart card applications, the card is used as a data storage medium, an authorization medium, or both at the same time. This has led to the generation of command sets optimized for these applications and transmission protocols, which are used only in the smart card realm. Due to the severely limited memory capacity of smart cards and cost-based market pressures that allow only moderate increases in memory capacity, command sets are usually tailored to specific applications. All commands that are not needed in a given application are relentlessly removed during program optimization. Only a few operating systems support extensive command sets that have not been reduced to the commands needed for a particular application.

A diversification effect can also be seen with smart card command sets, as is typical with new technologies. Every company active in this area tries to create its own commands that are tailored to the needs of its operating system or anticipated application. This often arises from necessity, since functionally equivalent commands may not exist in the standards. Companies may also deliberately attempt to improve their competitive positions or deny their competitors access to a particular application area by using commands that are highly optimized in terms of card functions and memory usage.



**Figure 11.1** The most important standards and specifications for smart card commands

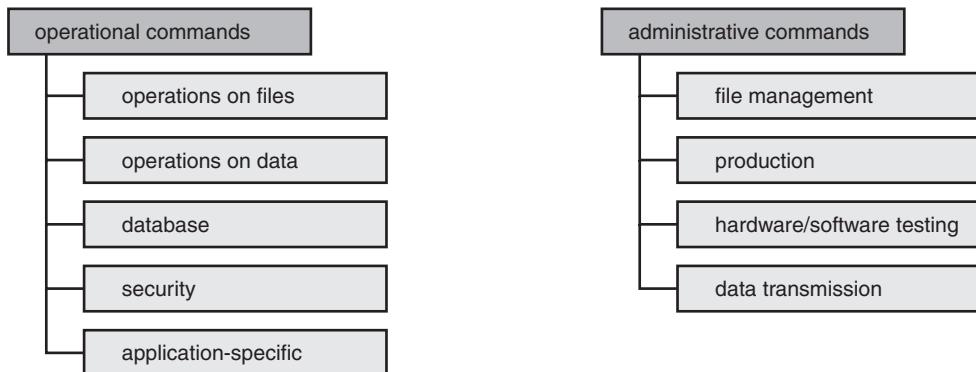
In any case, a decision to use commands based on existing standards always means choosing an open, more easily expandable and proven system, which may later allow additional functions to be incorporated in individual cards. On the other hand, there are many examples of systems in which the use of smart cards was only made possible by using highly optimized special commands.

There are now more than twenty international standards or relatively stable draft standards in which typical smart card commands are specified. They define more than 150 commands, along with their associated processes. To a large extent, the defined commands are mutually compatible in terms of coding and functionality. Figure 11.1 shows a classification of the most important standards for smart card commands.

Most of the commands currently used with smart cards are defined in the ISO/IEC 7816-4 standard, which is a general international standard. It is not dedicated to any particular area, such as telecommunication or financial transactions, but instead attempts to address all smart card applications. The commands defined in ISO/IEC 7816-4 are complemented by additional, specialized members of this family of standards: ISO/IEC 7816-7 defines commands for querying and managing smart card database structures based on structured query language (SQL); ISO/IEC 7816-8 contains commands for configuring and executing cryptographic functions; ISO/IEC 7816-9 adds file management commands to the command set; and ISO/IEC 7816-13 defines commands for managing code-based applications.

However, the de facto standard for application management is the Global Platform Specification [GP], which provides detailed descriptions of a broad range of mechanisms for managing multiple code-based applications and their data in multiapplication smart cards. It is the fundamental specification for managing applets in Java cards.

There is no single international standard or family of standards that covers all applications in the financial transactions sector, but there are several specialized international standards



**Figure 11.2** Typical smart card commands: (left) commands primarily used in the operational phase; (right) commands primarily used in the administrative phase (before and after the card is used)

and industry standards. One of these is the EMV specification, whose name comes from the initial letters of Europay, MasterCard and Visa, the three initiators of this specification. Due to the strong market position of the companies behind it, it has achieved the status of a standard reference for all related smart card operating systems, where it has the same level of significance as the ISO/IEC 7816 family of standards.

The command set for multisector electronic purse systems is defined in the CEN EN 1546 standard. This European standard defines all commands necessary for an electronic purse, along with the associated processes. The functions and processes defined in this standard appear in adapted form in nearly all electronic purse systems. An example of this is the Common Electronic Purse Specification (CEPS), which has achieved considerable importance in the electronic purse sector.

In the telecommunication sector, the TS 51.011 (formerly GSM 11.11) and TS 51.014 (formerly GSM 11.14) standards provide the basis for SIM cards, while the TS 102 221 and TS 102 222 standards specify the basic commands for USIM cards. Due to the virtually global use of smart cards in the telecommunication sector, these standards represent a de facto worldwide standard for the commands of smart card operating systems.

The commands in the standards and specifications described above can be classified according to their functionality as shown in Figure 11.2. However, it must be remembered that only subsets of these commands are implemented in real-life smart card operating systems. Depending on the producer of the operating system, more or less significant deviations from the functionality and coding described in this chapter may be encountered. However, the basic functions described here are in principle present in all operating systems. Of course, the functionality may be severely limited due to memory capacity or cost constraints. In the process of planning a new application, it is essential to request the exact specification of the coding and functions of each command from the producer of the operating system to be used.

The following sections describe the most important and most widely used smart card commands. This selection is based on the following standards and specifications: ISO/IEC 7816-4/-7/-8/-9, EMV, TS 51.011, TS 51.014, TS 102 221, TS 102 222, and EN 1546-3.

Depending on the intended application area and operating system, certain classes of commands may be supported more extensively than others. For example, issuers of multiapplication

cards certainly want to ensure that additional applications can be installed in the card after it has been personalized. Cards intended to be used for cryptographic applications, assuming they have sufficient memory capacity, will contain the full spectrum of cryptographic commands with their various algorithms. Every application area requires a different selection of commands from the various classes.

In each of the following descriptions, the standard or specification in which the command is defined is indicated in order to maintain an overview. If no reference is stated, the command in question is used internally by smart card manufacturers and cannot be assigned to any of the above-mentioned standards. Some of these commands are nonetheless very useful and will probably be incorporated in a standard in the future. They are thus listed here with descriptions of their basic functionality. In the interest of readability, in this chapter we have omitted any description of the coding of typical smart card commands. The internal processes of seven typical commands supported by nearly all smart card operating systems are described in more detail in the pseudocode of the Small-OS operating system (see Section 13.17, ‘The Small-OS Smart Card Operating System’, on page 521).

For any given application, smart card commands can be classified into operational commands, which are commands needed for normal use, and administrative commands, which are commands needed for managing the smart cards and the application. For reasons of interoperability, operational commands are usually specified in full detail in the standards. Administrative commands are sometimes specific to particular operating systems and are not necessarily defined by standards.

The response shown in the description of each command is always the response received by the terminal in case of successful execution. If execution is not successful, which means that an operation was forbidden or an error occurred in the card, the terminal receives only a two-byte return code. Some of the described commands have supplementary parameters for selecting additional functions. Sometimes these command variants are found only in the standard, rather than in actual operating systems, because they are too complicated or have no practical use. Consequently, this chapter does not list or explain every variant defined in the standards, since our aim is to provide a clear basis for understanding functions commonly used in practice. Each command description is based on the standard that defines the largest number of functional options for that command.

## 11.1 FILE SELECTION COMMANDS

Without exception, file management in all modern smart card operating systems is object-oriented. Among other things, this means that an object (in this case a file) must be selected before any operation can be performed on it. Only then does the operating system know which file is meant, and all subsequent file-specific commands apply to this file alone. Of course, the operating system must also check the access conditions of the file to determine whether the command in question is actually allowed or possible.

The master file (MF) is always selected implicitly after a card reset, so it does not have to be selected specifically. Other files are subsequently selected by using the SELECT command. A file is addressed by its two-byte file identifier (FID) or, in the case of directory file (DFs), a DF name with a length of one to sixteen bytes. A DF name can contain an internationally unique application identifier (AID) with a length of five to sixteen bytes. It is possible to supply only part the AID, omitting the less significant bytes (those to the right). Another parameter causes

**Table 11.1** The functionality of SELECT according to ISO/IEC 7816-4

SELECT	
Command	<ul style="list-style-type: none"> <li>• FID (if EF, DF or MF)           <ul style="list-style-type: none"> <li><i>or</i></li> <li>path to file from currently selected DF</li> <li><i>or</i></li> <li>path to file from MF</li> <li><i>or</i></li> <li>• <i>switch</i>: select next higher-level DF</li> <li><i>or</i></li> <li>first, last, next, or previous DF (if a partial AID is supplied)</li> <li>• <i>switch</i>: return information about the selected file</li> <li>• <i>if DF</i>: DF name</li> </ul> </li> </ul>
Response	<ul style="list-style-type: none"> <li>• information about the selected file (if selected by the switch)</li> <li>• return code</li> </ul>

the card to select the first, last, next, or previous DF relative to the DF identified by abbreviated AID.<sup>1</sup>

For compatibility with an earlier command definition, the TS 51.011 standard only supports file selection using the two-byte FID. The ISO command set, by contrast, also supports file selection using the path name of the file. The path can be either relative, in which case the file is selected from the currently selected DF, or absolute, in which case the file is selected from the MF.

Only successful selection of a new file causes the previously selected file to be deselected. If the selection process cannot be completed, for example because the requested file does not exist, the previous selection remains in force. This ensures that a file is always selected, even in the event of an error.

After successful file selection, the terminal may request information about the new current file. This request, including the desired number of data items, is sent to the card via the SELECT command. The exact contents of these data items are defined in the applicable standard. The data items returned by the card may include information about the structure, size, and access conditions of the newly selected file. The amount of data may also depend on the file type. Table 11.1 describes the options for explicit file selection using the SELECT command defined by ISO/IEC 7816-4, while Sequence Diagram 11.1 on the next page shows the sequence of events in a typical file selection process.

In addition to explicit file selection using the SELECT command with an FID, DF name or path, files can be selected implicitly, although this is only possible with standard read and write commands. The file can be selected before the command is actually executed by specifying its five-bit short FID as a supplementary parameter. This is only possible if the file is an EF and it is located in the current DF. The advantages of this method are a simplified command sequence and higher processing speed, since it is not necessary to send an explicit SELECT FILE command to the card.<sup>2</sup>

<sup>1</sup> See also Chapter 12, ‘Smart Card File Management’, on page 421

<sup>2</sup> See also Section 12.5, ‘File Names’, on page 426

Terminal (IFD)		Smart card (ICC)
SELECT		
<i>Command [FID = '3F00'; no other file information necessary]</i>	→	Search the file tree for a file with FID '3F00'  IF (file found) THEN return code = OK ELSE return code = file not found ← <i>Response [return code]</i>
IF (return code = OK) THEN file selection successful ELSE file could not be selected		

**Sequence Diagram 11.1** Example processing sequence for the SELECT command

**Table 11.2** The functionality of STATUS according to TS 51.011

STATUS	
Command	• -
Response	<ul style="list-style-type: none"> <li>• information about the currently selected file</li> <li>• return code</li> </ul>

The TS 51.011 standard defines the STATUS command as shown in Table 11.2. It returns the same data to the terminal as the SELECT command with successful file selection. This consists of information about the currently selected file: its type and structure, size, FID, access conditions, and whether it is locked. This command is rarely used; its main purpose is to allow the terminal to query, during the course of a session, which file is currently selected and determine various related conditions.

EN 726-3 specifies the CLOSE APPLICATION command, which supplements the SELECT and STATUS commands and is used to exit applications. The CLOSE APPLICATION command passes the FID of the application to be closed as a parameter, and the card responds by clearing the previously attained security state. This command is mainly useful when the terminal needs to ensure that the security state attained by the card is reset. If the smart card operating system does not support this command or a similar command, the security state of the selected DF can be reliably reset to its initial state by MF selection in accordance with the definition in ISO/IEC 7816-4.

## 11.2 READ AND WRITE COMMANDS

Read and write commands primarily support applications that utilize smart cards for secure data storage. These commands can be used to write data to EFs and subsequently read this data. If these EFs have specific access conditions, only authorized programs or users are allowed to read the files. The data stored in the card is thus protected against unauthorized access.

Several types of read and write commands are available, since EF files can have several different data structures. Unfortunately, this does not fully agree with the principles of

object-oriented file management. With a pure object-oriented structure, the operating system must be designed to allow an object to determine its own access mechanisms. This is not the case with smart card file management. This noncompliance arises from the historical development of commands that were later incorporated into current standards. Memory cards, which are the precursors of smart cards, have a single memory region that can be read and written using offset and length parameters. To the outside world, this memory appears as a single file with transparent structure. The first smart cards were built according to the same principle, and the definitions of the commands for reading and writing transparent files date from this time. Later, when other file structures were defined, new commands specifically adapted to these structures were defined for use with these files. As a result, there are two different types of file access.

Accordingly, this class of commands is divided into commands for accessing EFs with transparent structure and commands for accessing EFs with other structures (cyclic, linear fixed, and linear variable). However, several standards (such as EN 1546 for electronic purses) explicitly state that read commands for files with transparent structure may also be used to read files with other structures. In any case, such commands can be used to obtain additional information about the internal structure of the file.

EFs with transparent logical structure are amorphous, which means they do not have an internal structure. They correspond to a linearly addressable memory with byte access. The READ BINARY command (see Table 11.3) is used to read data from these files, while the WRITE BINARY command (see Table 11.4) and the UPDATE BINARY command (see Table 11.5) are used to write data.

The basic difference between WRITE BINARY and UPDATE BINARY involves the secure state of nonvolatile memory in the card. The secure state is the logical state of the bits in nonvolatile memory when the memory cells are in the minimum-energy state. As the memory cells are small capacitors, this is the state in which they contain no charge, which is usually

**Table 11.3** The functionality of READ BINARY according to ISO/IEC 7816-4

READ BINARY	
Command	<ul style="list-style-type: none"> <li>• number of bytes to be read</li> <li>• offset to the first byte to be read</li> <li>• <i>optional</i>: short FID for implicit file selection</li> </ul>
Response	<ul style="list-style-type: none"> <li>• data read from the file</li> <li>• return code</li> </ul>

**Table 11.4** The functionality of WRITE BINARY according to ISO/IEC 7816-4

WRITE BINARY	
Command	<ul style="list-style-type: none"> <li>• number of bytes to be written</li> <li>• data bytes to be written</li> <li>• offset to the first byte to be written</li> <li>• <i>optional</i>: short FID for implicit file selection</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

**Table 11.5** The functionality of UPDATE BINARY according to ISO/IEC 7816-4

UPDATE BINARY	
Command	<ul style="list-style-type: none"> <li>• number of bytes to be updated</li> <li>• update bytes</li> <li>• offset to the first byte to be updated</li> <li>• <i>optional:</i> short FID for implicit file selection</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

the logic 0 state. In order to set a bit in state 0 to state 1, it must be erased. This restores the charge on the capacitor.

A WRITE command can only be used to change bits from the nonsecure state, in this case logic 1, to the secure state, in this case logic 0. This means that the WRITE command effectively performs a logical AND on the supplied data and the data in the file. By contrast, if the secure state of the chip corresponds to logic 1, the WRITE command must perform a logical OR operation on the data supplied by the command and the data in the file. The logical operation performed on the data supplied by the command and the data in the file is always chosen such that the WRITE command causes the nonvolatile memory to attain its secure state. The WRITE command may also support ‘write once, read multiple’ (WORM) access, depending on the file. The WRITE command originates from a time when atomic operations for file access were unknown in the smart card realm. Even now, it is used only very rarely.

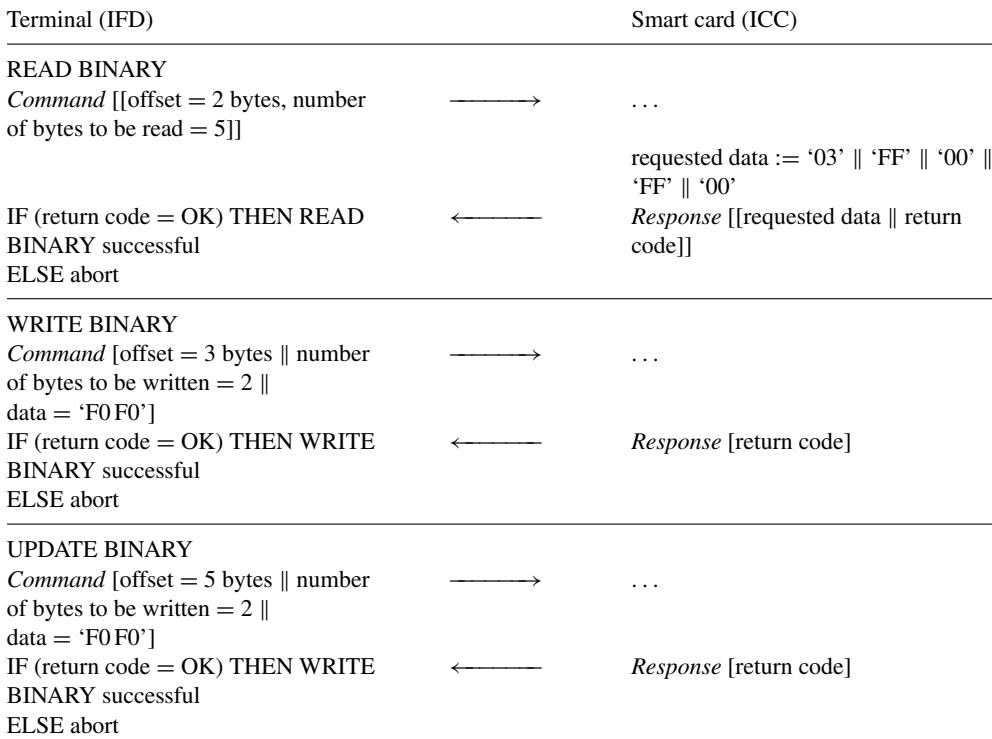
The UPDATE command, by contrast, performs a genuine write to the file. The previous state of the data in the file does not affect the contents of the file after execution of an UPDATE command. UPDATE is thus equivalent to first clearing the data in the file with ERASE and then writing new data with WRITE.

The UPDATE command can be used to construct physically secure smart card counters. This is based on a bit field in which each bit represents a monetary unit. When a payment is made, the counter is decremented bit by bit using logical OR operations generated by WRITE commands. After authentication, the counter value can be increased again by using an UPDATE command. The principal advantage of this method is that it makes it impossible to increase the counter value by manipulating the nonvolatile memory, such as by heating it, since the secure state of each bit represents a value of 0.

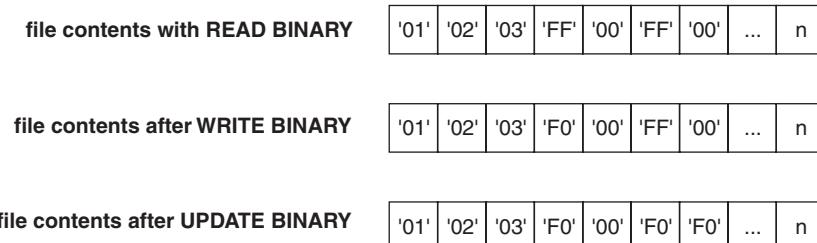
As their names indicate, READ BINARY is a read command, while WRITE BINARY and UPDATE BINARY are write commands. The file is accessed with a length parameter and an offset to the first byte to be accessed. Some operating systems also allow implicit file selection using a short-FID parameter before the actual file access occurs. However, this option is not provided by all standards or operating systems.

Sequence Diagram 11.2 on the facing page shows a typical command sequence with READ BINARY followed by WRITE BINARY and finally UPDATE BINARY. The effects on the contents of the selected file are shown in Figure 11.3 on the next page. Of course, this example assumes that file selection is successful and the access conditions of the file to be written are satisfied.

ERASE BINARY (see Table 11.6) is an exception among the commands for use with transparent EFs. It cannot be used to write data to a file, but only to erase data starting from an offset specified by the command. If a second offset is not specified, the command erases all



**Sequence Diagram 11.2** Example of access to the transparent-structure file illustrated in Figure 11.3



**Figure 11.3** Example of write accesses to an EF with transparent structure using the command sequence shown in Sequence Diagram 11.2

data to the end of the selected file. In this case, erasing means that the data region specified by the command is set to the logically erased state. This state must be defined individually for each operating system, since it may not be the same as the physically erased state of the memory.

As the structures of linear fixed, linear variable and cyclic EFs are fundamentally different from the structures of transparent EFs, specific commands for accessing these data structures are available in addition to the commands described above. All such file have record-oriented structures. For write access, the smallest addressable unit in the file body is a single record.

**Table 11.6** The functionality of ERASE BINARY according to ISO/IEC 7816-4

---

ERASE BINARY	
Command	<ul style="list-style-type: none"> <li>• offset to the first byte to be erased</li> <li>• <i>optional</i>: offset to the last byte to be erased</li> <li>• <i>optional</i>: short FID for implicit file selection</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

**Table 11.7** The functionality of READ RECORD according to ISO/IEC 7816-4

---

READ RECORD	
Command	<ul style="list-style-type: none"> <li>• number of record to be read</li> <li><i>or</i></li> <li>mode (current, first, last, next, or previous record)</li> <li><i>or</i></li> <li>read all records from <math>n</math> to the last record</li> <li><i>or</i></li> <li>read all records from the first record to <math>n</math></li> <li><i>optional</i>: short FID for implicit file selection</li> </ul>
Response	<ul style="list-style-type: none"> <li>• data read from the file</li> <li>• return code</li> </ul>

---

**Table 11.8** The functionality of WRITE RECORD according to ISO/IEC 7816-4

---

WRITE RECORD	
Command	<ul style="list-style-type: none"> <li>• record data to be written</li> <li>• number of the record to be written</li> <li><i>or</i></li> <li>mode (current, first, last, next, or previous record)</li> <li><i>optional</i>: short FID for implicit file selection</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

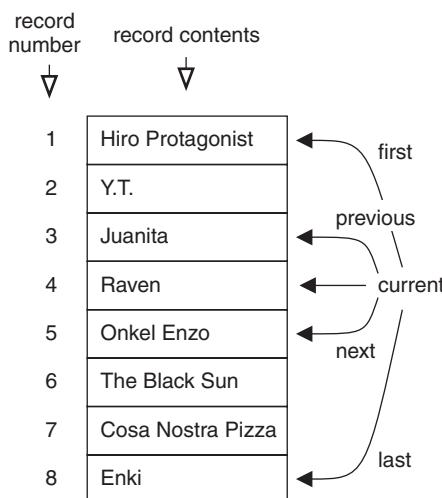
For read access, either an entire record or part of a record may be read, starting with the first byte of the record.

These file structures, which transform a linear, one-dimensional data store into a data store that can be addressed in two dimensions, yield access types that are significantly more complex than those used with a transparent structure. In principle, all possible data structures can be emulated using a transparent structure, but in specific cases this may be considerably more complicated than using a more elaborate data structure.

After an EF with a record-oriented structure has been selected, the smart card operating system creates a record pointer with an undefined initial value. The pointer value can be set by a READ RECORD, WRITE RECORD, UPDATE RECORD command (see Tables 11.7, 11.8 and 11.9) or a SEEK command. The record pointer of the current file persists as long as

**Table 11.9** The functionality of UPDATE RECORD according to ISO/IEC 7816-4

UPDATE RECORD	
Command	<ul style="list-style-type: none"> <li>• update data for the record</li> <li>• number of the record to be updated</li> </ul> <p>or</p> <ul style="list-style-type: none"> <li>• mode (current, first, last, next, or previous record)</li> <li>• optional: short FID for implicit file selection</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

**Figure 11.4** Access options for a file with a record-oriented structure

the file is selected. After successful explicit or implicit selection of another file, the value of the record pointer is again undefined.

All commands for record-oriented files can use a parameter byte to specify the type of access to the file. The basic type is direct access using the absolute number of the desired record. This type of access does not alter the record pointer. The number of the desired record is sent to the card, and the response contains the content of the record in question.

If the parameter is set to ‘first’, the operating system sets the record pointer to the first record in the file, and this record is read or written according to the type of command used. The parameter setting ‘last’ accesses the final record in a similar manner. The parameter settings ‘previous’ and ‘next’ enable read or write access to the previous and next records, respectively. Finally, the parameter setting ‘current’ can be used to address the record currently referenced by the record pointer. This is illustrated in Figure 11.4.

The large variety of access options for record-oriented data structures originates from the typical structure of a telephone directory. Consider a record whose first field contains a surname and given name, followed in the same record by the associated telephone number, as illustrated in Figure 11.5. The READ RECORD command with the parameter values described above

	record number	name	telephone number
1	Joshua Calvert	47 84 46	
2	Quinn Dexter	089 11 00 11	
3	Louise Kavanagh	089 47 51 22	
4	Alkad Mzu	089 178 098	
5	Ione Saldana	08933 178 234	
6	Kiera Salter	089 23 76 87 33	
7	Ralph Hiltch	089 78 123 78 1	
8	Fletcher Christian	089 12 111 222	
9	Kelly Tirrel	089 92 178 234	
10	Gerald Skibbow	089 129 167 189	

**Figure 11.5** Example telephone number list in a file with ‘linear fixed’ structure

**Table 11.10** The functionality of APPEND RECORD according to ISO/IEC 7816-4

APPEND RECORD	
Command	<ul style="list-style-type: none"> <li>• record data to be written</li> <li>• <i>optional:</i> short FID for implicit file selection</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

can be used with a telephone directory mapped into an EF to scroll forward and backward through the directory or jump to the first or last entry. The record pointer can also be set using the search command SEEK, which is described below.

Sequence Diagram 11.3 on the next page shows an example of a series of several read and write operations on the file illustrated in Figure 11.4 on the preceding page.

ISO/IEC 7816-4 also provides the option of reading all records from the first record to a specified record number. Similarly, all records from a specified record number to the last record can be read in a single command-response sequence using READ RECORD. As useful as these commands may be, the capacity of the I/O buffer can easily be exceeded if they are used with large files.

In terms of its functionality, the APPEND RECORD command (see Table 11.10) could also be classified as a file management command. It can be used to append records to existing record-oriented files. The entire data content of the new record is supplied with the command. The full functionality of this command can only be supported by smart cards with relatively elaborate memory management. The memory manager must create the link from the existing

Terminal (IFD)	Smart card (ICC)
<b>READ RECORD</b>	
Command [record number = 2]	————→ <i>command processing</i>
IF (return code = OK)	←———— <i>Response</i> ['Y.T.'    return code]
THEN READ RECORD successful	
ELSE abort	
<b>UPDATE RECORD</b>	
Command [first, 'Wolfgang']	————→ <i>command processing</i>
IF (return code = OK)	←———— <i>Response</i> [return code]
THEN UPDATE RECORD successful	
ELSE abort	
<b>UPDATE RECORD</b>	
Command [next, 'Alex']	————→ <i>command processing</i>
IF (return code = OK)	←———— <i>Response</i> [return code]
THEN UPDATE RECORD successful	
ELSE abort	
<b>READ RECORD</b>	
Command [record number = 2]	————→ <i>command processing</i>
IF (return code = OK)	←———— <i>Response</i> ['Alex'    return code]
THEN READ RECORD successful	
ELSE abort	

**Sequence Diagram 11.3** Example read and write operations on the record-oriented file illustrated in Figure 11.4 on page 363

records in the file to the new record. With this, any desired number of new records can be added to an existing file if the smart card has sufficient memory capacity.

Many smart card operating systems allow only a limited number of new records in order to simplify the implementation of file management functions. In such systems, memory space corresponding to the anticipated need for future file extensions is reserved when a record-oriented file is created. This space can later be filled using APPEND RECORD commands. Once this space is used up, further file extension using APPEND RECORD is not possible.

If APPEND RECORD is used with files having linear fixed or linear variable structure, the new record is always added to the end of the file. If the file structure is cyclic, the new record is always assigned record number 1, which usually corresponds to the current write record with files having this file structure.

APPEND RECORD can be used for various purposes. One possibility is a telephone directory, as already mentioned. Another possibility is a log file, in which the data to be logged is written immediately by generating a new record in the card.

ETSI specification TS 102 222 defines a command called RESIZE for changing the size of a directory (MF, DF, or ADF) or a data file (EF). Changing the size of a directory affects the amount of memory available to the associated application. After execution of this command, the application can have either more memory available (if sufficient memory is available in the card) or less memory available.

Data files can also be made larger or smaller with RESIZE. Both operations act on the end of the file, which in the case of an EF with linear structure means that more storage space is

**Table 11.11** The functionality of GET DATA according to ISO/IEC 7816-4

GET DATA	
Command	<ul style="list-style-type: none"> <li>• number of data objects to be read</li> <li>• tag of the data objects to be read</li> </ul>
Response	<ul style="list-style-type: none"> <li>• read data objects</li> <li>• return code</li> </ul>

**Table 11.12** The functionality of PUT DATA according to ISO/IEC 7816-4

PUT DATA	
Command	<ul style="list-style-type: none"> <li>• structure of the data objects to be written</li> <li>• data objects to be written</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

available at the end of the file after it has been enlarged. If such a file is made smaller, any user data that may be present at the end of the file is cut off. Execution of RESIZE requires prior authentication in conformance with the access rules of the file concerned. To provide maximum functionality, the data supplied in the file body is TLV-coded.

The file-based read and write commands are supplemented by commands that provide direct access to data objects. Depending on the selected DF or EF, they can be used to read specific TLV-coded data objects from internal operating system structures or write TLV-coded data objects to these structures. GET DATA can read data objects, while PUT DATA can write data objects (see Tables 11.11 and 11.12). Both commands can act on one or more TLV-coded data objects. The specified access conditions must be satisfied in advance for each of these commands.

The TS 102 221 specification defines two commands with similar functions: SET DATA and RETRIEVE DATA. They are both designed for access to TLV-coded files and are intended to be used for reading and writing large files. TLV-coded files of this sort are commonly used to store large volumes of data (in the megabyte range) in SIM cards. For example, they can contain photos that are assigned to specific telephone book entries after they are read by the mobile telephone.

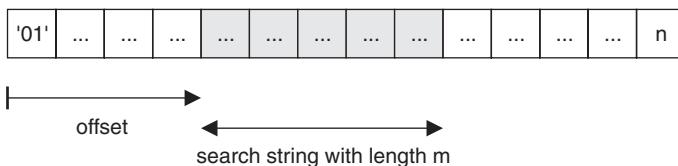
## 11.3 SEARCH COMMANDS

Record-oriented structures are well suited to storing sets of related data with identical structures in a single file. A typical example is a telephone directory containing names and telephone numbers. A search command can be used to avoid having to read the entire directory, record by record, when looking for a particular name.

The SEEK command can search for a specified character string in a record-oriented data structure. An offset can be supplied with the command. The length of the search string is variable. The search direction must be specified in the command. The direction can be forward (in the direction of increasing record numbers) or backward (in the direction of decreasing

**Table 11.13** The functionality of SEEK according to TS 51.011

SEEK	
Command	<ul style="list-style-type: none"> <li>length of the search string</li> <li>search string</li> <li>offset</li> <li>mode (forward from the start, backward from the end, forward from the next record, backward from the previous record)</li> <li><i>switch</i>: return the record number of the located record</li> </ul>
Response	<ul style="list-style-type: none"> <li>record number (if selected by the switch)</li> <li>return code</li> </ul>

**Figure 11.6** Search region definition for searching in a file with a record-oriented structure**Table 11.14** The functionality of SEARCH RECORD according to ISO/IEC 7816-9

SEARCH RECORD	
Command	<ul style="list-style-type: none"> <li>length of the search string</li> <li>search string</li> <li>offset</li> <li>mode (forward from the start, backward from the end, forward from the next record, backward from the previous record)</li> <li><i>switch</i>: return the record number of the located record</li> <li><i>optional</i>: short FID for implicit file selection</li> </ul>
Response	<ul style="list-style-type: none"> <li>record number (if selected by the switch)</li> <li>return code</li> </ul>

record numbers). The starting point of the search must also be specified. The starting point can be the first record, the last record, or the current record. If the search string is found, the operating system sets the record pointer to the entry holding the string and informs the terminal that the search was successful.

The ISO/IEC 7816-9 standard describes two commands that can be used to search for data in transparent and record-oriented files. The SEARCH RECORD command (see Table 11.14) is the ISO/IEC counterpart of the TS 51.011 SEEK command. The main difference between these two commands is that ISO/IEC 7816-9 provides the option of supplying a short FID with the command for implicit EF selection.

The SEARCH BINARY command (see Table 11.15) can search for specified data in a transparent file selected either explicitly by a previous command or implicitly using a command

**Table 11.15** The functionality of SEARCH BINARY according to ISO/IEC 7816-9

SEARCH BINARY	
Command	<ul style="list-style-type: none"> <li>• length of the search string</li> <li>• search string</li> <li>• offset</li> <li>• <i>optional</i>: short FID for implicit file selection</li> </ul>
Response	<ul style="list-style-type: none"> <li>• offset to the located data</li> <li>• return code</li> </ul>

Terminal (IFD)		Smart card (ICC)
SEEK		
<i>Command</i> [search string = ‘Enki’    search direction = forward from start    send record number]	→	<i>command processing</i>
IF (return code = OK) THEN search string was found ELSE search string was not found	←	<i>Response</i> [record number = 8    return code]
SEEK		
<i>Command</i> [search string = ‘Hiro Protagonist’    search direction = backward from end    send record number]	→	<i>command processing</i>
IF (return code = OK) THEN search string was found ELSE search string was not found	←	<i>Response</i> [record number = 1    return code]

**Sequence Diagram 11.4** Example of a SELECT FILE command sequence based on the file illustrated in Figure 11.4 on page 363

parameter. The result is the offset from the start of the file to the first byte of the located search string.

Sequence Diagram 11.4 shows several options for using the SEEK command. It is based on the file with linear fixed structure illustrated in Figure 11.4 on page 363.

## 11.4 FILE OPERATION COMMANDS

There are several commands that allow file contents to be modified by means other than simple writing. The two representatives of this group described here are INCREASE and DECREASE (see Tables 11.16 and 11.17). When applied to a file with cyclic structure that is used as a counter, these commands increment or decrement the counter by an amount specified in the command.

**Table 11.16** The functionality of DECREASE according to TS 51.011

DECREASE	
Command	<ul style="list-style-type: none"> <li>● value to be subtracted</li> </ul>
Response	<ul style="list-style-type: none"> <li>● subtracted value</li> <li>● new value of the record</li> <li>● return code</li> </ul>

**Table 11.17** The functionality of INCREASE according to TS 51.011

INCREASE	
Command	<ul style="list-style-type: none"> <li>● value to be added</li> </ul>
Response	<ul style="list-style-type: none"> <li>● added value</li> <li>● new value of the record</li> <li>● return code</li> </ul>

Terminal (IFD)		Smart card (ICC)
DECREASE		
<i>Command</i> [value to be subtracted = 3]	————→	<i>command processing</i>
	←————	<i>Response</i> [subtracted value = 3    new value = 7    return code]
IF (return code = OK) THEN DECREASE successful ELSE DECREASE could not be executed		
DECREASE		
<i>Command</i> [value to be subtracted = 2]	————→	<i>command processing</i>
	←————	<i>Response</i> [subtracted value = 2    new value = 5    return code]
IF (return code = OK) THEN DECREASE successful ELSE DECREASE could not be executed		
INCREASE		
<i>Command</i> [value to be added = 5]	————→	<i>command processing</i>
	←————	<i>Response</i> [added value = 5    new value = 10    return code]
IF (return code = OK) THEN INCREASE successful ELSE INCREASE could not be executed		

**Sequence Diagram 11.5** Example command sequence with INCREASE and DECREASE. The initial and final value of the file is 10

**Table 11.18** The functionality of EXECUTE according to EN 726-3

EXECUTE	
Command	<ul style="list-style-type: none"> <li>• data to be passed to the program in the executable file</li> </ul>
Response	<ul style="list-style-type: none"> <li>• data to be returned from the program in the executable file</li> <li>• return code</li> </ul>

The cyclic file structure is specified for this purpose because it supports the logging function. These commands are primarily used with simple electronic purses or counters.

For the sake of simplicity, Sequence Diagram 11.5 on the previous page assumes a cyclic file with only one record. The starting value of the record is 10. On conclusion of the process, the record has the same value again.

The EXECUTE command (see Table 11.18) can also be regarded as a file manipulation command in a certain sense. It can run executable EFs (whose structure is ‘executable’). The operating system mechanisms necessary for this are described in detail in Section 13.15, ‘Executable Native Code’, on page 493. The program to be run can receive data from the terminal in the command, and it can send data that it generates back to the terminal in the response. This command and the related file structure are contentious because they can potentially be used to bypass the entire security system of a smart card.

## 11.5 COMMANDS FOR AUTHENTICATING PERSONS

In addition to being used as secure data storage media, smart cards can be used to identify individuals. The usual method for this involves exchanging confidential information that is known only to the user and the card. This is usually a personal identification number (PIN).

The user enters a PIN at a terminal, and shortly thereafter a display indicates whether the PIN was correct or, if not, how many attempts are still allowed. The smart card receives the PIN (usually a four-digit number) from the terminal in a VERIFY command and compares it with a value previously stored in its nonvolatile memory.

If the entered PIN matches the stored PIN, the card’s internal state changes and the terminal is advised that the result of the comparison was positive. The PIN retry counter is also reset to its original value (0). If the entered PIN does not match the stored PIN, the retry counter is incremented. If the retry counter reaches its defined maximum value, the card is blocked for further PIN verification.

Many smart card operating systems can manage more than one PIN. With such systems, the identification number of the reference PIN must also be supplied with all commands related to PINs to enable selection of the proper reference PIN. However, card issuers usually have a strong preference for allowing only one PIN per card, even if it is technically possible to have more than one. This is an important consideration for customer acceptance and ease of use.

The ISO/IEC 7816-4 standard describes a PIN verification command that is largely equivalent to the TS 51.011 command. Its name is VERIFY, and it can be used not only for PIN comparison, but also for verification of biometric features (see Table 11.19). There is only one significant difference between the coding of the VERIFY CHV command defined for GSM applications (see Table 11.20). The ISO/IEC definition distinguishes two types of PINS: global

**Table 11.19** The functionality of VERIFY according to ISO/IEC 7816-4

VERIFY	
Command	<ul style="list-style-type: none"> <li>• PIN or biometric feature (the secret)</li> <li>• number of the PIN or biometric feature</li> <li>• <i>switch</i>: global or application-specific secret</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

**Table 11.20** The functionality of VERIFY CHV according to TS 51.011

VERIFY CHV	
Command	<ul style="list-style-type: none"> <li>• PIN</li> <li>• number of the PIN</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

and application-specific. It is thus possible to specify in the command which type of PIN is to be verified: a PIN that applies to the entire smart card or a PIN that applies only to the current DF.

In the telecommunication sector, the abbreviation CHV (cardholder verification) is often used in place of PIN. These two terms are equivalent. As some of the commands described here originate from the telecommunication sector, their names contain the abbreviation CHV instead of PIN.

In some applications, cardholders are expected to choose their own PINs when the card is first used. The null-PIN method can be used for this purpose. With this method, a default PIN code of 0 is entered when the card is personalized. This code has a special meaning for smart card operating systems, which reject all VERIFY commands with a null PIN and request a PIN change using the CHANGE REFERENCE DATA command before proceeding any further. This compels the user to change the PIN, since the security state of the card cannot be altered if a null PIN is entered. Although this method is not standardized, it can be used to advantage in certain scenarios if it is supported by the operating system.

The number of PINs in use has grown relentlessly since they were introduced as a cardholder identification feature. The average card user nowadays is expected to keep track of perhaps ten to twenty different PINs for a variety of cards and other access authorizations. This is unrealistic, as can be seen from the large number of people who write their PIN code on the card. In smart card systems, users can select their own PINs, which enables them to use the same PIN with all their cards. Although this may be questionable from a security perspective, since an attacker who manages to learn one PIN thereby knows all of them, it is still better than writing the PIN code on the card or an accompanying slip of paper where everyone can read it.

The ISO/IEC 7816-8 CHANGE REFERENCE DATA command provides the function for changing the PIN code. The equivalent command in the telecommunication sector, which has similar input and output parameters, is CHANGE CHV (see Table 11.21). If the PIN

**Table 11.21** The functionality of CHANGE CHV according to TS 51.011

---

CHANGE CHV	
Command	<ul style="list-style-type: none"> <li>• old PIN</li> <li>• new PIN</li> <li>• number of the PIN</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

**Table 11.22** The functionality of RESET RETRY COUNTER according to ISO/IEC 7816-8

---

RESET RETRY COUNTER	
Command	<ul style="list-style-type: none"> <li>• number of the PIN</li> <li>• <i>option</i>: PUK    new PIN</li> <li>• <i>if option selected</i>: number of the PUK</li> <li>• <i>switch</i>: global PIN or application-specific PIN</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

**Table 11.23** The functionality of UNBLOCK CHV according to TS 51.011

---

UNBLOCK CHV	
Command	<ul style="list-style-type: none"> <li>• PUK</li> <li>• new PIN</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

currently stored in the card is known, it can be replaced by a new one. If the current PIN is entered incorrectly, the operating system increments the retry counter to protect against possible attempts to use this mechanism to attack the PIN. If the current PIN is correctly sent to the card, the card stores the new PIN it has received in the appropriate memory location and resets the retry counter.

ISO/IEC 7816-8 has its own command for resetting the retry counter when it has reached its maximum value. This command is called RESET RETRY COUNTER (see Table 11.22). A specific security state, which is usually attained by mutual authentication, is required for execution of this command. Despite its name, this command can also be used to replace the current PIN by a new PIN.

If a PIN retry counter has reached its maximum value, it can be reset by using the UNBLOCK CHV command (see Table 11.23) with a second type of PIN called a personal unblocking key (PUK). The PUK is usually longer than the normal four-digit PIN (eight digits, for example). The user need not memorize the PUK, since it is only needed if the PIN has been forgotten. It is sufficient if the user has a record of the PUK somewhere at home for use if necessary. However, simply resetting the retry counter would not help the user very much because the user would still not know the correct PIN. Consequently, the UNBLOCK CHV command must also supply a new PIN to the card.

**Table 11.24** The functionality of ENABLE VERIFICATION REQUIREMENT according to ISO/IEC 7816-8

ENABLE VERIFICATION REQUIREMENT	
Command	<ul style="list-style-type: none"> <li>• reference data (e.g. PIN)</li> <li>• number of the reference data</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

**Table 11.25** The functionality of DISABLE VERIFICATION REQUIREMENT according to ISO/IEC 7816-8

DISABLE VERIFICATION REQUIREMENT	
Command	<ul style="list-style-type: none"> <li>• reference data (e.g. PIN)</li> <li>• number of the reference data</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

This command should not be used to change PIN codes in hybrid cards, which have a magnetic stripe as well as a chip, as otherwise the PIN calculated from the magnetic stripe data would not match the pin in the chip. Consequently, with such cards the retry counter is simply reset and the customer is sent a letter with the original PIN.

The ISO/IEC 7816-8 commands for controlling PIN verification are called ENABLE VERIFICATION REQUIREMENT and DISABLE VERIFICATION REQUIREMENT (see Tables 11.24 and 11.25). A specific security state, depending on the application, must be attained before these commands can be executed.

The GSM system also has two commands that can enable or disable PIN verification: DISABLE CHV and ENABLE CHV. If PIN verification is disabled, all file access restrictions that require prior PIN verification are also disabled. These two commands are very popular in the mobile telecommunication world because they eliminate the need to enter a PIN code every time the mobile telephone is switched on. They are questionable from a security perspective because they disable protection against unauthorized use, which is the purpose of PIN codes. Of course, users could also use CHANGE CHV to change the PIN to a trivial value such as ‘0000’, which offers just as little protection.

For obvious reasons, the PIN verification processes described here are targets of attack because a large financial gain could be realized by using a lost or stolen card with the right PIN. All commands involving PIN or PUK comparison must be protected against analysis of the card’s electrical and time behavior. For instance, the electrical current consumption during execution of the VERIFY PIN command must be constant, regardless of whether the entered PIN is correct. It is equally important that the time required to execute PIN commands is independent of whether the PIN is true or false. Variable execution times could have fatal consequences for the card’s security, and thus ultimately the security of the entire system. Such variations could be used to determine the value of the correct PIN in a very simple manner, causing all of the system’s PIN codes to be rendered worthless as a means of user identification.

## 11.6 COMMANDS FOR AUTHENTICATING DEVICES

In addition to commands for authenticating cardholders, there are several commands for authenticating terminals and cards. As each of these entities is equipped with a complete computer, the processes used for this purpose can be made much more complex, and thus more secure, than the processes used for PIN verification.

With PIN verification, the card receives a secret (the PIN code) via the interface and only has to compare it with a PIN code held in memory. Tapping the transmission line would thus have fatal consequences if the PIN is transferred in plaintext form. Modern authentication methods are designed to make such attacks impossible.

In principle, authentication involves verifying a secret known to both of the communicating parties without requiring it to be sent across the interface. The methods are constructed such that tapping the data transmission does not compromise the security of the authentication.<sup>3</sup>

Depending on the operating system, various commands are available for authenticating the card, the terminal, or both at the same time. For the sake of clarity, here and in the rest of this chapter we refer to this as authentication between the card and the terminal. However, in terms of information technology what actually happens is that the ‘outside world’ authenticates itself with respect to an application in the card. This does not verify that the card as a whole is genuine, but only that the embedded microcontroller shares a secret with the outside world. This must be borne in mind in certain situations.

In many operating systems, the keys used for authentication are protected by a retry counter. If a terminal unsuccessfully attempts authentication too many times, the card blocks the associated key for use in further authentication attempts. There is nothing wrong with this in terms of system security, but it does have a drawback: resetting the retry counter of the authentication key to its initial value often involves very elaborate procedures that are logistically difficult and costly to administer. Consequently, some systems do not use retry counters for authentication keys.

For security reasons, only card-specific keys should be used for authentication. A unique card feature can be used as the basis for generating these keys. A serial number or chip fabrication number is very suitable for this purpose. These numbers are public, and they can be stored in the card or read from the card by suitable commands, such as GET DATA. This command can fetch a serial number from the card, which should preferably have a length of eight bytes for use with the AES algorithm. It acts as a unique chip identifier and is used to calculate card-specific keys. There is one more command that is needed for authentication: GET CHALLENGE as defined in ISO/IEC 7816-4 (see Table 11.26), which requests a random number from the card. This number is subsequently used in the authentication process. It typically has a length of eight bytes with AES, although the lengths used with other cryptographic algorithms may be different.

In order to make the following example processes reasonably easy to understand and avoid unnecessary complexity, we have omitted the description of card-specific key derivation. However, it is indispensable for reasons of security.

The INTERNAL AUTHENTICATE command (see Table 11.27) enables the terminal to authenticate a card, or in the case of a multiapplication card, to authenticate an application. This command can be used to verify that a card is genuine. For this purpose, the card receives

<sup>3</sup> See also Section 7.4, ‘Authentication’, on page 166

**Table 11.26** The functionality of GET CHALLENGE according to ISO/IEC 7816-4

GET CHALLENGE	
Command	• —
Response	• random number • return code

**Table 11.27** The functionality of INTERNAL AUTHENTICATE according to ISO/IEC 7816-4

INTERNAL AUTHENTICATE	
Command	• random number • number of the algorithm to be used • number of the key to be used
Response	• $E(\text{key}, \text{random number})$ • return code

**Sequence Diagram 11.6** Example command sequence with INTERNAL AUTHENTICATE

Terminal (IFD)		Smart card (ICC)
INTERNAL AUTHENTICATE		
<i>Command</i> [random number, key number]	————→	$X_{\text{ICC}} := E(\text{key}, \text{random number})$
$X_{\text{IFD}} := E(\text{key}, \text{random number})$	←————	<i>Response</i> [ $X_{\text{ICC}} \parallel \text{return code}$ ]
IF (return code = OK) AND ( $X_{\text{IFD}} = X_{\text{ICC}}$ ) THEN command successfully executed and smart card authenticated ELSE authentication failed		

a random number and encrypts this number using a cryptographic algorithm such as AES with a key that is known only to the card and the terminal. The result of this encryption is returned to the terminal in the response. The terminal performs the same encryption as the card and compares its result with the result in the response received from the card. If the two results match, it follows that the card also knows the secret authentication key and must be genuine. The card has thus been authenticated.

This command implements the classic challenge-response method for authenticating a communication party. The exact data content of the challenge is not specified in detail in the ISO/IEC 7816-4 standard, which only stipulates that the value sent to the card must be random and session-specific. Consequently, when INTERNAL AUTHENTICATE and the other authentication commands described here are used in an application, a detailed specification is always necessary to ensure that they can be used interoperably.

**Table 11.28** The functionality of EXTERNAL AUTHENTICATE according to ISO/IEC 7816-4

EXTERNAL AUTHENTICATE		
Command	<ul style="list-style-type: none"> <li>• <math>E(\text{key}, \text{random number})</math></li> <li>• number of the algorithm to be used</li> <li>• number of the key to be used</li> </ul>	
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>	
Terminal (IFD)		Smart card (ICC)
GET CHALLENGE		
<i>Command[]</i>	→	generate random number
IF (return code = OK)	←	<i>Response</i> [random number    return code]
THEN command successfully executed		
ELSE abort		
$X_{\text{IFD}} := E(\text{key}, \text{random number})$		
EXTERNAL AUTHENTICATE		
<i>Command</i> [ $X_{\text{IFD}}$ , key number]	→	$X_{\text{ICC}} := E(\text{key}, \text{random number})$ IF ( $X_{\text{ICC}} = X_{\text{IFD}}$ ) THEN terminal authenticated ELSE authentication failed <i>Response</i> [return code]
IF (return code = OK)	←	
THEN terminal authenticated		
ELSE authentication failed		

**Sequence Diagram 11.7** Example command sequence with EXTERNAL AUTHENTICATE

The terminal uses the EXTERNAL AUTHENTICATE command (see Table 11.28) to show the card that it is in contact with a genuine terminal. This command has to be initiated by the terminal because the communication process must always operate within the challenge-response framework. However, the card can compel authentication of the terminal by blocking access to certain files until the terminal has been successfully authenticated.

The process, as shown in Sequence Diagram 11.7, starts when the terminal sends a GET CHALLENGE command to request a random number from the smart card, which the terminal then encrypts using a secret key. With the next command, EXTERNAL AUTHENTICATE, the terminal returns the encrypted random number to the smart card. The card performs the same encryption using the secret key, which it also knows, and then compares the result with the value received from the terminal. If they match, the terminal must also possess the secret authentication key and is thus authenticated.

**Table 11.29** The functionality of MUTUAL AUTHENTICATE according to ISO/IEC 7816-4

MUTUAL AUTHENTICATE	
Command	<ul style="list-style-type: none"> <li>• <math>E(\text{key}, \text{random\_number}_{\text{terminal}}, \text{random\_number}_{\text{smart\_card}}, \text{chip\_number})</math></li> <li>• number of the algorithm to be used</li> <li>• number of the key to be used</li> </ul>
Response	<ul style="list-style-type: none"> <li>• <math>E(\text{key}, \text{random\_number}_{\text{terminal}}, \text{random\_number}_{\text{smart\_card}})</math></li> <li>• return code</li> </ul>

After successful terminal authentication, the operating system changes the state of its state machine. This allows the terminal to access certain files for reading or writing. For the user, this is the visible result of external authentication.

If the INTERNAL AUTHENTICATE and EXTERNAL AUTHENTICATE commands are executed in sequence, the communication parties are mutually authenticated. Each one thus knows that the other is genuine. However, this requires a total of three complete command sequences. In order to simplify this complex and time-consuming process, these three sequences and their data have been merged into a single command called MUTUAL AUTHENTICATE.

The MUTUAL AUTHENTICATE command (see Table 11.29) is specified in the ISO/IEC 7816-8 standard. It can be used to perform mutual authentication of two parties in accordance with ISO/IEC 9798-2 and 9798-3. Using a single authentication command also increases the security of the overall process because it prevents the fraudulent insertion of commands between the two unilateral authentications. A further improvement in the security of the process results from the fact that it is impossible to obtain plaintext–ciphertext pairs by tapping the communication between the terminal and the card, which would otherwise provide an ideal basis for an attack.

The sequence of events for mutual authentication of two parties, as shown in Sequence Diagram 11.8 on the following page, begins when the terminal sends a GET DATA command to request the chip number of the card. It can then compute the card-specific key. The terminal then sends a GET CHALLENGE command to obtain a random number from the card and then generates its own random number.

After the terminal receives the random number from the card, it combines the two random numbers and the chip number to form a single data block, which it encrypts using the secret authentication key and a cryptographic algorithm operating in CBC mode. It sends the resulting ciphertext block to the smart card, which decrypts it and compares the resulting chip number and random number with the numbers it previously sent to the terminal. If they both match, the terminal has been authenticated.

Now the card swaps the two random numbers, discards the chip number, and again encrypts the resulting block using the secret key. After the terminal receives and decrypts this block, it can compare the known random numbers to determine whether the card possesses the secret authentication key. If it does, the smart card is also authenticated.

A future version of ISO/IEC 7816 will probably also include a generic authentication command called GENERAL AUTHENTICATE. This command can be used for a wide variety of authentication functions and processes, such as PACE.

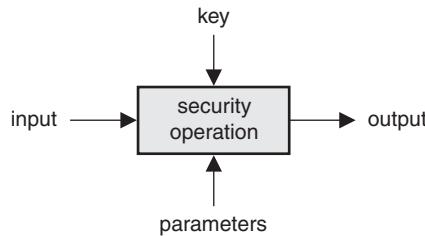
Terminal (IFD)	Smart card (ICC)
GET DATA	
<i>Command</i> [chip number tag]	————→
IF (return code = OK)	←————
THEN Command successful	<i>Response</i> [chip number    return code]
key derivation (compute the card-specific key $K$ )	
GET CHALLENGE	
<i>Command</i> []	————→
IF (return code = OK)	←————
THEN Command successful	<i>Response</i> [RND <sub>ICC</sub>    return code]
MUTUAL AUTHENTICATE	
generate RND <sub>IFD</sub>	
$X_{IFD} = E(K, \text{RND}_{IFD} \parallel \text{RND}_{ICC} \parallel \text{chip}$	
number)	
<i>Command</i> [ $X_{IFD}$    key number]	————→
$X'_{IFD} = D(K, X_{IFD})$	
IF (RND <sub>ICC</sub> and the chip number	
match the sent values)	
THEN terminal authenticated	
ELSE terminal not authenticated	
$X_{ICC} = E(K, \text{RND}_{ICC} \parallel \text{RND}_{IFD})$	
<i>Response</i> [ $X_{ICC}$    return code]	
$X'_{ICC} = D(K, X_{ICC})$	————→
IF (RND <sub>IFD</sub> matches the sent value)	
THEN smart card authenticated	
ELSE smart card not authenticated	

**Sequence Diagram 11.8** Example command sequence for mutual authentication with MUTUAL AUTHENTICATE and a card-specific key

## 11.7 COMMANDS FOR CRYPTOGRAPHIC ALGORITHMS

Commands for cryptographic algorithms are very important in many applications because they facilitate the use of smart cards as encryption and decryption devices or as devices for verifying digital signatures. Many smart card operating systems have their own command sets for performing cryptographic algorithms. Smart card commands such as ENCRYPT, DECRYPT, SIGN DATA, and VERIFY SIGNATURE were developed due to the prolonged absence of a standard for this functionality. However, commands specifically designed for executing cryptographic algorithms have now been defined in the ISO/IEC 7816-8 standard. It divides the functions related to cryptography between two commands: MANAGE SECURITY ENVIRONMENT and PERFORM SECURITY OPERATION (see Figure 11.7).

The MANAGE SECURITY ENVIRONMENT (MSE) command (see Table 11.30) allows various general parameters to be configured before the cryptographic algorithm is executed. This command supplies the relevant parameter values to the card in a structure called a template. They remain valid until they are updated by a new MANAGE SECURITY ENVIRONMENT command. The templates consist of TLV-coded data objects, which enables a high degree of flexibility (and unfortunately, complexity) for transferring parameters.



**Figure 11.7** The basic operating principle of the ISO/IEC 7816-8 MANAGE SECURITY ENVIRONMENT and PERFORM SECURITY OPERATION commands for cryptographic functions

**Table 11.30** The functionality of MANAGE SECURITY ENVIRONMENT according to ISO/IEC 7816-9

MANAGE SECURITY ENVIRONMENT (MSE)	
Command	<ul style="list-style-type: none"> <li>● template with cryptographic checksum parameters <i>or</i></li> <li>template with digital signature parameters <i>or</i></li> <li>template with hash algorithm parameters <i>or</i></li> <li>template with authentication parameters <i>or</i></li> <li>template with encryption and decryption parameters</li> </ul>
Response	<ul style="list-style-type: none"> <li>● return code</li> </ul>

After the parameters of the cryptographic function have been configured by the MANAGE SECURITY ENVIRONMENT command, the PERFORM SECURITY OPERATION (PSO) command can be used. A very wide variety of security operations can be performed using this command, provided that they are supported by the smart card operating system. However, the number of options with this command is so large that this support is not necessarily available. Although PERFORM SECURITY OPERATION is coded using only one instruction byte, it has eight fundamentally different functions that are selected by the P1 parameter byte. This was done because the number of instruction byte codes remaining for coding commands has become somewhat scant recently. As PERFORM SECURITY OPERATION can be used in so many different ways, the following description of its functionality is broken down according to its various options, without going into the details of any of them.

The PERFORM SECURITY OPERATION command with the COMPUTE CRYPTOGRAPHIC CHECKSUM option (see Table 11.31) is used to compute a cryptographic checksum (CCS), which is commonly called a MAC. The padding and the key to be used are either implicitly supplied by the operating system or explicitly specified using the MANAGE SECURITY ENVIRONMENT command. The counterpart to this command option is VERIFY CRYPTOGRAPHIC CHECKSUM, which computes the cryptographic checksum of the data supplied with the command and compares it with a reference value that is also supplied with

**Table 11.31** The functionality of PERFORM SECURITY OPERATION with the COMPUTE CRYPTOGRAPHIC CHECKSUM option according to ISO/IEC 7816-8

---

PERFORM SECURITY OPERATION (PSO)	
Option: COMPUTE CRYPTOGRAPHIC CHECKSUM	
Command	<ul style="list-style-type: none"> <li>• data to be encrypted</li> </ul>
Response	<ul style="list-style-type: none"> <li>• CCS</li> <li>• return code</li> </ul>

---

**Table 11.32** The functionality of PERFORM SECURITY OPERATION with the VERIFY CRYPTOGRAPHIC CHECKSUM option according to ISO/IEC 7816-8

---

PERFORM SECURITY OPERATION (PSO)	
Option: VERIFY CRYPTOGRAPHIC CHECKSUM	
Command	<ul style="list-style-type: none"> <li>• data to be encrypted</li> <li>• CCS</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

**Table 11.33** The functionality of PERFORM SECURITY OPERATION with the ENCIPHER option according to ISO/IEC 7816-8

---

PERFORM SECURITY OPERATION (PSO)	
Option: ENCIPHER	
Command	<ul style="list-style-type: none"> <li>• data to be encrypted</li> </ul>
Response	<ul style="list-style-type: none"> <li>• encrypted data</li> <li>• return code</li> </ul>

---

the command (see Table 11.32). The result is a match/no-match decision, which is returned to the terminal.

The ENCIPHER and DECIPHER options of the PERFORM SECURITY OPERATION are provided for pure data encryption and decryption. The ENCIPHER option (see Table 11.33) causes the data supplied with the command to be encrypted. The encryption algorithm to be used can be selected in advance by the MANAGE SECURITY ENVIRONMENT command, depending on the algorithms supported by the operating system. The mode of the encryption algorithm must also be specified in the same manner by parameter values supplied in advance. With a block encryption algorithm, either ECB or CBC mode can be selected. As the volume of data sent to the smart card is not always an exact multiple of the block length of the cryptographic algorithm, the padding method must be specified by another parameter. The identification of the key in the smart card that the algorithm needs for encryption must also be provided.

The reverse function of ENCIPHER is DECIPHER (see Table 11.34). It can be used in the same mode as ENCIPHER to decrypt the data supplied with the command. Naturally,

**Table 11.34** The functionality of PERFORM SECURITY OPERATION with the DECIPHER option according to ISO/IEC 7816-8

---

PERFORM SECURITY OPERATION (PSO)

Option: DECIPHER

---

Command	<ul style="list-style-type: none"> <li>• encrypted data</li> </ul>
Response	<ul style="list-style-type: none"> <li>• decrypted data</li> <li>• return code</li> </ul>

---

**Table 11.35** The functionality of PERFORM SECURITY OPERATION with the HASH option according to ISO/IEC 7816-8

---

PERFORM SECURITY OPERATION (PSO)

Option: HASH

---

Command	<ul style="list-style-type: none"> <li>• data to be hashed</li> <li>• hash value (if only part of the hash computation is to be performed in the card)</li> <li>• <i>switch</i>: perform only part of the hash computation in the card</li> <li>• <i>switch</i>: return the computed hash value in the response</li> <li>• <i>switch</i>: final command with data to be hashed</li> </ul>
Response	<ul style="list-style-type: none"> <li>• hash value (if selected by the switch)</li> <li>• return code</li> </ul>

---

the smart card must know the appropriate key, algorithm mode, and padding mode. This information must be sent to the operating system in advance by the MANAGE SECURITY ENVIRONMENT command.

The advent of public-key algorithms in the smart card world created a need for suitable commands to facilitate the use of the additional functions. Smart cards in particular are especially suitable for use in digital signature applications because the private key for the signature algorithm can be stored securely in memory, protected against reading.<sup>4</sup> ISO/IEC 7816-8 describes four command options for this functionality.

The HASH option of the PERFORM SECURITY OPERATION command can be used to compute a hash value (see Table 11.35). The command can supply either all of the data to be hashed or a hash value already computed outside the smart card along with the associated data necessary for the final step of the computation. In the latter case, the hash computation for the final block of data can be completed in the card. The advantage of this approach is that the hash computation can be performed significantly faster outside the card, while still allowing the final step to be performed inside the card. From a purely cryptological perspective, it provides only a small amount of extra protection, but it does somewhat restrict the scope of opportunity for manipulating the hash value. For this reason, it is used fairly often in practice.

As the amount of data to be hashed is usually larger than the maximum allowed length of the command body, the HASH option uses layer-7 chaining, which means that the data blocks

<sup>4</sup> See also Section 23.1, ‘Digital Signatures’, on page 899

**Table 11.36** The functionality of PERFORM SECURITY OPERATION with the COMPUTE DIGITAL SIGNATURE option according to ISO/IEC 7816-8

---

PERFORM SECURITY OPERATION (PSO)	
Option: COMPUTE DIGITAL SIGNATURE	
Command	<ul style="list-style-type: none"> <li>• data to be signed <i>or</i> hash value of the data to be signed</li> <li>• <i>switch</i>: final command with data to be signed</li> </ul>
Response	<ul style="list-style-type: none"> <li>• digital signature</li> <li>• return code</li> </ul>

---

**Table 11.37** The functionality of PERFORM SECURITY OPERATION with the VERIFY DIGITAL SIGNATURE option according to ISO/IEC 7816-8

---

PERFORM SECURITY OPERATION (PSO)	
Option: VERIFY DIGITAL SIGNATURE	
Command	<ul style="list-style-type: none"> <li>• data to be verified <i>or</i> hash value of the data to be verified</li> <li>• digital signature</li> <li>• <i>switch</i>: final command with data to be verified</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

are logically chained at the application level. The final data block to be hashed is assigned a marker that informs the command that it is the final block for the hashing process.

This command also has several variants. The computed hash value can either be transferred immediately to the terminal in the response to the command or stored in the card for use with a subsequent command. The padding and the key to be used are defined as necessary by a prior MANAGE SECURITY ENVIRONMENT command, in the same way as for the previously described commands.

The COMPUTE DIGITAL SIGNATURE option (see Table 11.36) can be used for signing data. The data string to be signed, which is usually compressed to a hash value, must be supplied to the smart card if it is not already present in the card as the result of a previous PERFORM SECURITY OPERATION command with the HASH option. The COMPUTE DIGITAL SIGNATURE option also allows the data to be signed to be supplied directly to the smart card, where it is hashed before being signed. As with the HASH option, layer-7 chaining can be used with large data volumes.

If the length of the hash value does not correspond to the input length of the public-key algorithm, padding must be added. The options for this are configured by parameters of the MANAGE SECURITY ENVIRONMENT command, and key selection is handled in the same manner.

The verification function for the COMPUTE DIGITAL SIGNATURE option is provided by the VERIFY DIGITAL SIGNATURE option (see Table 11.37). In principle, any sufficiently

**Table 11.38** The functionality of PERFORM SECURITY OPERATION with the VERIFY CERTIFICATE option according to ISO/IEC 7816-8

---

PERFORM SECURITY OPERATION (PSO)

Option: VERIFY CERTIFICATE

---

Command	<ul style="list-style-type: none"> <li>• certificate</li> <li>• <i>switch</i>: store the public key</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

**Table 11.39** The functionality of GENERATE ASYMMETRIC KEY PAIR according to ISO/IEC 7816-8

---

GENERATE ASYMMETRIC KEY PAIR

---

Command	<ul style="list-style-type: none"> <li>• —</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

fast digital computer could be used to verify a digital signature, since the necessary key is public. However, in many cases the validity of the public key must first be verified using an additional digital signature. As this is a security-related activity, it should not be performed using an insecure computer.

For verification of a digital signature, the associated public key must either be implicitly known to the smart card or be explicitly made known to the card using the command option VERIFY CERTIFICATE, which is described below. The data to be verified may be supplied by VERIFY DIGITAL SIGNATURE either directly or in the form of the associated hash value. All other parameters are the same as for the COMPUTE DIGITAL SIGNATURE option.

In open systems, public keys for verifying digital signatures are usually signed using the private key of the certification authority. The authenticity of a public key must be verified before it is used, since this is the only way to ensure that the key is not a forgery. This verification must take place in a secure environment, such as a smart card, as it would otherwise be vulnerable to manipulation.

The VERIFY CERTIFICATE option (see Table 11.38) is specifically intended to be used for verifying signed public keys, which are also called certificates. After a public key has been identified as genuine, it can be stored permanently in the smart card or used with an immediately following command with the VERIFY DIGITAL SIGNATURE option.

Sequence Diagram 11.9 on the next page illustrates how the commands described above can be used to generate a digital signature and then verify it.

If a smart card operating system supports the generation of key pairs for asymmetric cryptographic algorithms (RSA, DSA, or ECDSA), this process can be initiated by the GENERATE ASYMMETRIC KEY PAIR command defined in the ISO/IEC 7816-8 standard (see Table 11.39). As usual, all parameters needed for key generation must be configured in advance by the MANAGE SECURITY ENVIRONMENT command. After the key pair has been generated, it can be stored in the smart card in a specifiable EF for further use, such as for computing digital signatures. Alternatively, the public key can be exported to the terminal.

Terminal (IFD)	Smart card (ICC)
COMPUTE DIGITAL SIGNATURE <i>Command</i> [hash value of the data]	$\longrightarrow$ $S := S(\text{private key}, \text{hash value of the data})$ $\longleftarrow$ <i>Response</i> [ $S \parallel$ return code]
IF (return code = OK) THEN signature successful ELSE abort	
VERIFY DIGITAL SIGNATURE <i>Command</i> [hash value of the data    $S$ ]	$\longrightarrow$ $H' := E(\text{public key}, S)$ $H := \text{hash value of the data}$ $\text{IF } (H = H')$ $\text{THEN return code} = \text{OK}$ $\text{ELSE return code} = \text{not OK}$ $\longleftarrow$ <i>Response</i> [return code]
IF (return code = OK) THEN signature is genuine ELSE signature is false	

**Sequence Diagram 11.9** Example command sequence using PERFORM SECURITY OPERATION with the COMPUTE DIGITAL SIGNATURE and VERIFY DIGITAL SIGNATURE options. The basic parameters (key and algorithm to be used, etc.) are either specified implicitly or configured in advance by the MANAGE SECURITY ENVIRONMENT command

## 11.8 FILE MANAGEMENT COMMANDS

Most current smart card operating systems allow various file management operations to be performed within the constraints of the specified security conditions, such as extending, creating, deleting, and locking files.

However, most or even all management functions are often omitted in single-application cards because these functions generally require a large amount of program code, which would increase the memory size and thus the cost of the cards. Support for certain management functions is necessary with multiapplication cards, as otherwise all the applications would have to be loaded into the card when it is personalized.

From a system security standpoint, mutual authentication should always take place before any management functions are performed, since they provide ideal starting points for attacks. For example, suppose an unauthorized person could delete a file holding confidential data and then create a new file with the same name, but without any read access conditions set for the new file, so it could be read without invoking any protective mechanisms. The terminal would see a file with the same name as before, and it would write confidential data to the manipulated file as though it were genuine. This type of attack is by no means new; it has been around for many years in a somewhat different form. Nevertheless, it is repeatedly used with successful results in the file management realm.

Another possible point of attack is available if management functions are used in publicly accessible terminals. In such cases, data transfers must always be protected using secure

**Table 11.40** The functionality of CREATE FILE according to ISO/IEC 7816-9

CREATE FILE	
Command	<ul style="list-style-type: none"> <li>• file type of the new file</li> <li>• IF (file type = DF) THEN [DF name of the new file]</li> <li>IF (file type = EF) THEN [ <ul style="list-style-type: none"> <li>FID of the new file</li> <li>SFI of the new file</li> <li>access conditions</li> <li>structure of the new file</li> </ul> ]</li> <li>• IF (file structure = transparent) THEN [file size]</li> <li>IF (file structure = linear fixed) OR (file structure = cyclic) THEN [ <ul style="list-style-type: none"> <li>[number of records</li> <li>record length]</li> </ul> ]</li> <li>IF (file structure = linear variable) THEN [ <ul style="list-style-type: none"> <li>number of records</li> <li>length of each record]</li> </ul> ]</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

messaging functions.<sup>5</sup> This is an essential prerequisite for secure loading of files and applications into cards already in use, such as via public card phones, which for logistical reasons is an attractive option for application providers.

Particularly with multiapplication smart cards, which can be used by different application providers, it is necessary to partition the memory and assign authorization keys for file creation before the individual applications are generated. This prevents any individual application provider from taking over the entire available memory for its own application, leaving none for other applications. For a long time, this was handled by special commands such as REGISTER in combination with CREATE, but here the distinctly more flexible quota mechanisms<sup>6</sup> used with Global Platform have become firmly established. If a quota mechanism is available, the card issuer can use it to specify the maximum amount of memory (code and data) that can later be allocated to an individual application. This method yields a strict separation between memory space allocation and loading new files and applets into the card. This enables issuers of multiapplication cards to sell memory space to several application providers without having to worry about potential memory piracy.

The CREATE FILE command (see Table 11.40) can create DFs or EFs in smart cards after they have been completed. This requires prior attainment of a specific logic state, for example as a result of successful mutual authentication. Depending on the environment in which the CREATE FILE command is used, data transfers should be protected by secure messaging. The CREATE FILE command is defined in the ISO/IEC 7816-9 standard.

After a file has been created with all of its access conditions, attributes and other properties, it can be selected by SELECT FILE and then accessed. Creation of incomplete files by premature termination of the file creation process, in order to produce a file that can be used for attacks, must be prevented by the operating system. In addition, it must not be possible to

<sup>5</sup> See also Section 8.4, ‘Secure Data Transmission’, on page 225

<sup>6</sup> See also Section 13.7.6, ‘Quota mechanism’, on page 470

Terminal (IFD)	Smart card (ICC)
CREATE FILE	
<i>Command</i> [...]	→ <i>command processing</i>
IF (return code = OK)	← <i>Response</i> [return code]
THEN command successfully executed	
ELSE command failed	
UPDATE BINARY/RECORD	
<i>Command</i> [...]	→ <i>command processing</i>
IF (return code = OK)	← <i>Response</i> [return code]
THEN command successfully executed	
ELSE command failed	

**Sequence Diagram 11.10** An example of a typical process for installing a new application. Mutual authentication of the terminal and the smart card is normally necessary before this sequence in order to satisfy the execution conditions for these commands. The two commands may be executed repeatedly, depending on the amount of data to be transferred

**Table 11.41** The functionality of DEACTIVATE FILE according to ISO/IEC 7816–9

DEACTIVATE FILE	
Command	<ul style="list-style-type: none"> <li>• —</li> <li>or</li> <li>FID</li> <li>or</li> <li>short FID</li> <li>or</li> <li>DF name</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

read the residual contents of previously used memory areas that are only partially overwritten when a new file is created.

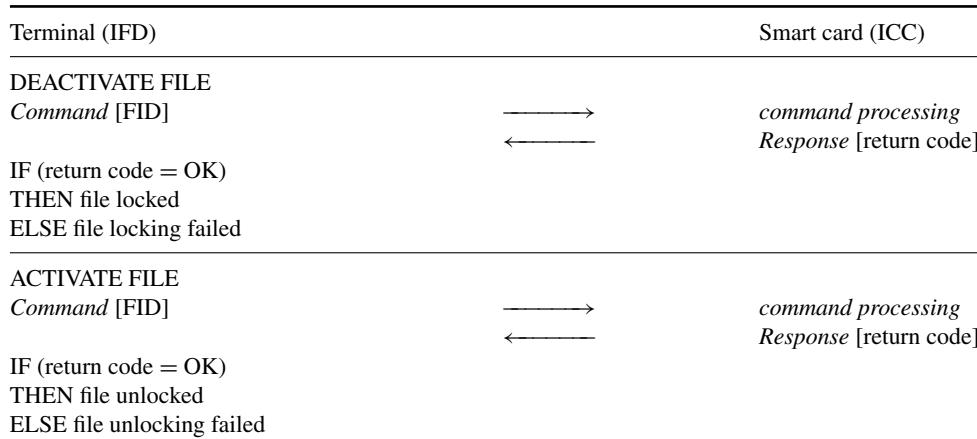
After successful mutual authentication of the smart card and the terminal, the application provider can use CREATE FILE to create its files in the allocated DF. This can be done either on the provider's premises or by other means, such as using a public terminal. Next, the provider fills the EFs with the necessary data and keys and configures the access attributes of the files. After this, the application is operational, and the user can utilize the new functions. The process of installing a new application in the file tree of a smart card is shown in Sequence Diagram 11.10.

The additional commands DEACTIVATE RECORD and ACTIVATE RECORD will be included in a future version of ISO/IEC 7816. They can be used to deactivate and reactivate specific records in a suitable EF.

The ISO/IEC 7816-9 command DEACTIVATE FILE (see Table 11.41) and the TS 51.011 command INVALIDATE enable the terminal to reversibly lock a file after it has been selected. Locked files cannot be accessed for read or write operations; only selection is still allowed. The

**Table 11.42** The functionality of ACTIVATE FILE according to ISO/IEC 7816–9

ACTIVATE FILE	
Command	<ul style="list-style-type: none"> <li>• —</li> <li>or</li> <li>FID</li> <li>or</li> <li>short FID</li> <li>or</li> <li>DF name</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>



**Sequence Diagram 11.11** Example command sequence using the ISO/IEC 7816 commands DEACTIVATE FILE and ACTIVATE FILE. Mutual authentication of the terminal and the smart card is normally necessary before this sequence in order to satisfy the execution conditions for these commands

EMV specification provides a similar command for reversibly blocking applications, called APPLICATION BLOCK. This command does not lock the EFs of the application, but instead only blocks the use of commands for file selection, authentication, and financial transactions in the DF of the application. The functionality of APPLICATION BLOCK is otherwise similar to that of DEACTIVATE FILE and INVALIDATE.

The functions for reversing the effects of INVALIDATE and DEACTIVATE FILE are provided by the REHABILITATE and ACTIVATE FILE commands (see Table 11.42), which can unlock a locked file. Sequence Diagram 11.11 shows an example of command sequences for these commands. The file must be selected before these commands are executed. It goes without saying that execution of these commands can only be allowed in a specific security state, as otherwise every file could be locked or unlocked at will. The EMV specification defines the APPLICATION UNBLOCK command, which can be used to restore access to an application previously blocked by APPLICATION BLOCK.

**Table 11.43** The functionality of TERMINATE EF according to ISO/IEC 7816-9

TERMINATE EF	
Command	<ul style="list-style-type: none"> <li>• currently selected EF</li> <li>    <i>or</i></li> <li>    FID</li> <li>    <i>or</i></li> <li>    file path from the currently selected DF</li> <li>    <i>or</i></li> <li>    file path from the MF</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

**Table 11.44** The functionality of TERMINATE DF according to ISO/IEC 7816-9

TERMINATE DF	
Command	<ul style="list-style-type: none"> <li>• currently selected EF</li> <li>    <i>or</i></li> <li>    DF name</li> <li>    <i>or</i></li> <li>    file path from the currently selected DF</li> <li>    <i>or</i></li> <li>    file path from the MF</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

The TERMINATE EF command (see Table 11.43) is the nonreversible counterpart of DEACTIVATE FILE. A file that has been locked with TERMINATE EF cannot be unlocked. After execution of the TERMINATE EF command, the file can only be selected; all other access commands are blocked by the operating system.

The ISO/IEC 7816-9 standard provides the TERMINATE DF command for irreversible locking of a DF (see Table 11.44). A locked DF can still be selected, but the files in the DF are no longer accessible. This command can be used to terminate an application without erasing the evidence that the application was previously present or deleting the memory allocation of the allocation.

A disadvantage of permanent (irreversible) file locking is that it permanently blocks further use of valuable memory space in the card. A significantly more elegant solution is to physically erase the memory occupied by files that are no longer needed, so that it can be made available to other applications or new applications.

Here it is important not only to remove the file from the file tree, but also to erase the entire memory area used by the file. This is the only way to ensure that all of the file contents, which certainly could be confidential and worth protecting, are overwritten and no longer accessible to anyone. File deletion is a complicated and costly process if the memory that is no longer used when a file is deleted has to be made available to other files. Consequently, it is not fully implemented in all operating systems.

The DELETE FILE command (see Table 11.45) can be used to lock and unlock files in essentially the same way as the previously described commands. A file that is implicitly

**Table 11.45** The functionality of DELETE FILE according to ISO/IEC 7816-9

---

DELETE FILE	
Command	<ul style="list-style-type: none"> <li>• FID</li> <li>or</li> <li>DF name</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

**Table 11.46** The functionality of TERMINATE CARD USAGE according to ISO/IEC 7816-9

---

TERMINATE CARD USAGE	
Command	<ul style="list-style-type: none"> <li>• —</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

selected by this command can be completely removed from the file structure of the smart card. Whether the memory that is released as a result can be used by other files depends on the operating system. Some operating systems provide only limited free memory management functionality, in which case the memory space is effectively lost forever after this command is executed.

The ISO/IEC 7816-9 TERMINATE CARD USAGE command (see Table 11.46) is similar to the TERMINATE DF command, but it locks the entire card and thus prevents execution of any subsequent commands. In this state, the only thing the card can do is to indicate its status in the ATR. The EMV equivalent of this command is CARD BLOCK, which has similar functionality.

## 11.9 APPLICATION MANAGEMENT COMMANDS

Smart card operating systems that support downloadable program code need mechanisms for managing this code. This includes not only secure loading of the program code and associated data and keys, but also deleting the data and programs at a later date. Particularly with Java cards, managing the Java program code is an essential component of the operating system. A significant aspect of program code management is that everything occurs under the control of the card issuer, so that unauthorized parties cannot load program code into the smart cards and authorized parties can only use the memory regions allocated to their applications. After many years characterized by a diversity of concepts, including company-specific concepts, the Global Platform specification generated by the Global Platform organization [GP] has now been accepted as the worldwide standard. It is described in Section 13.13, ‘Application Management with Global Platform’, on page 485. A common feature of all commands in this category is that all loading activities and state changes are performed as atomic operations, as otherwise it would be possible for an incomplete version of the program code to be present in memory, which could lead to unpredictable results when the code was executed.

**Table 11.47** The functionality of LOAD according Global Platform

---

LOAD	
Command	<ul style="list-style-type: none"> <li>• load data</li> <li>• <i>optional</i>: data for DAP verification</li> <li>• <i>switch</i>: final command with load data</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

**Table 11.48** The functionality of INSTALL [for install] according Global Platform

---

INSTALL [for install]	
Command	<ul style="list-style-type: none"> <li>• minimum amount of volatile memory to be made available by the operating system</li> <li>• minimum amount of nonvolatile memory to be made available by the operating system</li> <li>• ...</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

The very extensive and complex Global Platform specification defines ten commands that cover the entire scope of managing code-based applications in smart cards. They are aligned to the usual ISO 7816 commands to the extent that this is appropriate. For this reason, the functionality of the SELECT, MANAGE CHANNEL and GET DATA commands specified by Global Platform is a subset of the functionality of the ISO/IEC 7816 commands, and their coding largely corresponds to the ISO/IEC standard. However, there are also several special commands, which are described below.

Suitable methods for loading and deleting applications in smart cards are also described in ISO/IEC 7816. Part 13 of this family of standards is dedicated to this subject, and it defines the corresponding commands: APPLICATION MANAGEMENT REQUEST, LOAD APPLICATION, and REMOVE APPLICATION. However, this standard has been kept very abstract and is far from being a concrete specification, with the result that the provisions of the Global Platform specification are the standard reference.

The Global Platform LOAD command (see Table 11.47) loads an application in a smart card by transferring the load file to the card. If necessary, the load data can be protected using a cryptographic method to prevent manipulation of the data and maintain data confidentiality during the transfer. The LOAD command supports layer-7 chaining to allow relatively large programs to be loaded into smart cards.

The INSTALL command (see Table 11.48) uses the results of the LOAD command and has several subfunctions. Its basic task is to configure the status of the application loaded by the LOAD command to match the subsequent phases of its life cycle. The command options INSTALL [for load], INSTALL [for install], and INSTALL [for make selectable] are used for this purpose. The body of the command APDU is always a TLV-coded record used to transfer the various parameters of the INSTALL command to the smart card. Among other things, these parameters can specify the minimum size of volatile and nonvolatile memory for the loaded Java applet. Sequence Diagram 13.1, ‘Global Platform commands’, on page 491 shows a typical command sequence for installing a new application.

**Table 11.49** The functionality of DELETE according to Global Platform

DELETE	
Command	<ul style="list-style-type: none"> <li>• ID of the data or program to be deleted</li> <li>• ...</li> <li>• <i>switch</i>: final command with load data</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

If a code-based application is no longer needed, the DELETE command can be used to delete it from the smart card. In addition to deleting program code, this general-purpose command (see Table 11.49) can be used in the Global Platform environment to delete uniquely identifiable objects in smart cards, such as load files, applications, and keys. The body of this command is fully TLV-coded, which enables it to support a large number of command options. It also supports level-7 chaining for large deletion operations.

The GET STATUS command is intended to be used to query the current status during the life cycle of security domains, load files and applications. GET STATUS is complemented by SET STATUS, which can set states in accordance with the state machines specified by Global Platform.

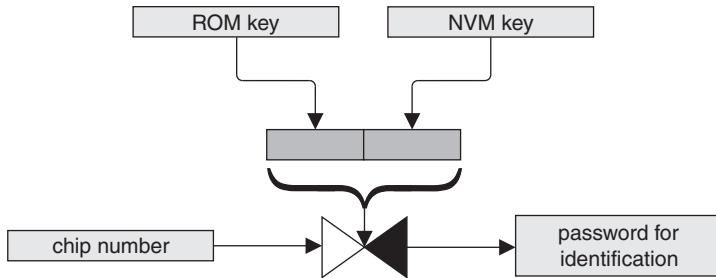
The STORE DATA command is intended to be used for loading application-specific data in code-based applications or security domains. Its operation is similar to that of PUT DATA. The data supplied by the command can have cryptographic protection. The PUT KEY command has a similar function. It is specifically designed to be used for entering and exchanging keys in the securing domains. For this purpose, it has a set of parameters for key version management.

## 11.10 COMPLETION COMMANDS

In the production of ROM-based smart card microprocessors, only the ROM is programmed during semiconductor fabrication. Except for the chip number and a card-specific key, the EEPROM remains empty. After the card body and the module are combined to produce a smart card, the operating system components that reside in EEPROM and extend the code in the ROM must be added. This is called completing the smart card. After completion, the smart card contains a complete operating system with all of its functionality.

To facilitate writing these components to the EEPROM, the ROM holds a relatively simple loader program that can write data to the EEPROM after key verification has been performed. The EEPROM memory is addressed linearly in this process, either byte by byte or page by page, using absolute physical addresses.

After all necessary data has been loaded into the EEPROM in this way, the operating system is switched over from pure ROM mode. From this point on, its functions also run from EEPROM. After the additional data in EEPROM has been verified by a checksum comparison, this switchover can be effected by a command whose execution conditions have been satisfied. A checksum comparison ensures that all data has been loaded correctly in the EEPROM. Completion does not involve especially complex functions or authentication methods, since the ROM portion of the operating system must be used for this purpose. Even the smallest error in the ROM can make it impossible to complete the smart card. Such an error would be time-consuming and, all told, very expensive.



**Figure 11.8** One of many possible ways to generate a chip-specific password for authorizing completion of the operating system

**Table 11.50** The functionality of COMPARE KEY

COMPARE KEY	
Command	• key
Response	• return code

The situation with flash-based microcontrollers is similar to that with ROM-based microcontrollers in that a unique chip number and a card-specific key are written to flash memory during semiconductor fabrication. There is also a small boot loader, which is either written to flash memory or located in a small ROM memory that also holds a variety of test routines used during semiconductor fabrication. If the secret key is known, this boot loader can subsequently be used to load the entire smart card operating system into the flash memory. The mechanisms and commands used with this method are similar to those used with ROM-based microcontrollers.

Three representative commands for completing the smart card operating system are described below. The commands used for this purpose vary greatly, depending on the operating system and chip manufacturer. Here we can only illustrate the necessary functions. However, practically all smart cards use this process or a similar process to complete the operating system. The COMPARE KEY command (see Table 11.50) checks a password sent to the card against a reference password stored in the ROM and nonvolatile memory by the semiconductor manufacturer during chip fabrication, as depicted in Figure 11.8. This key is card-specific and has a length of around 32 bytes. If the comparison is successful, subsequent load commands are permitted. Otherwise, a retry counter is incremented. When the retry counter reaches a predefined value (e.g. 3), further access to the card is blocked. The card is then ready for recycling, since the only thing it can do is generate an ATR.

After the load key has been successfully verified by COMPARE KEY, the WRITE DATA command (see Table 11.51) can write all necessary data to the nonvolatile memory. The entire nonvolatile memory region can be addressed and written bytewise with this command, which means that in addition to the operating system data, entire applications can also be loaded. This is the method normally used to load applications into smart cards with very little memory, since they do not have enough space for complex CREATE FILE commands and the associated state machines.

**Table 11.51** The functionality of WRITE DATA

---

WRITE DATA	
Command	<ul style="list-style-type: none"> <li>• data</li> <li>• memory address in NVM</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

**Table 11.52** The functionality of CHECK COMPLETION

---

CHECK COMPLETION	
Command	<ul style="list-style-type: none"> <li>• —</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

**Table 11.53** The functionality of COMPLETION END

---

COMPLETION END	
Command	<ul style="list-style-type: none"> <li>• —</li> </ul>
Response	<ul style="list-style-type: none"> <li>• return code</li> </ul>

---

If the amount of ROM in the smart card is so small that there is no room for NVM test commands, the basic functions of the test commands can be simulated using this command. This is done by repeatedly writing data to a specific memory location until the card reports a write error. If the number of write cycles is recorded in this process, the maximum number of usable write/erase cycles is known. This is the primary information expected from an NVM test command used for quality assurance.

After all the data has been written to the nonvolatile memory by the WRITE DATA command or a series of WRITE DATA commands, the content of the nonvolatile memory is checked for correctness and the completion process is concluded. Correct completion of a smart card operating system can be verified by the CHECK COMPLETION command (see Table 11.52). For this purpose, the smart card computes the checksum of the completion data and compares it with a reference value stored with the completion data. If the two values match, the completion is correct. Completion is concluded by the COMPLETION END command (see Table 11.53). After successful execution of this command, a card reset is usually triggered to reinitialize the operating system and enable it to attain a new state in its life cycle.

Sequence Diagram 11.9 on page 384 illustrates the specified sequence of commands for completing a smart card. This process is supervised by a state machine to ensure that only the indicated commands can be executed, and only in the precisely defined sequence.

**Table 11.54** The functionality of DELETE NVM

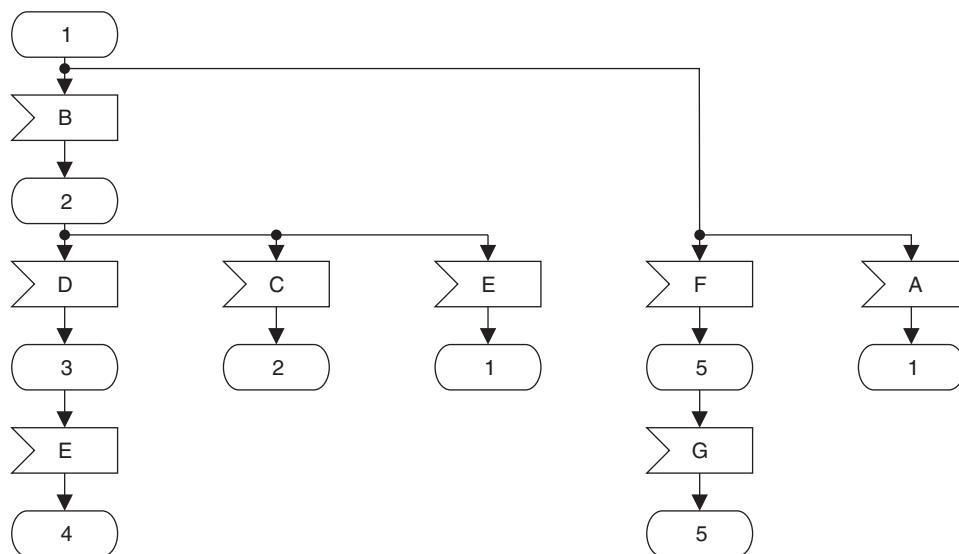
DELETE NVM	
Command	• —
Response	• return code

Terminal (IFD)		Smart card (ICC)
<b>COMPARE KEY</b>		
<i>Command</i> [key for completion]	→	<i>command processing</i>
IF (return code = OK)	←	<i>Response</i> [return code]
THEN command successfully executed		
ELSE command failed		
<b>WRITE DATA</b>		
<i>Command</i> [data    address]	→	<i>command processing</i>
IF (return code = OK)	←	<i>Response</i> [return code]
THEN command executed successfully		
ELSE command failed		
<i>iterated execution of the WRITE DATA command</i>		
<b>CHECK COMPLETION</b>		
<i>Command</i> []	→	<i>command processing</i>
IF (return code = OK)	←	<i>Response</i> [return code]
THEN command executed successfully		
ELSE command failed		
<b>COMPLETION END</b>		
<i>Command</i> []	→	<i>command processing</i>
IF (return code = OK)	←	<i>Response</i> [return code]
THEN command executed successfully		
ELSE command failed		

**Sequence Diagram 11.12** Example of a typical completion process

For tests and experiments, it is often desirable to be able to delete a previous completion. This is usually not possible, but a few operating systems include a command that provides this function. The DELETE NVM command (see Table 11.54) can undo the completion of the smart card operating system. With this command, it is not necessary to use a new smart card for each new completion. This command is used exclusively for test purposes during the development of operating systems and applications.

Sequence Diagram 11.12 shows an example of a typical completion process. A typical state machine for controlling this process is shown in Figure 11.9.



**Figure 11.9** A typical state machine for the completion of a smart card operating system. It has the following states: (1) initial state after a smart card reset; (2) smart card ready for loading data in EEPROM; (3) smart card completed; (4) initial state of the smart card after completion; (5) smart card irreversibly blocked. The state transitions are triggered by the following events: (A) all commands except COMPARE KEY; (B) successful execution of COMPARE KEY; (C) WRITE DATA; (D) COMPLETION END; (E) smart card reset; (F) three successive failures of COMPARE KEY; (G) all commands and reset

## 11.11 COMMANDS FOR HARDWARE TESTING

During initialization, the smart card operating system tests various hardware components, both implicitly and explicitly. The commands described here go far beyond the self-test routines integrated directly in the operating system. As part of end-to-end production quality assurance, specific tests must be performed on the critical components of the microcontroller. These tests are strongly focused on the nonvolatile memory (EEPROM or flash memory) because it is known to be the most common source of problems. Correct operation of the processor is implicitly verified when the terminal receives a correct ATR.

As there is no standard for test commands, their functionality and coding are determined by the producer of the operating system, and in some cases they depend directly on the operating system.

The test commands may be stored permanently in ROM, although it is common practice to load test commands in the nonvolatile memory as necessary and run them from this memory. Obviously, this can lead to problems if the nonvolatile memory is defective. The advantage of this approach is that it makes more space available in ROM. In the interest of operating system security, the use of test commands must be restricted to the phase before completion. This means that all test commands in the card are blocked after initialization or even after personalization. Possible exceptions to this are RAM testing and computing the checksum of the ROM or nonvolatile memory, since these commands do not impair security. Several commands that can be used for intensive hardware testing are described below.

**Table 11.55** The functionality of TEST RAM

TEST RAM	
Command	• —
Response	• return code

**Table 11.56** The functionality of CALCULATE EDC

CALCULATE EDC	
Command	• <i>switch</i> : ROM or NVM
Response	• checksum • return code

A defective RAM would cause a complete operating system crash even before the ATR could be sent, although it is possible for a few RAM bits or bytes to be defective. This would only affect certain functions. TEST RAM (see Table 11.55) checks the entire RAM and sends a corresponding response to the terminal. In this test, all available bytes must be written and be read using a variety of test patterns. A typical test consists of writing alternating memory locations with ‘55’ and ‘AA’, since these two hexadecimal values form a checkerboard pattern at the bit level. Another effective technique is the wave test, in which the RAM is first written and read from the bottom to the top and then from the top to the bottom. The specific implementation of this test depends on the operating system. Sometimes this test is omitted entirely.

CALCULATE EDC (see Table 11.56) is a very simple test that computes the checksum (EDC) of the entire ROM or defined portions of the nonvolatile memory and returns it to the terminal. This is one way to determine whether the ROM mask has changed or any cells in the nonvolatile memory have flipped. Testing of the nonvolatile memory includes only the static areas that cannot be intentionally changed during the lifetime of the card. The terminal compares the checksum received from the card with a reference value and decides whether the memory contents are still intact.

TEST NVM (see Table 11.57) performs an endurance test by repeatedly updating the contents of the nonvolatile memory. The smart card receives two patterns, which it writes alternately to an area of memory. The boundaries of this area and the number of write attempts can be specified in the command within certain limits. Naturally, the operating system checks the memory contents for errors after each write operation.

The number of write cycles is supplied to the smart card as a command parameter so that the test can be performed entirely inside the smart card, which is much faster than sending a sequence of individual commands. If the card detects a write error, it returns the number of completed write cycles and the address of the error to the terminal. The TEST NVM endurance test command can be used not only for destructive testing of the nonvolatile memory, but also for writing data to freely chosen regions of the memory. In the latter case, the number of write cycles is set to 1.

The COMPARE NVM (see Table 11.58) is used to test whether a pattern written to the nonvolatile memory is retained in memory. This command is primarily used in combination

**Table 11.57** The functionality of TEST NVM

TEST NVM	
Command	<ul style="list-style-type: none"> <li>● pattern 1</li> <li>● pattern 2</li> <li>● number of write cycles</li> <li>● start address in NVM</li> <li>● end address in NVM</li> </ul>
Response	<ul style="list-style-type: none"> <li>● number of write cycles completed (if errors occurred)</li> <li>● error address (if errors occurred)</li> <li>● return code</li> </ul>

**Table 11.58** The functionality of COMPARE NVM

COMPARE NVM	
Command	<ul style="list-style-type: none"> <li>● comparison value</li> <li>● memory address</li> </ul>
Response	<ul style="list-style-type: none"> <li>● return code</li> </ul>

**Table 11.59** The functionality of DELETE NVM

DELETE NVM	
Command	<ul style="list-style-type: none"> <li>● —</li> </ul>
Response	<ul style="list-style-type: none"> <li>● return code</li> </ul>

with TEST NVM to check data retention in nonvolatile memory at various temperatures. This is done by writing a specific pattern to several memory pages, placing the smart card in an environmental test chamber for a certain length of time, and then checking whether the pattern is still intact.

In principle, TEST NVM can also be used to erase the entire nonvolatile memory by setting the start and end addresses to the appropriate values. However, many operating systems have a separate command for this purpose, which here goes by the name DELETE NVM (see Table 11.59). It can be used to erase the entire nonvolatile memory in a single operation in order to clear the entire memory contents.

This command is employed for only two purposes. The first is to restore the nonvolatile memory to a defined state with a minimum of effort after several tests have left their test data in the memory. The second is to reduce the time required to initialize or complete the operating system. These processes can be accelerated by first executing DELETE NVM because this eliminates the need to erase memory pages before writing new data, which is often necessary with nonvolatile memory.

**Table 11.60** The functionality of GET RESPONSE according to ISO/IEC 7816-4 and TS 51.011

GET RESPONSE	
Command	<ul style="list-style-type: none"> <li>● amount of data to be sent</li> </ul>
Response	<ul style="list-style-type: none"> <li>● data</li> <li>● return code</li> </ul>

**Table 11.61** The functionality of ENVELOPE according to ISO/IEC 7816-4

ENVELOPE	
Command	<ul style="list-style-type: none"> <li>● command APDU</li> </ul>
Response	<ul style="list-style-type: none"> <li>● response APDU</li> <li>● return code</li> </ul>

## 11.12 COMMANDS FOR DATA TRANSMISSION

In theory, transmission protocols should be designed to be fully independent of data and commands in the application layer. This independence is also prescribed by the OSI layer model. Unfortunately, there is a gap here between the aspirations of theory and the realities of practice. There are two commands whose sole purpose is to allow the application layer to utilize transport mechanisms, namely GET RESPONSE (see Table 11.60) and ENVELOPE (see Table 11.61). There is another command, MANAGE CHANNEL, whose function is not used by the application layer alone.

With the T = 0 protocol, it is not possible to send a block of data to the smart card and receive a block of data back from the smart card in a single command–response cycle.<sup>7</sup> This means that the protocol does not support Case 4 commands, which are nevertheless often used. It is thus necessary to use a work-around with the T = 0 protocol. This operates in a simple manner. The Case 4 command is first sent to the card, and if it is successful, a special return code is sent to the terminal to inform the terminal that the command has generated data that is waiting to be retrieved. The terminal then sends a GET RESPONSE command to the smart card and receives the data in the response. This completes the actual command–response cycle of the first command. As long as no command other than GET RESPONSE is sent to the card, response data can be requested multiple times.

If commands are fully encrypted as part of secure messaging, problems can occur with the T = 0 protocol because it requires both an unencrypted instruction byte and an unencrypted L<sub>e</sub> byte. The ENVELOPE command is used to get around this restriction by embedding a complete APDU, with its header and data section, in the data section of the ENVELOPE command APDU. The latter APDU can be encrypted without any restrictions and transmitted using any protocol. The same technique is used with the response generated by the smart card,

<sup>7</sup> See also Section 9.3.1, ‘The T = 0 transmission protocol’, on page 255

**Table 11.62** The functionality of MANAGE CHANNEL according to ISO/IEC 7816-4

MANAGE CHANNEL	
Command	<ul style="list-style-type: none"> <li>• <i>switch</i>: open/close logical channel</li> <li>• IF (specific channel wanted) THEN (logical channel number)</li> </ul>
Response	<ul style="list-style-type: none"> <li>• logical channel number (if a new logical channel was opened)</li> <li>• return code</li> </ul>

which is embedded in the APDU of the ENVELOPE command. The ENVELOPE command is also used to implement proactive commands<sup>8</sup> for the SIM application toolkit (SAT or USAT). This corresponds to the previously described scheme, with the exception that the data is not encrypted.

Logical channels can be used to allow up to 19 applications (according to ISO/IEC 7816-4) in a single smart card to be accessed concurrently and independently of each other.<sup>9</sup> Two bits in the class byte of MANAGE CHANNEL (see Table 11.62) are used to associate the commands with the corresponding applications. The smart card must be explicitly informed that a new channel will be used before it is actually used. This is done with the MANAGE CHANNEL command. It informs the smart card that an additional logical channel is needed. The channel number can be specified explicitly, or the smart card can send the number of a free logical channel to the terminal in response to a request.

If a new logical channel is opened from the standard channel (channel 0), the behavior of the smart card with respect to the new channel is the same as after a reset, which means that the MF is selected and the security state is the base (initial) state. If a new logical channel is opened from a channel other than channel 0, the current DF selection and current security state are retained. After a logical channel is closed, the associated file selection and security state are deleted.

## 11.13 DATABASE COMMANDS (SCQL)

In the beginning, smart cards were used as identification media and thus represented typical single-user systems. However, the sophisticated security and access control mechanisms of multiapplication smart cards can certainly be used to configure and use smart cards as multiuser systems, although the cost and effort involved should not be underestimated. The cryptographic protective mechanisms needed for this can also quickly reach a worrisome level of complexity.

This can be regarded as analogous to the various file structures. In theory, a record-oriented telephone directory can be built using a file with a transparent structure, but it is much easier to use a file with a linear variable structure.

This is why ISO/IEC 7816-7 defines a subset of the SQL (structured query language) database query language as standard for smart cards, and why this subset is included in the product lines of several operating system producers. The subset of SQL for smart cards defined in the ISO/IEC 9075 standard is called ‘structured card query language’ (SCQL).

<sup>8</sup> See also Section 19.4.4, ‘TMSI’, on page 811

<sup>9</sup> See also Section 8.5, ‘Logical Channels’, on page 233

table		
column 1	column 2	column 3
row 1	row 1	row 1
row 2	row 2	row 2
row 3	row 3	row 3
row 4	row 4	row 4
row 5	row 5	row 5

view 1 (static)	
column 1	column 2
row 1	row 1
row 2	row 2
row 3	row 3
row 4	row 4
row 5	row 5

view 2 (dynamic)		
column 1	column 2	column 3
row 3	row 3	row 3
row 5	row 5	row 5

**Figure 11.10** An SCQL table with three columns and five rows. A static view and three dynamic views have been generated for this table. View 1 is static and shows columns 1 and 2 in their entirety. View 2 is dynamic and shows the rows in all three columns that satisfy a certain condition (not shown here)

table: acquaintances of Louis Wu		
name	tribe	protector
Teela Brown	Ball	yes
Nessus	Puppet	no
Bram	Vampire	yes
Chmee	Kzinti	no
Akolyth	Kzinti	no
Prill	Machine	no
Vala	Machine	no

view: protectors	
name	tribe
Teela Brown	Ball
Bram	Vampire

view: Kzinti	
name	
Chmee	
Akolyth	

view: name	
name	
Teela Brown	
Nessus	
Bram	
Chmee	
Akolyth	
Prill	
Vala	

**Figure 11.11** A sample SCQL table with the columns ‘Name’ and ‘Tribe’, the attribute ‘Protector’, and seven entries. Three views are also shown. Different access privileges for the various views can be defined in SCQL

One of the main application areas for smart cards with SCQL capability could be the health care industry, where a variety of people and organizations need to be able to access data with a corresponding variety of read and write privileges. However, there are presently no major applications that use SCQL.

SCQL, like SQL, supports table-oriented database systems. As illustrated by the examples shown in Figures 11.10 and 11.11, the tables in such systems have a table name, a fixed number of named columns, and a variable number of rows. Logical views of these tables can be generated. If a fixed set of columns is assigned to a view, it is called a static view. By contrast, a view that encompasses a set of rows that satisfy a particular condition (such as rows containing the given name ‘Wolfgang’) is called a dynamic view. Combinations of static and dynamic views are also possible. Each view has its own name and can be used as a basis for reading and writing data.

An additional file structure called ‘database’ is needed in the operating system to support SCQL in smart cards. A file with this structure is called a database file (DBF). It is a database

**Table 11.63** The SCQL command PERFORM SCQL OPERATION and its functions according to ISO/IEC 7816-7

---

#### PERFORM SCQL OPERATION

---

CREATE TABLE	Creates a new table with its columns and column names
CREATE VIEW	Generates a new table view (static or dynamic)
CREATE DICTIONARY	Generates the object description table, the user description table, and the privilege description table
DROP TABLE	Deletes a table
DROP VIEW	Deletes a view
GRANT	Grants access privileges to a single user, a group of users, or all users
REVOKE	Withdraws access privileges previously granted by GRANT
DECLARE CURSOR	Specifies a cursor that references a row in a table, view, or system table
OPEN	Activates a cursor in the first row
NEXT	Moves the cursor to the next row
FETCH	Reads the row marked by the cursor
FETCH NEXT	Reads the next logical row after the row containing the cursor. The cursor is moved to the row that was read
INSERT	Adds a row to the end of a table without changing the cursor
UPDATE	Writes data to one or more fields in a table row. The row is selected by the cursor
DELETE	Deletes the table row marked by the cursor. The cursor is set to the next logical row

---

object located directly below the MF or in a DF. The database file contains the data tables and related system tables, and it can be accessed by database commands without first being selected. A DBF is a logical construct, and depending on the smart card operating system it may be distributed over several EFs.

Three system tables must be created in the DBF to manage users and privileges. The first table is the object description table, which holds information about the tables and the associated views. It is complemented by the user description table, which specifies the users of the database system. The third DBF table is the privilege description table, which specifies the privileges of the individual users with respect to the tables and views and what they are allowed to do with the tables and views.

The operations that can be performed on an SCQL table or view are read, insert, write, and delete. All these operations are governed by access privileges. A cursor is defined for read and write operations on a table or view. It points to the row that is the target of the action.

There are three basic SCQL commands: PERFORM SCQL OPERATION, PERFORM TRANSACTION OPERATION, and PERFORM USER OPERATION. Only three instruction codes (INS) are needed for these commands, since the actual SCQL operations are initiated under the control of parameter byte P2. All data in the command body and the response body consists of TLV-coded data objects. The functions of these three commands are summarized in Tables 11.63, 11.64 and 11.65 on the facing page.

**Table 11.64** The SCQL command PERFORM TRANSACTION OPERATION and its functions according to ISO/IEC 7816-7

---

#### PERFORM TRANSACTION OPERATION

BEGIN	Reserves space for a memory image for the COMMIT and ROLLBACK functions, for example to hold the contents of a table row
COMMIT	Checks all changes that have been made to a table since the last BEGIN command
ROLLBACK	Restores a table to the state it had before the last BEGIN command

---

**Table 11.65** The SCQL command PERFORM USER OPERATION and its functions according to ISO/IEC 7816-7

---

#### PERFORM USER OPERATION

PRESENT USER	Identifies a user by means of the user's ID
CREATE USER	Generates an entry for a new user
DELETE USER	Deletes a user entry

---

Although SCQL has many limitations relative to SQL, such as no sorting, no nested queries, and no joins, it has definite potential for use in new areas. Despite the fact that SCQL has been standardized since 1997 and provides technically sophisticated functionality, it has not been used in any major application to date.

## 11.14 COMMANDS FOR ELECTRONIC PURSES

EN 1546, the European standard for cross-sector electronic purses, defines six commands for electronic purses and twelve commands for security modules in terminals, which can take the form of a smart card. The basic structures of the four most important commands for smart card electronic purses are described here.<sup>10</sup> These commands can be used to operate a smart card application with functions for making cashless payments from a prepaid purse and reloading the purse. The commands for error recovery, currency conversion, parameter modification and canceling a payment, as well as all the security module commands, are not described here. The Common Electronic Purse System (CEPS) specification is also based on the commands defined in EN 1546.

EN 1546 defines a three-stage process for electronic purse transactions. In the first stage, the purse is initialized using the INITIALIZE IEP for Load command (see Table 11.66) or the INITIALIZE IEP for Purchase command (see Table 11.68). In the second stage, a command is executed to perform the actual transaction, such as loading the purse or paying with the purse. In the optional third stage, the transaction just performed is confirmed. The purse commands have direct read and write access to the files of the smart card purse application. These files hold the purse balance, log entries, and various parameters. The individual steps of a purse

<sup>10</sup> Command sequences and general system structures of electronic purse systems are described in detail in Section 18.3.1 on page 760

**Table 11.66** The functionality of INITIALIZE IEP for Load according to EN 1546-3

INITIALIZE IEP for Load	
Command	<ul style="list-style-type: none"> <li>• load amount (<math>M_{LDA}</math>)</li> <li>• currency code (<math>CURR_{LDA}</math>)</li> <li>• PPSAM descriptor (PPSAM)</li> <li>• random number (R)</li> <li>• user-defined data (DD)</li> </ul>
Response	<ul style="list-style-type: none"> <li>• purse provider identifier (PPIEP) (<math>PP_{IEP}</math>)</li> <li>• IEP identifier</li> <li>• cryptographic algorithm used (<math>ALG_{IEP}</math>)</li> <li>• expiry date (<math>DEXP_{IEP}</math>)</li> <li>• purse balance (<math>BAL_{IEP}</math>)</li> <li>• IEP transaction number (<math>NT_{IEP}</math>)</li> <li>• key information (<math>IK_{IEP}</math>)</li> <li>• signature <math>S_1</math></li> <li>• return code (<math>CC_{IEP}</math>)</li> </ul>

**Table 11.67** The functionality of CREDIT IEP according to EN 1546-3

CREDIT IEP	
Command	<ul style="list-style-type: none"> <li>• key information (<math>IK_{PPSAM}</math>)</li> <li>• signature <math>S_2</math></li> <li>• user-defined data (DD)</li> </ul>
Response	<ul style="list-style-type: none"> <li>• signature <math>S_3</math></li> <li>• return code (<math>CC_{IEP}</math>)</li> </ul>

transaction are executed using the commands described below. For each command, the exact internal process in terms of functions as well as the sequence of the individual steps (which is specified in detail in EN 1456-2) are described such that all implementations have similar internal processes.

The INITIALIZE IEP command can be used for several purposes. A parameter selects the type of transaction to be initialized: purse loading, purchasing, or another type. In the first step of a load transaction, the INITIALIZE IEP for Load command initiates loading (crediting) of the purse in the smart card. The supplied data, such as the currency code and the amount to be loaded, is checked in the card to see whether it matches the values specified in the parameter files. Freely definable data (user-defined data) can also be stored in a log file. Next, a transaction counter is incremented and signature  $S_1$  is computed from various data (such as the current balance and the expiry date) in order to protect this data against manipulation when it is transferred to the terminal.

In the second step of a load transaction, the card essentially receives information about the keys to be used and signature  $S_2$  in the CREDIT IEP command (see Table 11.67). This information comes from the security module in the terminal. It serves to protect the data against manipulation and enable the smart card to authenticate the security module. The smart card

**Table 11.68** The functionality of INITIALIZE IEP for Purchase according to EN 1546-3

INITIALIZE IEP for Purchase	
Command	• —
Response	<ul style="list-style-type: none"> <li>• purse provider identifier (PPIEP) (<math>PP_{IEP}</math>)</li> <li>• IEP identifier</li> <li>• cryptographic algorithm used (<math>ALG_{IEP}</math>)</li> <li>• expiry date (<math>DEXP_{IEP}</math>)</li> <li>• purse balance (<math>BAL_{IEP}</math>)</li> <li>• currency code (<math>CURR_{LDA}</math>)</li> <li>• authentication mode (<math>AM_{IEP}</math>)</li> <li>• IEP transaction number (<math>NT_{IEP}</math>)</li> <li>• key information (<math>IK_{IEP}</math>)</li> <li>• signature <math>S_1</math></li> <li>• return code (<math>CC_{IEP}</math>)</li> </ul>

**Table 11.69** The functionality of DEBIT IEP according to EN 1546-3

DEBIT IEP	
Command	<ul style="list-style-type: none"> <li>• PSAM identifier (PSAM)</li> <li>• PSAM transaction number (<math>NT_{PSAM}</math>)</li> <li>• amount to be debited (<math>M_{PDA}</math>)</li> <li>• currency code (<math>CURR_{LDA}</math>)</li> <li>• key information (<math>IK_{PSAM}</math>)</li> <li>• signature <math>S_2</math></li> <li>• user-defined data (DD)</li> </ul>
Response	<ul style="list-style-type: none"> <li>• signature <math>S_3</math></li> <li>• return code (<math>CC_{IEP}</math>)</li> </ul>

was already authenticated with respect to the security module in the terminal by the previous INITIALIZE IEP for Load command. After successfully verifying signature  $S_2$ , the smart card increases the purse balance, updates the current entry in the log file, and generates signature  $S_3$  for confirmation. This signature is then received by the security module in the terminal as confirmation that the amount to be loaded has been credited correctly.

The second process described here, with the corresponding commands, is paying for a purchase with an electronic purse. This transaction is again initiated by the INITIALIZE IEP command, this time with the ‘for Purchase’ option (see Table 11.68). The smart card does not receive any data with this command, but it does increment the transaction counter in the card. Next, signature  $S_1$  is computed from data such as the expiry date, the transaction counter, and the IEP identifier. The information protected by this signature is then sent to the terminal along with some other information.

The actual payment transaction is performed by the DEBIT IEP command (see Table 11.69). It sends the electronic purse in the smart card information about the amount to be debited and current key versions, along with a signature. This signature is used to verify the authenticity of

the security module in the terminal in the same manner as for loading the purse. If verification is successful, the purse balance is decreased, the purchase transaction log is updated, and another signature ( $S_3$ ) is generated to confirm the entire process. This signature is placed in the response to the DEBIT IEP command. It serves as confirmation to the security module in the terminal that the amount was properly debited in the smart card.

Here we have only presented a brief summary of the four most important commands for electronic purses as specified in EN 1546. This standard is extremely comprehensive and includes many options and alternatives for system design, which naturally can also affect the command structure.<sup>11</sup>

## 11.15 COMMANDS FOR CREDIT AND DEBIT CARDS

The EMV specification<sup>12</sup> for smart cards used in financial transaction systems specifies two commands specifically designed for financial transactions, along with several variants on ISO/IEC 7816 commands. In principle, these two extremely flexible commands could be used to implement an electronic purse in a smart card. However, their intended use lies in the realm of credit and debit transactions, which is why aspects related to these two applications are described here.

The GET PROCESSING OPTIONS and GENERATE APPLICATION CRYPTOGRAM commands are based on TLV-coded data objects in the data section of the command and the response. This provides a considerable variety of alternatives and options, which can be utilized in individual applications as necessary.

The GET PROCESSING OPTIONS command (see Table 11.70) initiates a payment transaction. It sends the terminal's processing options data object list (PDOL), which contains TLV-coded data for processing the rest of the payment transaction, to the smart card. This data could be the transaction amount, for example. In its response, the card returns a BER-TLV coded data object containing the application interchange profile (AIP), which describes the functions supported by the smart card, and the application file locator (AFL).

The second command for payment transaction processes in a smart credit card is GENERATE APPLICATION CRYPTOGRAM (see Table 11.71). TLV coding is used for the data in the command and response APDUs of this command. It sends all data necessary for a payment transaction to the card and specifies the desired application cryptogram. The card uses the supplied data and its stored data to determine how to proceed with the transaction, and it returns the application cryptogram to the terminal in the response. In the simplest case, this may be the transaction cryptogram. This concludes the payment transaction process.

The application cryptogram returned by the smart card may contain an authorization request instead of a transaction cryptogram. If in the course of determining how to proceed with the transaction the smart card decides that online authorization is necessary, the application cryptogram that it returns to the terminal includes a request to the terminal's higher-level authorization center. After this request is processed by the authorization center, the smart card receives the resulting data in a second GENERATE APPLICATION CRYPTOGRAM

<sup>11</sup> See also Section 18.3.1, 'Inter-sector electronic purses compliant with EN 1546', on page 760

<sup>12</sup> See also Section 18.4, 'EMV Application', on page 776

**Table 11.70** The functionality of GET PROCESSING OPTIONS according to EMV

GET PROCESSING OPTIONS	
Command	<ul style="list-style-type: none"> <li>• processing options data object list (PDOL)</li> </ul>
Response	<ul style="list-style-type: none"> <li>• application interchange profile (AIP)</li> <li>• application file locator (AFL)</li> <li>• return code</li> </ul>

**Table 11.71** The functionality of GENERATE APPLICATION CRYPTOGRAM according to EMV

GENERATE APPLICATION CRYPTOGRAM	
Command	<ul style="list-style-type: none"> <li>• desired application cryptogram</li> <li>• transaction-related data</li> </ul>
Response	<ul style="list-style-type: none"> <li>• cryptogram information data</li> <li>• application transaction counter (ATC)</li> <li>• application cryptogram (AC)</li> <li>• return code</li> </ul>

command. It can then generate the transaction cryptogram for the payment transaction and send it to the terminal.<sup>13</sup>

## 11.16 APPLICATION-SPECIFIC COMMANDS

There are a large number of commands that are tailored to specific applications. They generally arise from the desire to minimize memory space or processing time. Most of these commands are so specific that they are not included in any standard, or they are defined in a standard for use in a particular application area.

A list of all application-specific commands would exceed the scope of this chapter. As an example of such commands, we present the only application-specific command in the TS 51.011 specification. It is called RUN GSM ALGORITHM (see Table 11.72), and it is used to simultaneously generate a dynamic, card-specific key and authenticate the card with respect to the GSM background system. This function is so specific to the GSM application that it makes no sense to include it in a general smart card standard. The command employs a GSM-specific cryptographic algorithm, which it uses with a random number and a key stored in the smart card to generate a dynamic key for the mobile telephone and an authentication value for the background system.

<sup>13</sup> See also Section 18.4, ‘EMV Application’, on page 776

**Table 11.72** The functionality of RUN GSM ALGORITHM according to TS 51.011

RUN GSM ALGORITHM	
Command	<ul style="list-style-type: none"> <li>• random number</li> </ul>
Response	<ul style="list-style-type: none"> <li>• dynamic key</li> <li>• <math>E(\text{key}, \text{random number})</math></li> <li>• return code</li> </ul>

## 11.17 COMMAND PROCESSING TIMES

Command processing time is a decisive factor in certain systems. For example, consider a motorway toll system in a country with no speed limit: the onboard unit with its inserted smart card has only a few tens of milliseconds to complete the entire automatic debiting process. Applications such as this can only be implemented with suitably fast smart cards. The tables and formulas presented here are intended to enable prediction of the time performance of an application before it is actually implemented.

Here we can mention another practical example that illustrates the unpredictable side effects of commands that are not fast enough. Some time ago, during system testing it was found that when certain smart card operating systems were used with a particular model of mobile telephone, the standby time of the mobile telephone was reduced. This was quite puzzling for everyone involved, since the extra power consumption resulting from relatively slow commands was not expected to have any noticeable effect on battery life.

After a lengthy process of analysis and measurement, a surprising conclusion emerged. The mobile telephone executed the STATUS command every 30 seconds to check for the presence of the smart card. Due to supplementary built-in security functions in the operating system, the execution time of this command was around twice as long as usual, which caused the power consumption of the smart card to be roughly twice as high as usual. However, this power consumption was not so high that it could have a significant effect on the standby time. However, with this specific model of mobile telephone the application processor (which ran at 400 MHz and had significant power consumption) was switched on while the smart card was processing the command. As a result, the longer execution time of the command in the smart card resulted in substantially higher power consumption due to the application processor, and thus a noticeably shorter standby time.

This example clearly shows the importance of the time performance of smart card applications, and that in certain cases it can have consequences that even experts cannot easily foresee.

### 11.17.1 Processing time estimation

In the high-level design stage of smart card applications or the process of designing new smart card commands, it is often necessary to estimate command processing times. Even when values acquired from experience are used, it is relatively difficult to obtain sufficiently accurate numbers from empirical estimates. In this section, we provide a collection of basic formulas for calculating command processing times for smart card operating systems that use contact interfaces for data transmission. If these formulas are used properly, they will give results that are acceptably accurate.

However, they should by no means be regarded as foolproof or perfectly accurate under all conditions. They are intended to be used to make numerical estimates with an error tolerance of the order of ten percent. We can thus highly recommend adding an appropriate safety margin in critical situations or making time measurements with a suitable smart card.

The names of the variables in the following formulas are largely self-explanatory, which makes relatively long names necessary. Unless otherwise explicitly stated, all fixed times are based on a standard clock rate of 3.5712 MHz. However, the indicated times can be adjusted for other clock frequencies by using the proportionality factor PF from Formulas (11.9) and (11.10) on page 409.

In all cases, it is assumed that the command is executed without any errors and that no errors occur during NVM operations or data transmission. With regard to data transmission, it is further assumed that command processing starts immediately after the last bit has been received, which means that the shortest possible character waiting time (CWT) has been chosen.

### 11.17.1.1 Command processing

The following three formulas form the basis for all processing time calculations. They divide the total processing time into two parts: the data transfer time and the time inside the smart card:

$$t_{\text{total}} = t_{\text{data\_transfer}} + t_{\text{ICC\_internal}} \quad (11.1)$$

$$t_{\text{data\_transfer}} = t_{\text{command\_data\_transfer}} + t_{\text{response\_data\_transfer}} \quad (11.2)$$

$$t_{\text{ICC\_internal}} \approx t_{\text{command\_interpreter}} + t_{\text{command\_execution}} \quad (11.3)$$

The time that the command interpreter needs for its activities depends only on the frequency of the applied clock signal, which is here taken to be 3.5712 MHz:

$$t_{\text{command\_interpreter}} \approx 1.5 \text{ ms} \quad (11.4)$$

$$t_{\text{command\_execution}} = t_{\text{NVM}} + t_{\text{cryptographic\_algorithm}} + t_{\text{command\_code}} \quad (11.5)$$

The exact command execution time can only be determined by detailed analysis of program execution at the machine-code level. Program branches and loops, as well as the various command options, would result in very complex formulas that would be unusable in practice. Consequently, we have simply divided commands into three groups according to their complexity.

Simple commands, such as SELECT FILE and READ BINARY, have the shortest execution time. Moderately complex commands, such as INTERNAL AUTHENTICATE, require somewhat more time for their internal processes. Highly complex commands, such as DEBIT IEP, have the longest execution time. Here it should be borne in mind that these blanket values do not include actions such as calling cryptographic algorithms or performing NVM write operations, but only the essential internal computations and queries.

$$t_{\text{command\_code}} \approx 5 \text{ ms (for simple commands)} \quad (11.6)$$

$$t_{\text{command\_code}} \approx 12 \text{ ms (for moderately complex commands)} \quad (11.7)$$

$$t_{\text{command\_code}} \approx 20 \text{ ms (for highly complex commands)} \quad (11.8)$$

**Table 11.73** Definitions and descriptions of the variables in Formulas (11.1)–(11.8)

Variable	Unit	Description
$t_{data\_transfer}$	s	Data transfer time of a command and its associated response
$t_{response\_data\_transfer}$	s	Transfer time of the response data
$t_{command\_data\_transfer}$	s	Transfer time of the command data
$t_{NVM}$	s	Time required to write data to the NVM
$t_{total}$	s	Time required to receive a command, process it, and send the response
$t_{ICC\_internal}$	s	Time required to process a command inside the smart card
$t_{command\_execution}$	s	Time required to execute a command
$t_{command\_code}$	s	Time required to execute a special routine (such as a cryptographic algorithm) for a specific command
$t_{command\_interpreter}$	s	Time required to analyze a command and call the associated program code
$t_{cryptographic\_algorithm}$	s	Time required to execute a cryptographic algorithm

**Table 11.74** Definitions and descriptions of the variables in Formulas (11.9) and (11.10)

Variable	Unit	Description
$f_{actual}$	MHz	Actual clock rate
$f_{reference}$	MHz	Reference clock rate used to determine the duration of certain clock-dependent activities
$t_{actual}$	s	Actual duration of an activity
$t_{reference}$	s	Stated value of a specific clock-dependent time
$PF$	—	Proportionality factor for clock-dependent routines

### 11.17.1.2 Proportionality factor for predefined functions

If the stated times for individual functions are based on a specific clock rate and the actual system has a different clock rate, the proportionality factor PF must be used to convert them to the correct values for the actual clock rate:

$$PF = \frac{f_{reference}}{f_{actual}} \quad (11.9)$$

$$t_{actual} = t_{reference} \times PF \quad (11.10)$$

### 11.17.1.3 NVM operations

Before data can be written to the NVM, the region of NVM to be written may first have to be erased, depending on content of the data to be updated. With some smart card microcontrollers, the page size for erasing can be different from the page size for writing. This is reflected in the following formulas.

$$t_{NVM} = t_{erase\_NVM} + t_{write\_NVM} \quad (11.11)$$

**Table 11.75** Definitions and descriptions of the variables in Formulas (11.11)–(11.13)

Variable	Unit	Description
$n$	byte	20 bytes
		Number of bytes to be written to NVM; the value must be rounded up to the actual page size
$PS_{erase}$	byte	32 bytes
$PS_{write}$	byte	4 bytes
$t_{NVM}$	s	21 ms
		Time required to write $n$ bytes to the NVM, including prior erasing if necessary
$t_{erase\_NVM}$	s	3.5 ms
$t_{write\_NVM}$	s	17.5 ms
$t_{erase\_page}$	s/byte	3.5 ms (4 bytes)
$t_{write\_page}$	s/byte	3.5 ms (4 bytes)
		Time required to write one NVM page

$$t_{erase\_NVM} = \frac{t_{erase\_page}}{PS_{erase}} \times n \quad (11.12)$$

$$t_{write\_NVM} = \frac{t_{write\_page}}{PS_{write}} \times n \quad (11.13)$$

To determine whether an NVM page must first be erased, the current page contents and the new data must be known. For conservative estimates, however, it should always be assumed that the page to be written must first be erased.

At present, microcontrollers commonly used in smart cards do not have internal clocks, so the only time reference for the operating system is the external clock signal. If the microcontroller has a maximum rated clock rate of 5 MHz, for example, all NVM write routines will be designed for this clock rate. This means that if the actual clock rate is less than the maximum value, the NVM write time will be proportionally longer. For precise calculations, this should be taken into account by using the proportionality factor. However, this depends on the maximum clock rate configured in the design of the smart card operating system, which depends on the microcontroller type, so this aspect is not taken into account here. In the future, it will anyhow not be a significant consideration because the latest microcontrollers have internal clocks and can thus perform NVM operations with fixed timing, independent of the frequency of the external clock.

#### 11.17.1.4 Data transfer

The time required to transfer the command and the subsequent response depends primarily on the amount of data to be transferred. The structures of the transmission protocol data units (TPDUs) and application protocol data units (APDUs) used for this purpose are described in detail in Sections 9.3.1, ‘The T = 0 transmission protocol’, on page 255 and 9.3.2, ‘The T = 1 transmission protocol’, on page 260.

$$t_{command\_data\_transfer} = t_{byte\_transfer} \times n_{command\_data} \quad (11.14)$$

**Table 11.76** Definitions and descriptions of the variables in Formulas (11.14)–(11.18)

Variable	Unit	Typical value	
$n_{response\_data}$	byte	—	Amount of data transferred
$n_{body}$	byte	—	Number of bytes in the command body or associated response body. If the command has a data part, it includes a one-byte or two-byte length parameter for the command body or response body.
$n_{header}$	byte	4 bytes	Number of bytes in the command header. With the T = 1 protocol, these are the CLS, INS, P1, and P2 bytes
$n_{command\_data}$	byte	—	Amount of data transferred
$n_{layer\_2}$	byte	4 bytes	Number of bytes for the transport layer (layer 2). With the T = 1 protocol, these are the NAD, PCB, LEN and EDC bytes.
$n_{trailer}$	byte	2 bytes	Amount of data necessary in the response trailer. This consists of the SW1 and SW2 bytes
$t_{byte\_transfer}$	s	—	Time required to transfer one byte.
$t_{response\_data\_transfer}$	s	—	Time required to transfer the response to a command
$t_{command\_data\_transfer}$	s	—	Time required to transfer a command
$D$	bit · $\frac{\text{byte}}{\text{MHz}}$ · ms	1 bit · $\frac{\text{byte}}{\text{MHz}}$ · ms	Bit-rate adjustment factor
$f$	MHz	3.5712 MHz	Clock rate
$F$	1	372	Clock rate conversion factor
$n$	bit	12 bits	Number of bits per byte (1 start bit, 8 data bits, 1 parity bit, 2 stop bits)
$t_{byte\_transfer}$	ms	1.25 ms	Transfer time of one byte

$$t_{response\_data\_transfer} = t_{response} \times n_{response\_data} \quad (11.15)$$

$$n_{command\_data} = n_{layer\_2} + n_{header} + n_{body} \quad (11.16)$$

$$n_{response\_data} = n_{layer\_2} + n_{trailer} + n_{body} \quad (11.17)$$

$$t_{byte\_transfer} = \frac{1}{D} \times \frac{F}{f} \times n \quad (11.18)$$

### 11.17.1.5 Calculated example: READ BINARY command

Here we present a sample estimation of the time required to process a smart command with a simple structure. We chose the READ BINARY command for this example. As general conditions, we selected the T = 1 transmission protocol with a divisor value of 372 and a clock frequency of 5 MHz. The time required for data transfer can be calculated using Formula (11.18) on page 410.

$$t_{byte\_transfer} = \frac{1}{D} \times \frac{F}{f} \times n = \frac{1}{1} \times \frac{372}{5000\,000\,\text{Hz}} \approx 0.89 \frac{\text{ms}}{\text{byte}}$$

For data transmission using the T = 1 protocol, four layer-2 bytes are needed in addition to the layer-7 data. The command header of READ BINARY has a length of four bytes (CLA, INS, P1, and P2), and the associated body contains one byte ( $L_e$ ). The response consists of the read data with a length of  $L_e$  and a trailer containing the status bytes SW1 and SW2. With this information, we can use Formulas (11.16) and (11.17) on page 410 to express the amount of data to be transmitted as a function of  $L_e$ :

$$\begin{aligned}n_{command\_data} &= n_{layer\_2} + n_{header} + n_{body} \\n_{command\_data} &= (4 + 4 + 1) \text{ bytes} = 9 \text{ bytes} \\n_{response\_data} &= n_{layer\_2} + n_{trailer} + n_{body} \\n_{response\_data} &= (4 + L_e + 2) \text{ bytes} = (L_e + 6) \text{ bytes}\end{aligned}$$

From this, we can determine the times required to transmit the command and the response using Formulas (11.14) and (11.15) on page 410, respectively:

$$\begin{aligned}t_{command\_data\_transfer} &= t_{byte\_transfer} \times n_{command\_data} \\t_{command\_data\_transfer} &= 0.89 \frac{\text{ms}}{\text{byte}} \times 9 \text{ bytes} = 8.01 \text{ ms} \\t_{response\_data\_transfer} &= t_{response\_data} \times n_{response\_data} \\t_{response\_data\_transfer} &= 0.89 \frac{\text{ms}}{\text{byte}} \times (L_e + 6 \text{ bytes}) = 0.89 \times (L_e + 6 \text{ bytes}) \frac{\text{ms}}{\text{byte}}\end{aligned}$$

This command is a simple command, so an execution time of 5 ms at a clock frequency of 3.5712 MHz is assumed. This can be adjusted to correspond to the actual 5-MHz clock frequency by using Formulas (11.9) and (11.10) on page 409:

$$\begin{aligned}\text{PF} &= \frac{f_{reference}}{f_{actual}} = \frac{3.5712 \text{ MHz}}{5 \text{ MHz}} = 0.714 \\t_{actual} &= t_{reference} \times \text{PF} = 5 \text{ ms} \times 0.714 = 3.57 \text{ ms}\end{aligned}$$

READ BINARY does not require writing data to the NVM or executing a cryptographic algorithm. Consequently, the time required for internal command processing can be calculated as follows:

$$\begin{aligned}t_{command\_execution} &= t_{NVM} + t_{cryptographic\_algorithm} + t_{command\_code} \\t_{command\_execution} &= 0 + 0 + 3.57 \text{ ms} = 3.57 \text{ ms}\end{aligned}$$

Under the additional assumption that the command interpreter needs around 1.5 ms to do its job with a 3.5712-MHz clock, we can use Formulas (11.10) on page 409 and (11.3) on

page 408 calculate the internal processing time for the command:

$$t_{actual} = t_{reference} \times PF = 1.5 \text{ ms} \times 0.714 \approx 1 \text{ ms}$$

$$t_{ICC\_internal} = t_{command\_interpreter} + t_{command\_execution} = 1 + 3.5 \text{ ms} = 4.5 \text{ ms}$$

All the values determined above can now be inserted into Formulas (11.2) and (11.1) on page 408 to yield an expression for the total processing time of the READ BINARY command as a function of the amount of data read:

$$t_{data\_transfer} = t_{command\_data\_transfer} + t_{response\_data\_transfer}$$

$$t_{data\_transfer} = 8.01 \text{ ms} + 0.89 \times (L_e + 6 \text{ bytes}) \frac{\text{ms}}{\text{byte}}$$

$$t_{total} = t_{data\_transfer} + t_{ICC\_internal}$$

$$t_{total} = 8.01 \text{ ms} + 0.89 \times (L_e + 6 \text{ bytes}) \frac{\text{ms}}{\text{byte}} + 4.5 \text{ ms}$$

$$t_{total} = (12.51 + 0.89 \times (L_e + 6 \text{ bytes})) \frac{1}{\text{byte}} \text{ ms} = (17.85 + 0.89 \times L_e) \frac{\text{ms}}{\text{byte}}$$

The result of these calculations is a reasonably good match to the empirically determined formula for READ BINARY at a clock rate of 3.5712 MHz MHz, see Formula (11.21) on page 417). The difference between the two results arises from timing differences in data transmission and the above-mentioned assumptions regarding processing times inside the operating system.

#### 11.17.1.6 Calculated example: smart card initialization

In the following numerical example, we calculate a rough estimate of the time required to initialize a smart card. Here we assume that initialization requires writing 5 KB of data to the NVM. The initialization data is transferred using the T = 1 protocol with a divisor value of 372 and a clock frequency of 3.5712 MHz.

From Formula (11.18) on page 410, the calculated transfer time for a single byte is:

$$t_{byte\_transfer} = \frac{1}{D} \times \frac{F}{f} \times n = \frac{1}{1} \times \frac{\text{bit} \times \text{byte}}{\text{MHz} \times \text{ms}} \times \frac{372}{3.5712 \text{ MHz}} \times 12 \text{ bits} \approx 1.25 \frac{\text{ms}}{\text{byte}}$$

If we assume that the initialization command has a four-byte header (CLA, INS, P1 and P2), the length parameter in the body is one byte ( $L_c$ ), 100 bytes of user data are transferred, and the response consists only of SW1 and SW2, the calculated number of bytes to be transferred for the command and response is:

$$n_{command\_data} = n_{layer\_2} + n_{header} + n_{body}$$

$$n_{command\_data} = (4 + 4 + 1 + 100) \text{ bytes} = 109 \text{ bytes}$$

$$n_{response\_data} = n_{layer\_2} + n_{trailer} + n_{body}$$

$$n_{response\_data} = (4 + 0 + 2) \text{ bytes} = 6 \text{ bytes}$$

From this, we can determine the command and response data transfer times:

$$\begin{aligned} t_{\text{command\_data\_transfer}} &= t_{\text{byte\_transfer}} \times n_{\text{command\_data}} \\ t_{\text{command\_data\_transfer}} &= 1.25 \frac{\text{ms}}{\text{byte}} \times 109 \text{ bytes} = 136.25 \text{ ms} \\ t_{\text{response\_data\_transfer}} &= t_{\text{response}} \times n_{\text{response\_data}} \\ t_{\text{response\_data\_transfer}} &= 1.25 \frac{\text{ms}}{\text{byte}} \times 6 \text{ bytes} = 7.5 \frac{\text{ms}}{\text{byte}} \end{aligned}$$

Our assumption is that 100 bytes of data must be written to the NVM. Under the additional assumption that the NVM page size is 4 bytes and the write time is 3.5 ms per page, we can determine the time required to write the data to the NVM for each command:

$$t_{\text{write\_NVM}} = \frac{t_{\text{write\_page}}}{PS_{\text{write}}} \times n = \frac{3.5 \text{ (ms/4 bytes)}}{4 \text{ bytes}} \times 25 = 3.5 \text{ ms} \times 25 = 87.5 \text{ ms}$$

We also assume that the NVM has been erased as part of microcontroller testing, so it does not have to be erased before performing write operations:

$$t_{\text{NVM}} = t_{\text{erase\_NVM}} + t_{\text{write\_NVM}} = 0 + 87.5 \text{ ms} = 87.5 \text{ ms}$$

It is not necessary to execute a cryptographic algorithm for initialization, and the command used for initialization has a simple internal structure. An execution time of around 5 ms can therefore be assumed for the command code:

$$\begin{aligned} t_{\text{command\_execution}} &= t_{\text{NVM}} + t_{\text{cryptographic\_algorithm}} + t_{\text{command\_code}} \\ t_{\text{command\_execution}} &= 87.5 + 0 + 5 \text{ ms} = 92.5 \text{ ms} \end{aligned}$$

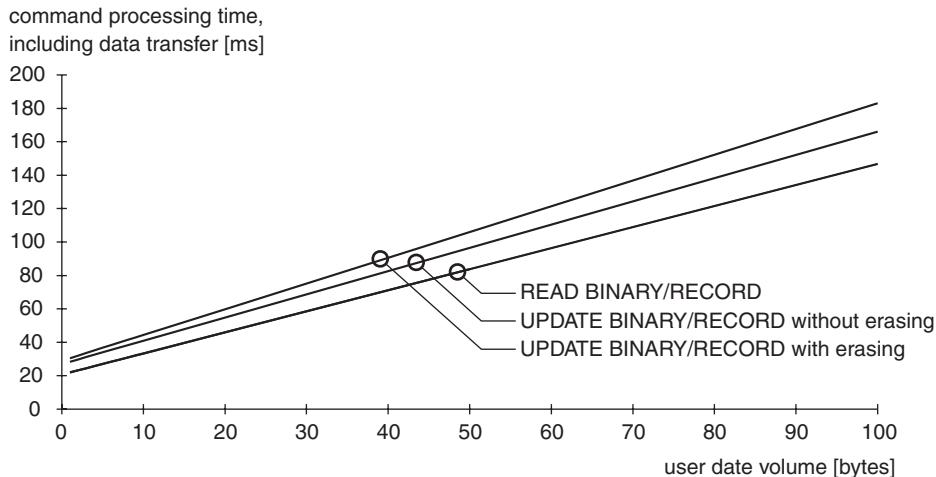
Now we only have to insert the values calculated using Formulas (11.3) and (11.2) on page 408 to complete the calculation of the time required for an initialization command with 100 bytes of initialization data. The command interpreter needs 1.5 ms on top of this:

$$\begin{aligned} t_{\text{ICC\_internal}} &= t_{\text{command\_interpreter}} + t_{\text{command\_execution}} = 92.5 + 1.5 \text{ ms} = 94 \text{ ms} \\ t_{\text{data\_transfer}} &= t_{\text{command\_data\_transfer}} + t_{\text{response\_data\_transfer}} \\ t_{\text{data\_transfer}} &= 136.5 + 7.5 \text{ ms} = 144 \text{ ms} \\ t_{\text{total}} &= t_{\text{data\_transfer}} + t_{\text{ICC\_internal}} = 144 + 94 \text{ ms} = 238 \text{ ms} \end{aligned}$$

We have thus calculated that it will take 238 ms to transfer 100 bytes of data to the smart card, write the data to the NVM, and send a response to the terminal to confirm successful command execution.

However, according to our initial assumptions a total of 5 KB of data (5120 bytes) must be written to the memory. For the sake of simplicity, we assume that this will take 52 times as long:

$$t_{\text{initialization}} = t_{\text{total}} \times 52 = 238 \text{ ms} \times 52 = 12.4 \text{ s} \approx 13 \text{ s}$$



**Figure 11.12** Processing times of smart card commands as a function of the amount of user data, including the command response data transfer time. This chart assumes the  $T = 1$  transmission protocol with a divisor value of 372, a smart card clock rate of 3.5712 MHz, and an NVM write/erase time of 3.5 ms for a four-byte page

According to our calculations, initialization will take 12.4 seconds. If we add a small safety margin, we can assume that initialization will not take longer than 13 seconds.

However, our calculation does not make any allowance for transmission errors or NVM errors that may occur during initialization. In practice, these errors must be regarded as a sort of ‘natural phenomenon’, and they can only be dealt with by using statistical estimates.

### 11.17.2 Processing times of typical smart card commands

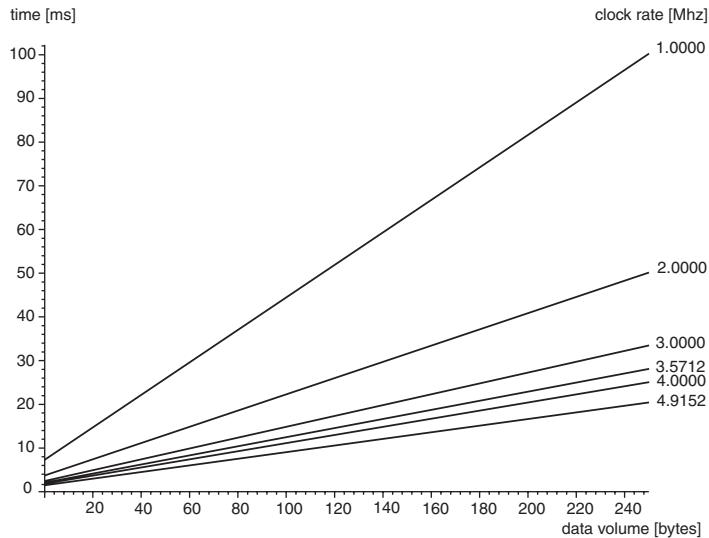
The formulas and charts presented here were obtained by measuring the processing times of actual smart card operating systems and fitting linear equations to the results. They are based on a smart card operating system with the  $T = 1$  transmission protocol, a clock rate conversion factor of 372, and a bit-rate adjustment factor  $D$  of 1. The microcontroller is clocked at 3.5712 MHz, and the write/erase time of the NVM is 3.5 ms for a four-byte page. It is also assumed that data transmission and any necessary NVM write/erase operations are free of errors. The formulas are valid for all values of  $n$  from 1 to 254.

Formulas (11.19) and (11.20) can be used to calculate the processing time of the SELECT FILE command with file selection using a two-byte FID or an  $n$ -byte DF name:

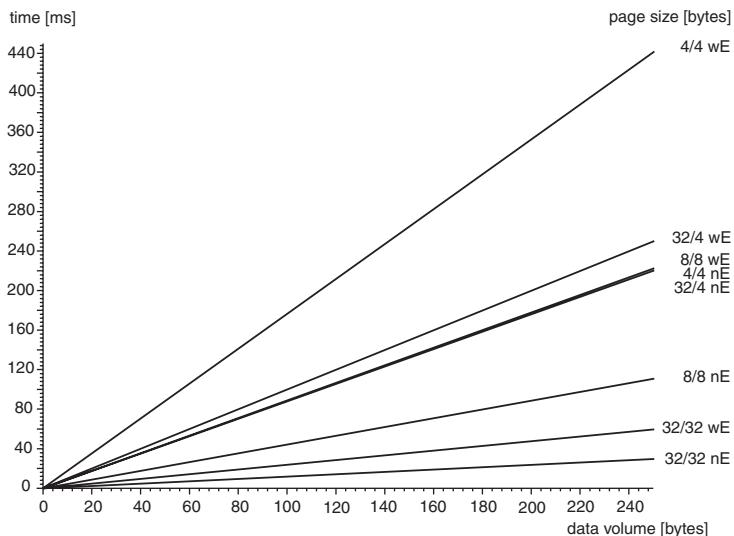
$$t_{total}(\text{SELECT FILE with FID}) \approx 23 \text{ ms} \quad (11.19)$$

$$t_{total}(\text{SELECT FILE with DF Name}) \approx (20.75 + 1.26)n \text{ ms} \quad (11.20)$$

Formulas (11.21 on page 417)–(11.24 on page 417) can be used to estimate the processing time of read commands with files having transparent or record-oriented structures. The variable  $n$  is the number of bytes to be read. These formulas can also be used when READ BINARY or READ RECORD is used with implicit file selection, since the time difference due to handling



**Figure 11.13** Data transfer time as a function of data volume and clock frequency. The values shown here assume the T = 1 data transmission protocol with a clock rate conversion factor of 31, no extra guard time ( $N = 12$ ), no chaining, XOR checksum, no transmission errors, and a Case 2 or Case 3 command such as READ BINARY or UPDATE BINARY



**Figure 11.14** NVM write time as a function of data volume with various page sizes, independent of the clock rate. Here it is assumed that no errors occur during the erase/write cycles. In the notation  $p1/p2 \times E$ ,  $p1$  is the page size for erase operations,  $p2$  is the page size for write operations,  $wE$  indicates that all pages are erased before writing, and  $nE$  indicates that the pages are not erased before writing

implicit file selection is negligible:

$$t_{total}(\text{READ BINARY}) \approx (20.77 + 1.26n) \text{ ms} \quad (11.21)$$

$$t_{ICC\_internal}(\text{READ BINARY}) \approx (2.02 + 0.01n) \text{ ms} \quad (11.22)$$

$$t_{total}(\text{READ RECORD}) \approx (20.70 + 1.26n) \text{ ms} \quad (11.23)$$

$$t_{ICC\_internal}(\text{READ RECORD}) \approx (1.95 + 0.01n) \text{ ms} \quad (11.24)$$

Formulas (11.25)–(11.32) can be used to estimate the processing time of the UPDATE BINARY and UPDATE RECORD commands with and without implicit file selection. The required time depends primarily on whether the NVM pages to be written must be erased before the write operation, so two sets of formulas are provided here. With the formulas that include erasing before writing, it is always assumed that all pages must be erased. The number of bytes to be written is indicated by  $n$ .

$$t_{total}(\text{UPDATE BINARY without erasing}) \approx (25.55 + 1.39n) \text{ ms} \quad (11.25)$$

$$t_{ICC\_internal}(\text{UPDATE BINARY without erasing}) \approx (6.8 + 0.14n) \text{ ms} \quad (11.26)$$

$$t_{total}(\text{UPDATE BINARY with erasing}) \approx (27.26 + 1.54n) \text{ ms} \quad (11.27)$$

$$t_{ICC\_internal}(\text{UPDATE BINARY with erasing}) \approx (8.51 + 0.29n) \text{ ms} \quad (11.28)$$

$$t_{total}(\text{UPDATE RECORD without erasing}) \approx (25.35 + 1.38n) \text{ ms} \quad (11.29)$$

$$t_{ICC\_internal}(\text{UPDATE RECORD without erasing}) \approx (6.7 + 0.14n) \text{ ms} \quad (11.30)$$

$$t_{total}(\text{UPDATE RECORD with erasing}) \approx (27.13 + 1.5n) \text{ ms} \quad (11.31)$$

$$t_{ICC\_internal}(\text{UPDATE RECORD with erasing}) \approx (8.4 + 0.28n) \text{ ms} \quad (11.32)$$

The charts shown in Figures 11.13 and 11.14 on the preceding page are based on these formulas. They can be used for quick initial estimates of data transfer times and NVM write times.

### 11.17.3 Typical command processing times

The numbers in Tables 11.77 and 11.78 are based on average smart card command processing times with successful execution. Time measurements were made on various types of smart cards with various operating systems. The listed values are averages; the actual values may vary significantly in specific instances, depending on the operating system. All measurements were made with the T = 1 transmission protocol, a clock rate conversion factor of 372, a clock frequency of 3.5712 MHz, an NVM write/erase cycle time of 3.5 ms for a four-byte page, and a software-based DES algorithm operating at a rate of 17 ms for each 8 bytes.

**Table 11.77** Average command processing times measured with several different smart card operating systems. The values in parentheses are the amount of data read. All other measurement conditions are described in the text

Command	Processing time without data transfer (ms)	Processing time with data transfer (ms)
READ BINARY (1 byte)	2.02	22.02
READ BINARY (2 bytes)	2.03	23.28
READ BINARY (3 bytes)	2.04	24.54
READ BINARY (4 bytes)	2.04	25.79
READ BINARY (5 bytes)	2.05	27.05
READ BINARY (10 bytes)	2.12	33.37
READ BINARY (20 bytes)	2.23	45.98
READ BINARY (50 bytes)	2.54	83.79
READ BINARY (100 bytes)	2.98	146.73
UPDATE BINARY without erasing (1 byte)	6.95	26.95
UPDATE BINARY without erasing (2 bytes)	7.01	28.26
UPDATE BINARY without erasing (3 bytes)	7.03	29.53
UPDATE BINARY without erasing (4 bytes)	7.11	30.86
UPDATE BINARY without erasing (5 bytes)	7.12	32.12
UPDATE BINARY without erasing (10 bytes)	7.33	38.58
UPDATE BINARY without erasing (20 bytes)	12.33	56.08
UPDATE BINARY without erasing (50 bytes)	18.16	99.41
UPDATE BINARY without erasing (100 bytes)	18.81	162.56
UPDATE BINARY with erasing (1 byte)	9.42	29.42
UPDATE BINARY with erasing (2 bytes)	9.51	30.76
UPDATE BINARY with erasing (3 bytes)	9.52	32.02
UPDATE BINARY with erasing (4 bytes)	9.48	33.23
UPDATE BINARY with erasing (5 bytes)	9.62	34.62
UPDATE BINARY with erasing (10 bytes)	9.85	41.10
UPDATE BINARY with erasing (20 bytes)	17.41	61.16
UPDATE BINARY with erasing (50 bytes)	25.87	107.12
UPDATE BINARY with erasing (100 bytes)	35.34	179.09

**Table 11.78** Average processing times of typical commands, as measured with several different smart card operating systems. All other measurement conditions are described in the text. The processing time of the commands marked '\*' is strongly dependent on the specific command implementation and the scope of the supported functions. Consequently, the processing times of these commands can vary widely

Command	Processing time without data transfer (ms)	Processing time with data transfer (ms)
GET CHALLENGE (8-byte random number)	26	55
CREDIT IEP *	175	222
DEBIT IEP *	235	270
EXTERNAL AUTHENTICATE *	22	51
GET CARD DATA (8 bytes)	4	33
INITIALIZE IEP for Load *	89	173
INITIALIZE IEP for Purchase *	135	201
INTERNAL AUTHENTICATE *	26	65
INVALIDATE	15	34
MUTUAL AUTHENTICATE *	95	163
REHABILITATE	15	33
SEEK	3	22
SELECT ((with an 8-byte AID))	3	32
SELECT ((with a 2-byte AID))	3	24
VERIFY ((8-byte PIN))	27	56

# 12

## Smart Card File Management

In addition to containing mechanisms for identification and authentication, smart cards are primarily data storage media. They have a decisive advantage relative to other storage media, such as diskettes, in that access to the data can be tied to certain conditions.

The first smart cards had only directly addressable memory regions, which could be used for writing or reading data. The data was accessed by specifying physical memory addresses. Nowadays, nearly all smart cards have complete, hierarchically structured file management systems with symbolic, hardware-independent addressing.

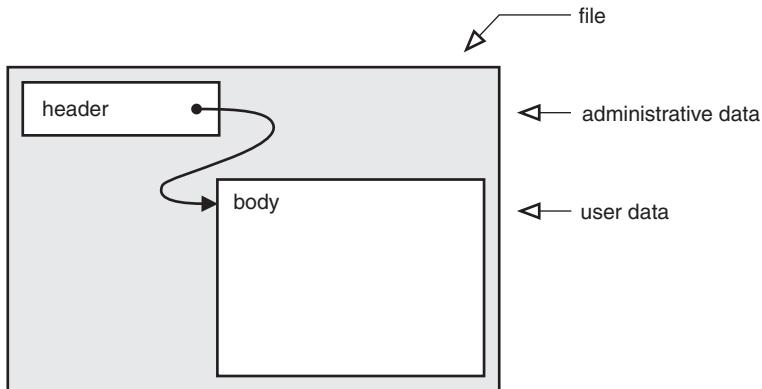
Naturally, these file management systems have certain features that are specific to smart cards. The most striking feature is that there is no man–machine interface. All files are addressed using hexadecimal codes, and all commands are strictly based on this addressing scheme, since all communication in this situation is conducted between two computers. Equally typical of these file management systems is that they are designed to use a small amount of memory. Every redundant byte is avoided if possible. As the ‘user’ is a computer in a terminal, this does not present any problems.

To minimize memory usage, in many cases there is no form of elaborate memory management. If a file is deleted – and only some operating systems have this capability – the space released does not necessarily become available for use by a newly created file. In many cases, all the files are created and loaded into the smart card when it is initialized or personalized. After this, changes to the file contents are limited.

Naturally, the characteristics of the memory that is used also affect the nature of the file management system. The memory pages of nonvolatile memory cannot be written or erased an unlimited number of times, unlike the hard disk of a PC. Consequently, there are special file attributes that allow information to be stored redundantly so it can be corrected if necessary.

### 12.1 FILE STRUCTURE

Modern file management systems for smart cards store all of the information about the file in the file itself. This principle is illustrated in Figure 12.1 on the following page. As a result, all files are divided into two parts. The first part, called the file header, contains information



**Figure 12.1** The internal structure of a file in a smart card file management system

about the structure of the file and its access conditions. The modifiable user data is stored in the second part, called the file body, which is linked to the file header by a pointer.

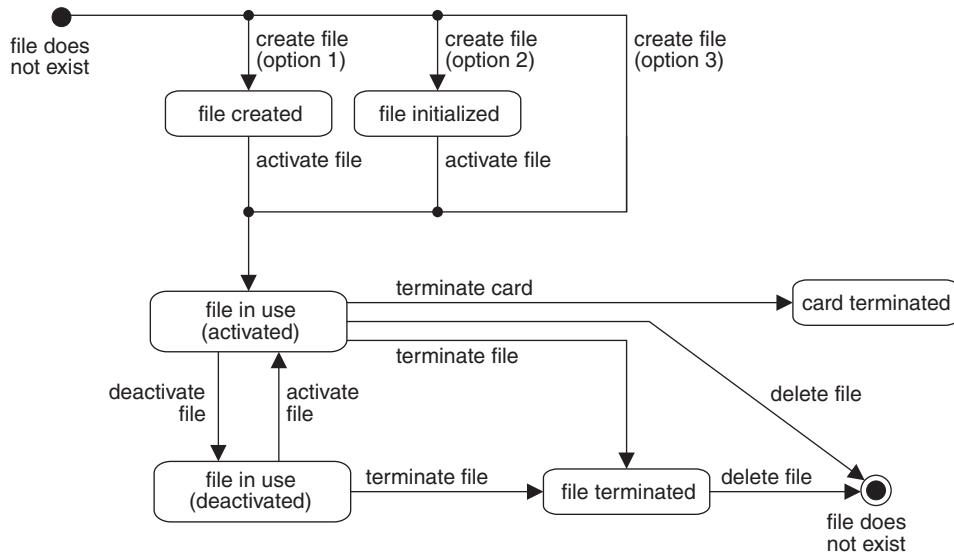
In addition to providing improved data structuring, this scheme has the advantage of providing better physical security for the data items. The page-oriented EEPROM or flash memory containing all the files allows only a limited number of write/erase cycles. The file header and file body are always located on separate memory pages. The header, which is normally altered only rarely, holds all the access conditions, so they cannot be affected by a write or erase error involving the file body. If the header and the file body were located on the same memory page, it would be possible to utilize deliberately generated write errors to alter the access conditions such that confidential information could be read from the file body.

Some smart card operating systems offer the option of addressing a file body from two different headers, which is called shared file capability. These two headers are usually located in DFs belonging to two different applications. This allows data to be shared between two applications in a technically elegant manner. In this case, it is important for the access conditions specified in the two headers to be identical.

## 12.2 THE LIFE CYCLE OF FILES

The life cycle of a file from its creation to its deletion is specified in ISO/IEC 7816-9 and illustrated in Figure 12.2 on the next page. The first stage in the life cycle of a file is file creation, which according to the previously mentioned standard can be done in three different ways. With the first option, the file is created but not immediately filled with data or activated, which means it cannot be selected externally. An ACTIVATE FILE command must be executed to enable file selection. The second option consists of creating a file and filling it at the same time with initial data. File activation is then necessary in a subsequent step. With the third option, the file is created, filled with data, and activated at the same time, so it is immediately available for use.

A common feature of all three options is that the created file is subsequently activated, which means that it can be accessed externally using standard commands such as READ BINARY.



**Figure 12.2** States and state transitions in the full life cycle of files in a smart card operating system as specified by ISO/IEC 7816-9. The different states after file creation result from the various options for the parameters of the CREATE command

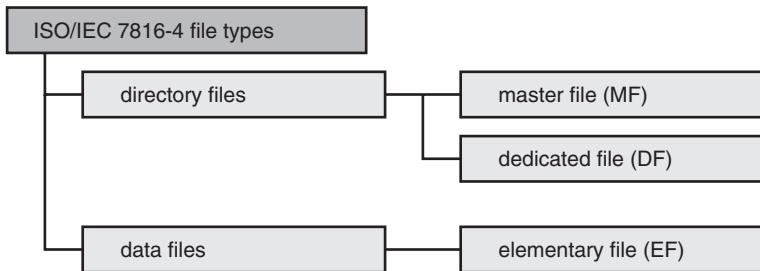
The DEACTIVATE FILE and ACTIVATE FILE commands can be used to deactivate and reactivate individual files. This makes it possible to temporarily block a file so that it cannot be used externally.

In the further course of its life cycle, a file or a directory can be permanently blocked by the TERMINATE EF or TERMINATE DF command. Although it remains in the file tree, a permanently blocked file or directory can no longer be read or written. A file can also be deleted with the DELETE FILE command, so that the memory space it occupies can be assigned to the free memory pool of the file management system and the file (and its contents) no longer exist.

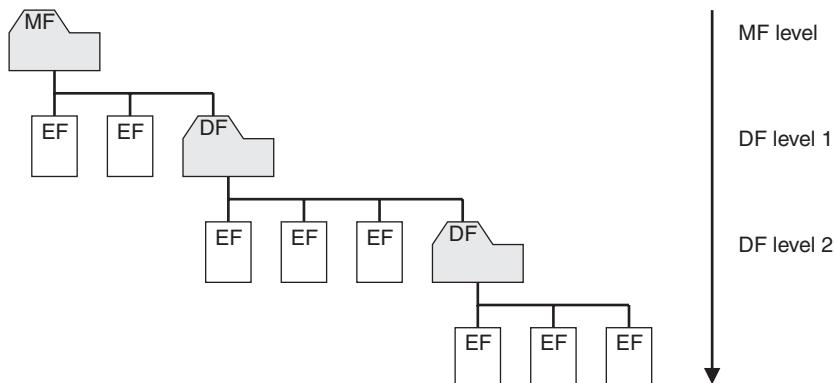
## 12.3 FILE TYPES

The structure of smart card file systems is specified in the ISO/IEC 7816-4 standard and is similar to the structure of file systems in the PC world. The main difference is that smart cards do not have application-specific files, such as specific file types for a particular word processor. Only a few standardized file structures are used in smart cards, as indicated in Figure 12.3.

There are basically two categories of files in smart cards. The first category is directory files, which are called dedicated files (DFs), and the second category consists of files that hold user data, which are called elementary files (EFs). A DF acts as a sort of folder containing other, lower-level DFs or EFs that logically belong together, as illustrated in Figure 12.4. EFs can be classified into files for the outside world (working EFs) and files used by the operating system (internal EFs). The various file types are described below, and their relationships are shown in Figure 12.4 on the next page.



**Figure 12.3** Classification of standardized smart card file structures according to ISO/IEC 7816-4



**Figure 12.4** The various file types in the file tree of a smart card, using an arrangement with two DF levels as an example

### 12.3.1 Master file (MF)

The root directory, which is selected implicitly by the operating system after the smart card is reset, is called the master file (MF). It contains the other directories and files. The MF is a special instance of a dedicated file, and it represents the entire extent of the smart card memory available for file storage. A master file must be present in every smart card.

### 12.3.2 Dedicated file (DF)

Dedicated files (DFs) may be present below the MF as necessary. They are often called directory files, although this is not the official name according to ISO/IEC 7816-4. A DF is a directory in which other files (DFs and EFs) can be grouped together. A DF may contain other DFs. In principle, there is no limit to the number of DF levels. However, smart cards rarely have more than two levels of DFs below the MF.

### 12.3.3 Application dedicated file (ADF)

The UICC specification (TS 102 221) introduced another special type of DF called an application dedicated file (ADF). This is a DF for an application, and it can be selected using the appropriate mechanism (SELECT command with an AID), but it is not located below the MF. This means that an ADF combines the characteristics of a DF and the MF. An ADF is organizationally distinct from the MF, and from a logical perspective it is not located below the MF.

### 12.3.4 Elementary file (EF)

The user data necessary for an application is located in elementary files (EFs). EFs may be located directly below the MF or below a DF. To enable data storage with logically optimized structures and the least possible memory usage, an EF always has an internal file structure. This is the main difference between EFs and files in a PC, whose internal data structures are determined by applications (such as word processors) instead of the operating system. EFs are classified into working EFs and internal EFs.

### 12.3.5 Working EF (WEF)

All application data that must be read or written by the terminal, or in other words, all data that is intended for the outside world (as seen by the smart card), is located in working EFs. Data located in a working EF is not intended to be used directly by the operating system.

### 12.3.6 Internal EF (IEF)

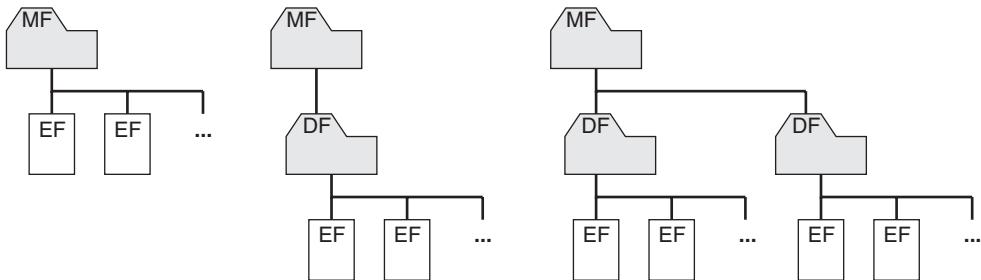
In addition to EFs for applications, there are internal system files that hold data for the operating system, data for running an application, secret keys, and program code. Access to these files is specially protected by the operating system. According to ISO/IEC 7816-4, these system files can be incorporated in the file management system in two different ways.

The first option is to store these files in the relevant application DFs as hidden EFs. Such files cannot be selected, and they are managed fully transparently by the smart card operating system in a manner similar to resource files in the Mac OS.

With the second option, these system files are assigned regular file names (in other words, FIDs) and can be selected using these names. This approach essentially corresponds to the file management system in Windows. Each system has its advantages and disadvantages, with both providing the same functionality in somewhat different manners.

## 12.4 APPLICATION FILES

According to convention, all files holding user data for a particular application (the EFs of that application) are always grouped together in a single DF. This produces a clear and readily comprehensible structure, and it makes it easy to add a new application to a smart card by creating the appropriate DF.



**Figure 12.5** Comparison of the file structures of smart cards with only one application and smart cards with several applications. The schemes commonly used in smart cards with only one application are shown on the left and in the middle, while the scheme used in smart cards with several applications is shown on the right, with a separate DF for each application

As the MF is simply a special type of DF, it is naturally possible to place all the application files directly below the MF in a single-application smart card. In a typical single-application smart card, all the EFs can be located either directly below the MF or in a single DF, as shown in Figure 12.5. Smart cards with several applications have a corresponding number of DFs, each of which holds the EFs belonging to the associated application.

Additional DFs can be placed inside these application DFs if necessary. For example, a DF placed directly under the MF could be dedicated to a traffic control application. An additional DF level in this application DF could hold the files for the supported languages, such as English and German.

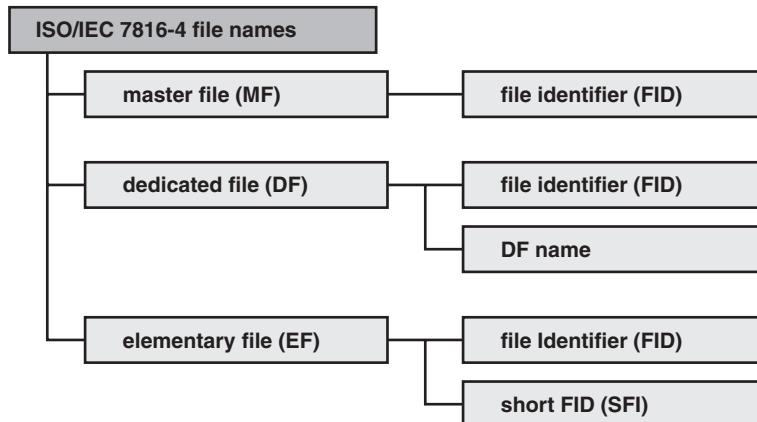
## 12.5 FILE NAMES

In modern smart card operating systems, files are without exception accessed by logical names instead of direct physical addresses. The latter approach was quite common in the smart card world during the early years, and there are still a few isolated smart cards that use this method. With simple applications that occupy precisely defined memory regions, access using direct physical addressing can save a lot of memory space. As all files are accessed by the computer in the terminal, this does not degrade user friendliness. However, direct physical addressing does not in any way correspond to the standards of modern software design, and it creates considerable problems in case of software extensions or smart card microcontrollers with different address spaces. All schemes that use logical file names are significantly better, and above all they are much easier to extend. Without question, it can be assumed that within a few years file access using logical names will be the only scheme used in smart cards with microcontrollers. However, physical addressing of files will continue to be used in memory cards for the foreseeable future.

The various types of files defined by ISO/IEC 7816-4 and their associated file names are depicted in Figure 12.6.

### 12.5.1 File identifier (FID)

The file identifier system described here is based on the ISO/IEC 7816-4 standard, and the same principle is adopted in all other international smart card standards. Every file, including directory files, has a two-byte file identifier (FID) that can be used to select the file.



**Figure 12.6** Classification of smart card operating system file names as specified in ISO/IEC 7816-4

**Table 12.1** FIDs reserved by major smart card standards

FID	Name and purpose	Standard
'2F00'	FID reserved for the directory file EF <sub>DIR</sub> , which holds application identifiers (AIDs) and the path names of associated applications	ISO/IEC 7816-4
'2F01'	FID reserved for EF <sub>ATR</sub> , which holds ATR extensions	ISO/IEC 7816-4
'2FFF'	FID reserved for file selection using a path name	ISO/IEC 7816-4
'3F00'	MF is the root directory of all files in a smart card	ISO/IEC 7816-4, TS 51.011, TS 102 221, EMV
'FFFF'	FID reserved for future use by ISO/IEC	ISO/IEC 7816-4

For historical reasons, the FID of the MF is '3F00'. This FID is reserved for the MF within the entire logical address space. The logical file name 'FFFF' is reserved for future applications and may not be used. There are also other FIDs that are reserved by the ISO standard or other standards. They are listed in Table 12.1.

The GSM application is a typical example of the fact that certain FIDs have predefined uses. In the TS 51.011 specification, the more significant byte is determined by the location of the file in the directory structure (file tree). This coding has developed historically and originates from the first French smart cards. GSM DFs have '7F' as the first (more significant) byte. The FIDs of EFs located directly below the MF have '2F' as the first byte, and the FIDs of EFs located below a DF begin with '6F'. The less significant bytes are numbered sequentially. This specification applies only to the GSM application and is not a general standard. In other situations, the full two-byte address range of the FID can be used as desired and is not subject to any restrictions.

EF<sub>DIR</sub> has an FID of '2F00'. As indicated in Table 12.2, it has a linear fixed structure and contains at least one record. Each record is in turn a constructed data object containing information about a particular application in the smart card. This typically consists of the AID and a textual designation of the application concerned. EF<sub>DIR</sub> can also hold additional data, such as the path to the application. The purpose of EF<sub>DIR</sub> is to indicate the applications present in the smart card to the terminal in a standardized format.

**Table 12.2** Typical structure of a UICC EF<sub>DIR</sub> file compliant with ISO/IEC 7816-4 and ISO/IEC 7816-5)

EF <sub>DIR</sub>	Directory EF
Description: File:	This file contains information about the applications present in the smart card FID = ‘2F00’; structure: linear fixed, file size: $n$ bytes; accesses: READ always, UPDATE depending on the application, but in general only with administrator privileges
Record coding:	byte 1: ‘61’ (application template tag) byte 2: Application template length (3 . . . 127) byte 3: ‘4F’ (AID tag) byte 4: AID length (1 . . . 16) bytes 5 . . . $n$ : AID byte $n + 1$ : ‘50’ (application label tag) byte $n + 2$ : Application label length bytes $n + 3 . . . m$ : application label in ASCII (1 . . . 16)
Example:	‘61 0E 4F 05 D2 76 00 00 60 50 05 52 61 6E 6B 6C’ ‘61’ Application template tag ‘0E’ Application template length = 14 bytes ‘4F’ AID tag ‘05’ AID length = 5 bytes ‘D2 76 00 00 60’ AID ‘4F’ Application label tag ‘05’ Application label length = 5 bytes ‘52 61 6E 6B 6C’ Application label = “Rank1”

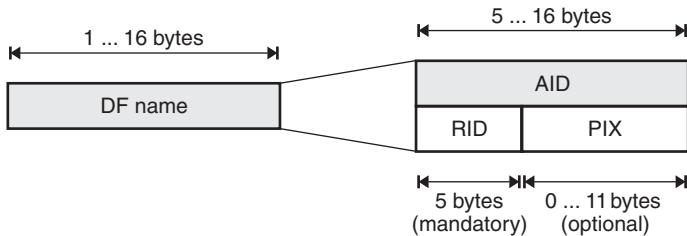
The FIDs in the file tree must be chosen such that the files can be selected unambiguously. It is thus prohibited for two different files in the same DF to have the same FID. In addition, a DF may not have the same FID as an EF located directly below it, since this would mean that the operating system would have to decide whether to select the DF first or the EF.

The following rules apply to the selection of unique FIDs:

- All DFs and EFs in the same directory must have different FIDs.
- Nested directories (DFs) may not have the same FIDs.
- An EF in a directory (MF or DF) may not have the same FID as the next higher or next lower directory.

### 12.5.2 Short file identifier (SFI)

Short file identifiers are used for implicit file selection in the immediate context of a command. Short file identifiers are optional for EFs, so they do not necessarily have to be assigned. If a short file identifier is defined, it can be passed as a command parameter for implicit file selection and has a length of only five bits. A short FID can thus assume values between 1 and 30, since the short file identifier ‘0’ addresses the current EF.



**Figure 12.7** The relationship between the DF name and the AID, which consists of an RID (registered identifier) and a PIX (proprietary application identifier extension)

**Table 12.3** Coding of the five-byte (ten-digit) registered identifier (RID)

Registered identifier (RID)	D1	D2 ... D4	D5 ... D10	Meaning
X	...	...	...	Registration category: ‘A’: international registration ‘D’: national registration
...	X	...	...	Country code according to ISO 3166
...	...	X	...	Application provider number (assigned by a national or international registration authority)

### 12.5.3 DF name

A DF comprises a set of files used by an individual application. A DF is a sort of directory or folder that can hold EFs and other DFs. In the future, the address space provided by the two-byte FID could become too small. Consequently, each DF has a DF name in addition to its FID. As specified in the ISO/IEC 7816-4 standard, the DF name has a length of 1 to 16 bytes. The DF name provides sufficient address space to allow every smart card application to be identified uniquely throughout the world.

As DF names can be freely chosen, it is possible for two different DFs to have the same DF name. Consequently, the DF name is normally used only in combination with an application identifier (AID) defined in the ISO/IEC 7816-5 standard. An AID consists of two data elements defined by the ISO standard, and it can have a length of 5 to 16 bytes. Every AID is unique worldwide. The AID effectively forms part of the DF name. Figure 12.7 shows the relationship between the AID and the DF name.

### 12.5.4 Application identifier (AID) structure and coding

The application identifier (AID) consists of two data elements. The first data element is the registered identifier (RID), which has a fixed length of five bytes as described in Table 12.3. It is assigned by a national or international registration authority and includes a country code, an application category, and a number that identifies the application provider. This numerical coding scheme means that each RID is assigned only once, so it can be used worldwide to identify a particular application.

**Table 12.4** Example of a nationally registered RID that complies with the ISO/IEC 7816-5 standard (in this case, the RID of Wolfgang Rankl)

Registered identifier (RID)			
DI	D2 ... D4	D5 ... D10	Meaning
'D'	...	...	The registration category is 'national'
...	'276'	...	The ISO 3166 country code for Germany
...	...	'00 00 60'	The application provider number assigned by the national registration authority.

Unfortunately, lists of assigned RIDs are confidential and cannot be published. Some examples of known RIDs, including the author's own RID shown in Table 12.4, are listed on the website of Wolfgang Rankl [Rankl]. Addresses of national and international RID registration authorities are also shown there.

If necessary, an application provider can suffix a proprietary application identifier extension (PIX) to the RID as an optional second part of the AID. The PIX may have a length of up to eleven bytes. It could for example consist of a serial number and version number used for application management.

## 12.6 FILE SELECTION

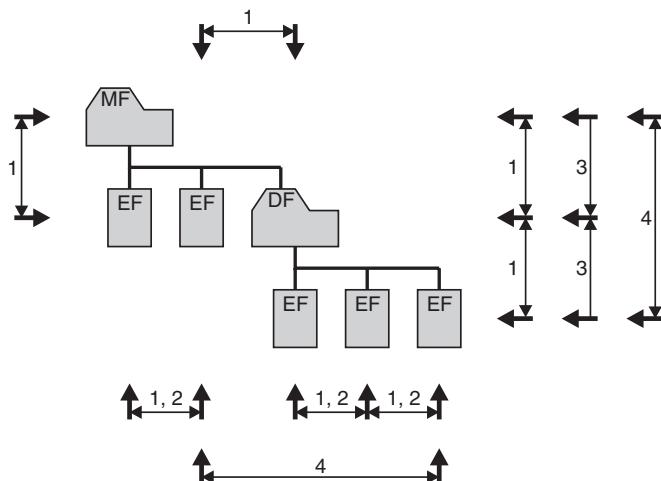
Object-oriented file management systems always require that a file be selected before it can be accessed. File selection tells the operating system which file will subsequently be addressed. Successful selection of a new file causes the previous selection to become invalid. This means that only one file can be selected at any given time.

As FIDs can be freely chosen, certain restrictions must be imposed on the free selection of files. Otherwise the operating system would be faced with the problem of deciding which file is meant when several files with the same FID are available in the file tree, which can easily happen. In order to avoid this ambiguity and remain independent of the search algorithm of the file manager in the operating system, certain limits are placed on file selection.

Things would be different if all FIDs in the file tree were always unique. In this case, it would be easy to select the desired file across several directory boundaries. However, this situation is exactly what cannot always be guaranteed. Consequently, file selection is only possible within certain boundaries, as otherwise unambiguous selection of the desired file cannot be assured. The MF can always be selected from anywhere in the file tree, since its FID is unique in the file system. Selecting a DF located in the first level below the MF is only possible from a DF at the same level or from the MF. Figure 12.8 on the facing page shows examples of several allowed and prohibited selection schemes.

### 12.6.1 Selecting directories (MF and DF)

The MF can be selected from anywhere in the file tree, either by using a specific selection option of the file selection command or by using its FID ('3F00'), which is unique in the file tree. Selecting the MF restores the selection state that exists immediately after a smart card



**Figure 12.8** Various options for selecting files in a smart card file tree. Explicit selection with a file identifier (FID) is indicated by ‘1’, while implicit selection with a short file identifier (SFI) is indicated by ‘2’. Selection using a DF name is indicated by ‘3’, while selection using the combination of a file identifier (FID) and a path is indicated by ‘4’

reset, since the MF is selected implicitly by the operating system after a reset. DFs can be selected either by their FIDs or by their DF names, which contain their registered and thus unique AIDs.

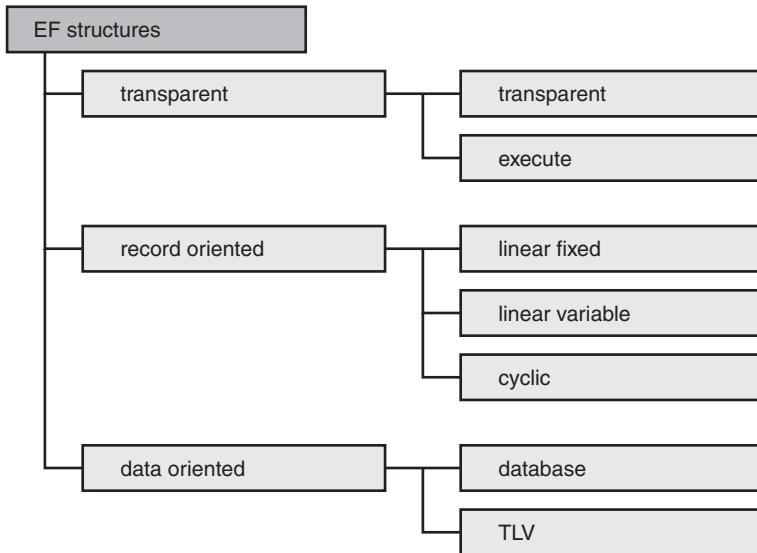
### 12.6.2 Explicit EF selection

There are basically two options for selecting EFs. With explicit selection, a specific command (SELECT) is sent to the smart card before the actual access to the file takes place. This command has a parameter that holds the two-byte FID of the file to be selected. After the file has been selected, it can be accessed by all subsequent commands.

### 12.6.3 Implicit EF selection

Implicit selection occurs when a file is selected using a short file identifier (SFI) passed as a parameter of the command that actually accesses the file. Implicit file selection is subject to several restrictions. It only works with EFs that are located in the currently selected DF or the MF. It is thus not possible to implicitly select files across directory boundaries. In addition, implicit selection can only be used with certain access commands that allow a short file identifier to be passed as a parameter, such as READ BINARY, UPDATE BINARY, READ RECORD, and UPDATE RECORD.

The main advantage of implicit selection is that it allows a file to be selected and accessed with a single command. This makes the SELECT command unnecessary in many cases, which simplifies the command sequence. Due to the reduced communication overhead, using implicit selection allows distinctly higher processing speeds to be achieved.



**Figure 12.9** Classification of EF file structures for smart card operating systems

#### 12.6.4 Selection using a path name

In addition to direct selection, the ISO/IEC 7816-4 standard provides two supplementary methods for explicit file selection using a path name. With the first method, the path from the currently selected file to the target file is passed to the operating system. The second method uses the path from the MF to the target file. Both methods are implemented in many smart card operating systems. Using these additional capabilities results in a measurable reduction in the time required to process command sequences.

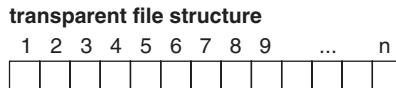
### 12.7 EF FILE STRUCTURES

In contrast to many files in PC systems, EFs in smart cards have standardized internal structures. The structure of each EF can be selected according to its intended use, which yields major benefits for the outside world because the internal structure allows the data elements to be constructed such that they can be accessed very quickly and efficiently.

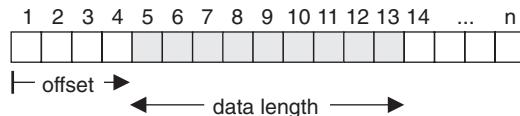
Managing these data structures requires a significant amount of program code in the smart card. For this reason, only the structures that are most often needed in practice are defined, instead of all possible orthogonal structures.

#### 12.7.1 Transparent file structure

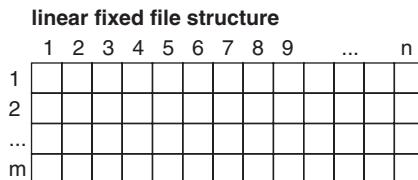
The transparent data structure is often called binary or amorphous, which expresses the fact that a transparent file has no internal structure. In terms of current programming languages, this file structure corresponds to a string that can be accessed at any desired location, as illustrated



**Figure 12.10** Transparent file structure



**Figure 12.11** Reading nine bytes from a transparent file using an offset of four bytes



**Figure 12.12** Linear fixed file structure

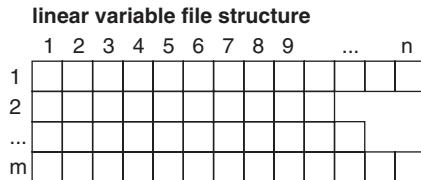
in Figure 12.10. The data contained in the file can be accessed using an offset value for reading or writing individual bytes or groups of bytes. The READ BINARY, WRITE BINARY, and UPDATE BINARY commands are used for this purpose. Figure 12.11 illustrates the process of reading nine bytes from a transparent file using an offset of four bytes.

The minimum size of a file with transparent structure is one byte. The maximum size is not specified explicitly in any standard. However, the maximum number of bytes that can be read or written in short format (255) or extended format (65 536), combined with the maximum offset value (32 767), allows a maximum size of 65 791 bytes or 98 303 bytes, respectively. Although files of this size are feasible with the memory capacities of current smart cards, transparent files are rarely larger than a few hundred bytes in practice.

The primary use for the transparent file structure is storing unstructured data or very small quantities of data. A typical example is a file holding an ID number or a digitized passport photo. However, this linear, one-dimensional data structure can also be used to simulate other data structures if necessary. Of course, this makes file access somewhat more complicated for the terminal because the parameters that define the logical structure of the data in the file must also be stored in the file.

## 12.7.2 Linear fixed file structure

A linear fixed file structure consists of series of equal-length records, as illustrated in Figure 12.12. Each record consists of a series of individual bytes. The individual records in this data structure can be accessed at random. The smallest access unit is a single record, which



**Figure 12.13** Linear variable file structure

means that it is not possible to access only part of a record.<sup>1</sup> The READ RECORD, WRITE RECORD, and UPDATE RECORD commands can be used to read data from this structure and write data to this structure.

The first record is always record number 1. The largest allowed record number is ‘FE’, or 254 in decimal notation, since ‘FF’ is reserved for future extensions. Due to the constraints of the access commands, the record length can range from 1 to 254 bytes, but all records in the file must have the same length.

A typical use for this file structure is a telephone directory with each name listed first, followed by the associated telephone number, starting at a fixed location.

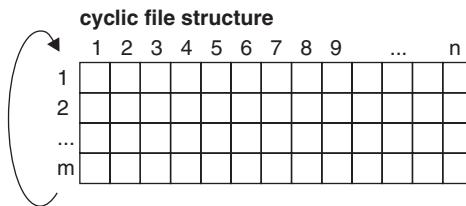
### 12.7.3 Linear variable file structure

All records in a linear fixed file structure have the same length, which in many cases results in wasteful use of memory because many types of record-oriented data objects have variable length. One example is the names in a telephone directory. The linear variable structure meets the demand for minimizing the amount of memory space that is used. With this structure, each record can have an individually defined length, as illustrated in Figure 12.13. The unavoidable consequence of this is that each record must have a supplementary field that specifies its length. This structure is otherwise similar to the linear fixed structure.

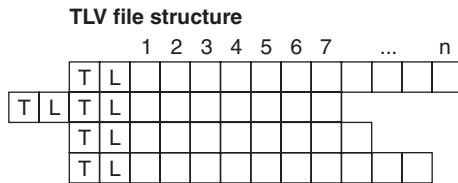
The first record of a file with linear variable structure is always record 1, and the maximum number of records in the file is 254. Due to the constraints of the access commands, each record may have an individually configurable length of 1 to 254 bytes. The access commands are the same as for the linear fixed structure, namely READ RECORD, WRITE RECORD, and UPDATE RECORD.

This file structure is preferably used when records of highly variable length must be stored and smart card memory usage must be minimized. For example, the previously mentioned telephone directory could be optimized by making each record exactly as long as the actual entry instead of using the same length for each record. However, managing this file structure requires program code in the smart card operating system as well as extra memory space to hold the length information of the individual records. For this reason, operating systems for microcontrollers with small memory capacities often do not support this file structure. The ISO/IEC 7816-4 standard explicitly allows this restriction in several profiles.

<sup>1</sup> ISO/IEC 7816-4 allows access to partial records using suitable extended commands, but up to now this capability has not been used in practice



**Figure 12.14** Cyclic file structures



**Figure 12.15** TLV file structures

### 12.7.4 Cyclic file structure

The cyclic file structure is based on the linear fixed structure and consists of a certain number of records of the same length. In addition, the EF has a pointer that always indicates the record that was last written, which is always designated record number 1. When the pointer has reached the last record in the EF, the operating system automatically resets it to point to the first record in the EF when the next write access occurs. It thus behaves the same as the hour hand of an analog clock.

If a cyclic file contains  $n$  records, the most recently written record is record number 1, the one written just before it is number 2, and the oldest record is number  $n$ . This file structure, like the other two record-oriented file structures, allows access to the first, last, previous, or next record.<sup>2</sup>

The number of records and the record length are fully analogous to the corresponding parameters of the linear fixed structure. Accordingly, due to the constraints of the write and read commands the maximum number of records is 254 and the maximum record length is 254 bytes.

This structure is typically used in smart cards for log files, since the oldest entry is always overwritten by the newest entry.

### 12.7.5 Data objects file structure

The ISO/IEC 7816-4 commands PUT DATA and GET DATA can be used to store TLV-coded data objects in smart cards and read these objects from smart cards.<sup>3</sup> The file management system supports this with a specific data structure, which is often based on a modified linear

<sup>2</sup> See also Section 11.2, ‘Read and Write Commands’, on page 358

<sup>3</sup> See also Section 11.2, ‘Read and Write Commands’, on page 358

variable or transparent file structure. In this case, the PUT DATA and GET DATA commands access these modified file structures via the file management system.

In the specifications of the SET DATA and RETRIEVE DATA commands, TS 102 221 defines TLV-structured files that are intended to be used for storing large data volumes. These data volumes can easily extend to the two-digit megabyte range. The working name for this sort of file during the specification stage has become established in regular usage, with the result that TLV-structured files are commonly called ‘large files’ in the mobile telecommunication sector.

### 12.7.6 Database file structure

The ISO/IEC 7816-7 standard defines a subset of SQL for smart cards with the name SCQL (Smart Card Query Language).<sup>4</sup> In order to store data in the file system of a smart card such that it can be read using SCQL commands, it is necessary to define a suitable file structure. The specific form of this structure is not standardized, but instead left to the individual operating system producer. A database file holds the actual user data of the database, various views of the database, the access privileges, and the user profiles.

### 12.7.7 Execute structure

The execute structure is in principle not a separate structure, since it is based on the transparent structure. It is described in the EN 7826-3 standard and offers a wealth of extension options within the scope of the operating system. The execute structure is not intended to be used for storing data, but instead for storing executable program code. A file with execute structure can be accessed using the same commands as for transparent files. Of course, this structure creates a sort of back door, since anyone who is able to write to such a file can download their own program code into the card, possibly including a Trojan horse.

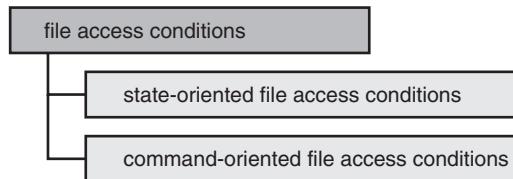
### 12.7.8 Sequence control file structure

If a smart card operating system has a command sequence controller, the information regarding the commands that can be accepted must also be stored in memory. A file whose structure is specifically adapted to this task is normally used for this purpose. However, this structure is not standardized, so every operating system that supports sequence control has its own format, which is not compatible with that of any other operating system.

## 12.8 FILE ACCESS CONDITIONS

As part of their object-oriented structure, all files include information that governs access to the file in the context of file management. This information is always stored in the file header

<sup>4</sup> See also Section 11.13, ‘Database Commands (SCQL)’, on page 399



**Figure 12.16** The two possible types of file access conditions

in coded form. Several essential aspects of the security of a smart card file management system are based on administering file access privileges.

The access conditions are specified when the file is created, and they are normally altered only rarely. The possible file access conditions are strongly dependent on the commands supported by the operating system. For example, there is little point in defining access conditions for READ RECORD if this command is not supported by the smart card operating system.

In case of the MF and DFs (in contrast to EFs), the stored access conditions do not relate to data access (read or write privileges), but instead govern operations such as creating new files. Accordingly, different types of access conditions are stored for different file types. The access conditions of EFs govern access to the data held in the EFs, while the access conditions of the MF and DFs govern the use of commands that are valid within this organizational structure.

As indicated in Figure 12.16, a distinction is made between state-oriented and command-oriented access conditions. With state-oriented conditions, the current security state is compared with the corresponding file access conditions by means of definable logical comparison operations. By contrast, command-oriented access conditions define the commands that must be executed correctly in order to obtain access. This applies in particular to authentication and identification commands.

Command-oriented access conditions are widely used in the smart card world, with the best-known example being smart cards used in the GSM system.<sup>5</sup> With command-oriented access conditions, an access table in the file contains information that specifies which commands must be successfully executed for each type of access. In many cases, the commands are further subdivided according to the various keys that are used. For example, the condition for read access to a file may specify that the user must first be identified by means of the VERIFY command with the user's PIN code. In this case, the file can be read only after this command has been successfully executed.

The advantage of this type of access conditions is their simple structure, which is sufficient to handle most application scenarios. However, using command-oriented access conditions in multiapplication smart cards often entails additional overhead and expense. For example, with operating systems that support downloadable program code the access tables in the files must be extended because it is necessary to reference specific commands explicitly in these tables. Consequently, command-oriented file access conditions are somewhat inflexible in some cases.

There are two options for the current security state: the global security state and the local security state. The global security state is the security state of the MF, which means the state of the smart card as a whole. The local security state is the state of the currently selected directory, which means the state of the DF or the state of the directory above the DF. The global security state and the local security state can both be altered by successful execution

<sup>5</sup> See also Section 3.5.13, 'Moduliertes Merkmal', on page 48

of identification and authentication commands using specific keys. When access to a file is requested, a logical comparison operation is used to compare the current security state with the state specified in the file as the access condition. If the result of this comparison is positive, the file may be accessed.

The error counter (retry counter) of the associated secret is usually reset to zero if the comparison is successful. If the comparison is not successful, the error counter is incremented, and if it reaches its maximum allowed value, the associated key is blocked.

The access conditions of a DF are fundamentally different from those of an EF. They specify the conditions under which particular commands can be executed in the directory concerned. The three most important access conditions for directory files are:

- CREATE              Create a new file
- DELETE FILE        Delete a file
- REGISTER           Register a new file

All possible types of access to an EF must be governed precisely by access privileges. The number of commands that this involves varies, depending on the operating system. Some of the most commonly used file access commands are:

- INCREASE / DECREASE      Perform calculations within a file
- INVALIDATE                Block a file
- LOCK                      Permanently block a file
- READ / SEEK              Read or search for data in a file
- REHABILITATE             Unblock a file
- WRITE / UPDATE           Write data to a file

State-oriented access conditions are used primarily in multifunctional smart card operating systems such as STARCOS and SECCOS because they are very flexible and adaptable. However, other large smart card applications such as the SIM in the GSM system use command-oriented access conditions to control file access. This access control option method requires slightly less program code and memory space than the state-oriented method. However, this comes at the price of somewhat lower flexibility.

Both methods have their advantages and disadvantages, and most arguments in favor of one or the other are ultimately based on philosophical aspects of operating system design. A method that can be used to flexibly fashion access conditions in a generalized form for controlling access to resources (including files) has now been specified in the ISO/IEC 7816-9 standard. It is described in Section 13.9, ‘ISO/IEC 7816-9 Resource Access’, on page 474.

## 12.9 FILE ATTRIBUTES

The object-oriented definition of each EF includes attributes that define supplementary properties of the file, although this depends on the operating system and the application area of the smart card. Attributes define properties of EFs that are most often related to nonvolatile memory, and they arise from the potential uncertainty of file contents and the possibility of write errors during operations on nonvolatile memory. The attributes are defined when the file is created and usually cannot be changed afterward.

### 12.9.1 WORM attribute

An attribute that is equally applicable to EEPROM and flash memory is WORM (write once, read multiple). If a file has this attribute, data can be written to the file one time only, although it can be read an unlimited number of times. This attribute can be implemented either in the hardware of the EEPROM or flash memory or as a software function. The WORM attribute is used for purposes such as writing a permanent serial number in a file. This attribute is also used in connection with personalization when information such as the cardholder's name and the expiry date are permanently written to the card.

The objective of this attribute is to protect sensitive data against modification. The best possible protection is provided when WORM access is implemented at the hardware level, which means that the nonvolatile memory has hardware protection that allows data to be written only once.

### 12.9.2 High update activity attribute

The high update activity attribute is primarily defined and used in the GSM realm. This attribute is necessary because nonvolatile memory has a limited number of write/erase cycles. A file with this attribute can be written very many times without having its data content affected by write errors.

This mechanism is implemented by having several copies of the file body available. If a write error occurs or a memory error is detected in the file by means of a checksum (EDC), the current file body is disabled and replaced by another copy. Depending on the number of reserve file bodies that are available, this mechanism can easily extend the useful life of the nonvolatile memory by a factor of ten.

### 12.9.3 EDC utilization attribute

This attribute provides special protection for especially sensitive user data in a file by means of an error detection code (EDC). This allows unintentionally altered bits (flipped bits) in the nonvolatile memory to at least be detected. If redundant storage is used together with EDC protection, it is also possible to correct errors due to flipped bits. This error correction (ECC) property is primarily used with electronic purses. Here the flipping of memory cells is equivalent to the loss of real money because the current purse balance is stored in the file. To minimize the consequences of cell flipping, this EDC or ECC attribute is used with files that hold sensitive data of this sort.

### 12.9.4 Atomic write access attribute

Smart card operating systems often include a mechanism that ensures that when a file is accessed for writing, the write operation is performed either completely or not at all.<sup>6</sup> As this mechanism more than doubles the write access time of a file, it should not be used as a matter of course for all files. A specific attribute allows this write mechanism to be used selectively with individual files.

<sup>6</sup> See also Section 13.10, 'Atomic Operations', on page 480

### 12.9.5 Concurrent access attribute

Smart card operating systems that support multiple logical channels often include a file attribute for concurrent access. This attribute explicitly allows concurrent read or write access to a file by commands received by the smart card via different logical channels that are open at the same time. It is important for this file attribute to be set specifically, since with parallel access via different channels it is possible for the file data to be modified via one channel immediately before or after it is read via another channel. If the two processes are not synchronized, the data that is read will vary depending on when the commands arrive in the smart card. Consequently, concurrent access is usually not allowed.

After a file has been selected, access via any other channel is temporarily blocked. Only after the file has been deselected is it possible for it to be accessed via another logical channel. The concurrent access attribute disables this access blocking mechanism for a specific file. In this case, the relevant applications in the terminal are responsible for the synchronization of parallel read and write operations. Of course, there is no problem if the file is only accessed for reading.

### 12.9.6 Data transfer selection attribute

Some file management systems of smart cards that have both contact and contactless interfaces support a file attribute that determines which of the two interfaces may be used for accessing the file. This makes it possible to specify for each file whether commands may access the file via the contact interface and/or via the contactless interface. In an electronic purse card, for example, this mechanism provides a simple means to restrict payment transactions to the contactless interface and card loading transactions to the contact interface.

### 12.9.7 File encryption attribute

Although smart cards are by nature very secure data storage media, there are smart card operating systems that support a file encryption attribute in addition to the other security mechanisms. If this attribute is set, the data in the file is encrypted using a symmetric cryptographic algorithm with a secret key. The necessary key is generated internally by the operating system and never leaves the smart card. With respect to the outside world, a file with this attribute appears the same as a file without the attribute. The only difference is that the data in the file is encrypted in a manner that is transparent to the outside world, and the access time is somewhat longer.

The main purpose of this attribute is to prevent the file contents from being revealed in plaintext form, even in case of a successful attack on the nonvolatile memory or an operating system malfunction. Although this attribute has a certain whiff of paranoia, it is supported by diverse operating systems for working EFs and internal EFs.

# 13

## Smart Card Operating Systems

It may seem presumptuous to refer to the program code of a smart card microcontroller as an operating system, but the software fully deserves this name. According to the DIN 44300 standard, an operating system is no more and no less than ‘the programs of a digital computer system that together with the properties of the computing system form the basis for the possible operating modes of the digital computing system, and which in particular control and monitor program execution’.

The term ‘operating system’ is thus not automatically limited to enormous programs and data volumes. Instead, it is completely independent of size, since it only refers to functionality. The term ‘operating system’ should not be exclusively associated with multi-megabyte programs for PCs or Unix computers. These operating systems are designed just as specifically for a particular man-machine interface, with a monitor, keyboard and mouse, as smart card operating systems are designed to work with a bidirectional serial interface to the terminal.

Ultimately, the decisive factor with an operating system is its functionality, which results from the interaction of mutually compatible and interdependent library routines. The fact that an operating system provides an interface between the computer hardware and the actual application software is also important, since it makes it unnecessary for the application software to directly access the hardware. This is a significant benefit because it gives the application software a certain amount of portability, although this is often very limited.

In the early 1990s, there were very few true smart card operating systems. This was in part due to the very limited memory capacity of smart card microcontrollers at the time. The usual situation then was not so much an operating system as a well-structured collection of library routines in ROM, which were utilized as necessary for a particular application when the card was completed. The structures of these systems were largely monolithic and could be modified only at considerable expense. The next generation was already structured as a layered operating system, and present-day systems still have this structure, with innumerable refinements.

One of the first true smart card operating systems was STARCOS, which was developed by Giesecke & Devrient [GD] and the Gesellschaft für Mathematik und Datenverarbeitung (GMD, now the Fraunhofer SIT [SIT]). This operating system, whose development began in 1990, allowed several applications to be stored, used and managed independently in a single smart card, even at that relatively early date.

**Table 13.1** Smart card operating systems of various producers and their Web addresses. This list is only a selection and is not intended to be complete

Operating system name	Producer
ACOS	Austria Card [AC]
BasicCard	Zeitcontrol [Zeitcontrol]
CardOS	Siemens [Siemens]
ComboOS	Trüb [Trüb]
Cryptoflex, Cyberflex, GeGKOS, MPCOS	Gemalto [Gemalto]
JCOP	NXP [NXP]
Multos	Maosco [Maosco]
SIMply, Micardo	Sagem Orga [Sagem Orga]
SIMtonIC, SIMphonIC, GIGAntIC	Oberthur [Oberthur]
STARCOS, STARSIM, SkySIM, UniverSIM, GalaxSIM, STARDC, Sm@rtCafé Expert	Giesecke & Devrient [GD]
TCOS	Telesec [Telesec]

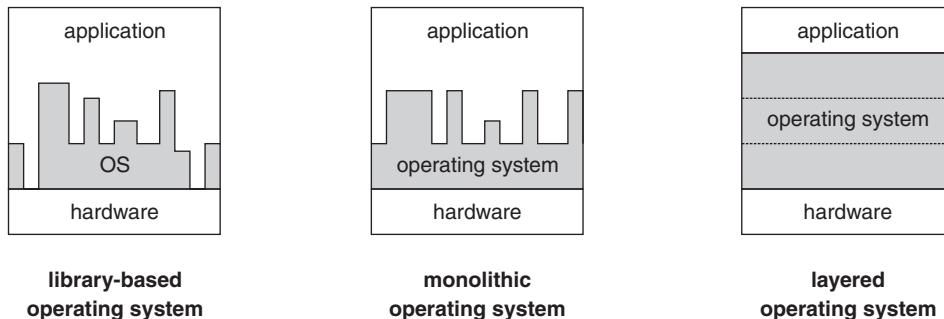
In the course of time, the term ‘card operating system’ (COS) has become accepted throughout the world as a designation for a smart card operating system. It often forms part of the product name, such as STARCOS, MPCOS or TCOS. Presently, there are more than a dozen companies worldwide that produce general-purpose and application-specific smart card operating systems. They are summarized in Table 13.1.

It is conceivable that a consolidation of smart card operating systems, with their various features and functions, could occur in the medium term. This would have the same effect as with PCs, most of which now work with a ‘uniform’ operating system. Whether this will actually happen with smart card operating systems remains to be seen, since here the circumstances are less favorable for the development of a uniform solution. Extremely severe requirements regarding security and software quality, limited memory capacity, and the demand for confidentiality of the operating system software, taken together, certainly have the potential to make it impossible to produce a general-purpose smart card operating system in the foreseeable future that can satisfy everyone’s wishes.

In this chapter, we explain many of the features and varieties of modern smart card operating systems on the basis of various specifications, standards and descriptions of software for smart cards. However, there is one thing we must explicitly mention first: the spectrum of modern smart card operating systems has become very large due to the extreme diversity of the related requirements. Some operating systems for access control systems are still written in assembly language, occupy only 10 KB of ROM, and run on 8-bit processors. At the other extreme, there are operating systems in the telecommunication sector that run on 32-bit processors, encompass several hundred kilobytes of program code, and provide complex functions such as multitasking and HTTP servers. Consequently, the subject of operating systems can only be dealt with in a very differentiated form, and it should always be seen in connection with the associated application area.

## 13.1 EVOLUTION OF SMART CARD OPERATING SYSTEMS

The evolution of operating systems for smart cards has passed through the same stages as for all other computer systems, as depicted in Figure 13.1. The initial special-purpose



**Figure 13.1** Schematic representation of the historical evolution of smart card operating systems. Until around 1991, new operating systems were library-based. They were gradually replaced by monolithic operating systems by around 1998. Monolithic operating systems have now been replaced by layered operating systems. For simplicity, this example shows only one application in each case

programs for individual applications were repeatedly generalized and extended, ultimately leading to the development of general-purpose operating systems designed for ease of use in structured environments.

From a modern perspective, the programs available for smart card microcontrollers in the early 1980s, at the start of this evolution, cannot be considered true operating systems. They were nothing more than application software embedded in the ROM of a chip. However, manufacturing mask-programmed microcontrollers is expensive and time-consuming, so the need for general-purpose kernel routines that could form the basis for special-purpose application software in EEPROM arose very quickly. This also increased the demand for memory, which led some companies to move in the opposite direction, back to special-purpose software for individual applications.

As the market demand for individually tailored solutions has constantly increased up to now, producers of operating systems have more or less been compelled to design their software to meet this demand. Presently, the customized approach with specially developed ROM software is only used in applications involving a very large number of cards. General-purpose operating systems based on standardized commands are the norm, since in principle they can be used for any type of application. If this is not possible for some reason, they are at least designed such that they can be adapted to the requirements of any particular application with a minimum of time and effort.

The evolution of smart card operating systems from 1980 to the present can be illustrated nicely by the smart cards used in German mobile telecommunication systems. The smart cards used in the C-Netz (the German precursor of the GSM system) starting in 1987 had an operating system that was optimized for this application. It included a specific transmission protocol, special commands, and a file structure tailored to the application. All in all, this sort of card certainly has a complete operating system, but it is fully tailored for use in the C-Netz system. The essential components of the application based on this operating system were matched to the specific requirements of the library-oriented operating system and its underlying hardware.

The next step was the transition from special-purpose solutions to a somewhat more open operating system architecture. One example of this is the first GSM cards, which had a significantly more open and multifunctional design. When GSM smart cards were specified, there were already draft standards for the command set and data structures of smart cards,

which meant that the cornerstone had been laid for compatibility among various operating systems. This formed the basis for further incremental development. Applications using this monolithic operating system are largely independent of the hardware and are based on the various interfaces of the operating system.

Modern operating systems for GSM or UMTS now have functions such as memory management, multiple file structures and state machines that correspond to the capabilities of multiapplication operating systems. They can manage several applications independently while preventing interactions between the applications. Most of them also have very elaborate state machines and a large command set, and in some cases they support more than one data transmission protocol.

Even this does not represent the end of the evolution. Smart cards for mobile telephones are taking over more and more of the functions of the telephone itself, such as driving the display and polling the keypad. In order to provide the maximum possible flexibility, it is necessary to break with what has up to now been a rigid fundamental principle in the smart card world. A modern smart card operating system must be able to run third-party program code in the card. With smart card operating systems, it goes without saying that this must not have any detrimental effect on the functionality or security of other applications in the card. All modern smart card operating systems have a layered structure, with only the bottom layers being dependent on the hardware. The hardware is increasingly abstracted in the layers built on top of these lower layers.

In the foreseeable future, it is certainly possible for the evolution of smart card operating systems to lead via several intermediate steps to an international quasi-standard for general-purpose operating systems, as has already occurred with many other types of operating systems. It may take a while, but sooner or later a certain industry standard comes to predominate, and competitors in the market must support it as the ‘lowest common denominator’ if they wish to continue to operate successfully. Such a standard does not yet exist in the smart card world, but the first signs of one are foreseeable. The basis for this, in contrast to the PC world for example, is formed by international standards and specifications. These are primarily the ISO/IEC 7816 family of standards, the UICC specifications (including TS 102 221), and the EMV specifications.

Various characteristics of smart card operating systems for telecommunication applications are listed in Tables 13.2 and 13.3.

## 13.2 FUNDAMENTAL ASPECTS AND TASKS

Smart card operating systems, unlike familiar PC operating systems, do not have user interfaces or the ability to access external storage media. This is because they are optimized for entirely different functionality. Security during program execution and protected access to data have the highest priority. Due to the limitations imposed by the amount of available memory, smart card operating systems have a small program code size in the range of 10 to 400 KB. The lower end of this range applies to specific applications, while the upper end applies to multiapplication operating systems, some of which support multitasking and interpreters for downloadable program code such as Java Card. However, the average memory requirement is around 100 KB for operating systems that do not support downloadable third-party software.

**Table 13.2** Source code size and characteristic program structures of typical smart card operating systems for the GSM telecommunication application

Type of smart card operating system	OS for GSM circa 1990, implemented in assembly language	OS for GSM circa 1997, implemented in assembly language	OS for GSM circa 1996, implemented in C
Functionality	SIM, administrative commands	SIM, OTA, administrative commands	SIM, administrative commands
Microcontroller	6 KB ROM 3 KB EEPROM 128 byte RAM	16 KB ROM 8 KB EEPROM 256 byte RAM	16 KB ROM 8 KB EEPROM 256 byte RAM
Total size of linked object code (ROM + NVM)	6 KB + 0.3 KB	16 KB + 0.8 KB	16 KB + 0.8 KB
Lines of source code (including comments)	≈ 10 100	≈ 22 000	≈ 12 000
Source code files	22	14	37
Subroutine calls and function calls	≈ 470	≈ 930	≈ 115
Number of returns	≈ 95	≈ 35	≈ 205
Number of constants	≈ 360	≈ 250	≈ 100
Branches and IF instructions	≈ 200	≈ 560	≈ 1100

**Table 13.3** Source code size and characteristic program structures of typical smart card operating systems for telecommunication applications (GSM and UMTS). For comparison, Windows Vista has 50 million lines of source code

Type of smart card operating system	OS for GSM circa 2002, implemented in C	OS for GSM circa 2006, implemented in C	OS for GSM circa 2008, implemented in C
Functionality	SIM, SAT, WIM, 3 microbrowsers, OTA	UICC, USIM, JC 2.2, GP 2.1, WIM, 2 microbrowsers, OTA	UICC, USIM, JC 2.2, GP 2.1, WIM, OTA, CAT_TP, SWP/HCI, Quota, 2 microbrowsers, multitasking
Microcontroller	196 KB ROM 68 KB EEPROM 4 KB RAM	368 KB ROM 72 KB EEPROM 8 KB RAM	896 KB flash 52 KB RAM
Total size of linked object code (ROM + NVM)	190 KB + 4.9 KB	368 KB + 5 KB	≈ 500 KB
Lines of source code (including comments)	≈ 120 000	≈ 365 000	≈ 635 000
Source code files	184	704	2 182
Subroutines calls and function calls	≈ 900	—	—
Number of returns	≈ 1 500	—	—
Number of constants	≈ 150	—	—
Branches and IF instructions	≈ 3 100	—	—

The program modules are often written as ROM code. This restricts the programming techniques that can be used, since many techniques typically used with RAM code (such as self-modifying code) are not possible with ROM code. The use of ROM code is also the reason why no changes at all can be made after the microcontroller ROM has been programmed and manufactured. Correcting an error is thus extremely expensive and takes ten to twelve weeks. If the smart cards have already been issued, errors can only be corrected by a large-scale recall campaign, which could destroy the reputation of a smart card system. ‘Quick and dirty’ programming is clearly out of the question. Consequently, the amount of time spent on testing and quality assurance is usually significantly greater than the amount of time spent on programming.

These operating systems must not only have a very small number of defects, they must also be very reliable and robust. They must not allow their functionality, and above all their security, to be impaired by any command coming from the outside world. System crashes or uncontrolled responses to erroneous commands or page faults in EEPROM or flash memory must never occur under any conditions.

From the perspective of operating system design, it is an unfortunate fact that the implementation of certain mechanisms is influenced by the hardware that is used. In particular, the secure state of the nonvolatile memory has a small but nevertheless noticeable effect on the design of the operating system. For example, all retry counters must be designed such that their maximum value corresponds to the erased state of the EEPROM or flash memory. Otherwise it would be possible to reset the retry counter to its initial value by means such as intentionally de-energizing the card while it is updating the counter, since an EEPROM or flash memory must be erased before certain types of write operations are performed. If power can be removed exactly at the moment between erasing the memory and writing the new value, the portion of the EEPROM or flash memory used for the retry counter will be in the erased state. With improper operating system design, this would amount to resetting the retry counter to its initial value.

This type of attack can be countered by properly coding the counter states as previously described or by using an atomic operation to update the retry counter. Similar considerations apply with regard to the retry counter and the secure (lowest-energy) state of the EEPROM or flash memory. The retry counter must be coded such that its maximum value corresponds to the secure state of the EEPROM or flash memory. Otherwise it would be possible to reset the retry counter, such as by selectively heating certain memory cells. These are only two examples of hardware dependences in the design of a smart card operating system; many others could be mentioned. For security reasons, a smart card operating system must be closely coupled to the hardware of the microcontroller used. Consequently, it can never be fully hardware-independent.

There is also another aspect to the concept of a secure operating system. Trap doors and other forms of hidden access for system programmers are often incorporated in large operating systems, where they are perfectly normal features. However, they must be totally excluded from smart card operating systems. For example, there must not be any possibility that someone could use some mechanism to bypass the operating system and obtain unauthorized read access to data.

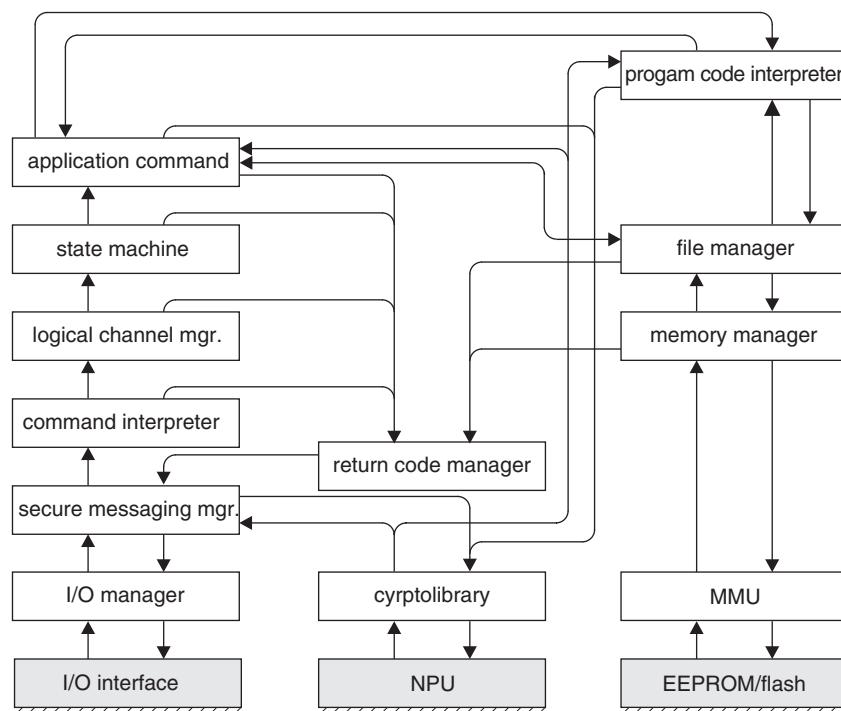
Another aspect that should not be underestimated is the required processing capacity. The cryptographic functions of the operating system must execute very quickly. It is thus common to spend weeks of painstaking effort in the design process to optimize the algorithms concerned in assembly language.

In summary, the main tasks of a smart card operating system are:

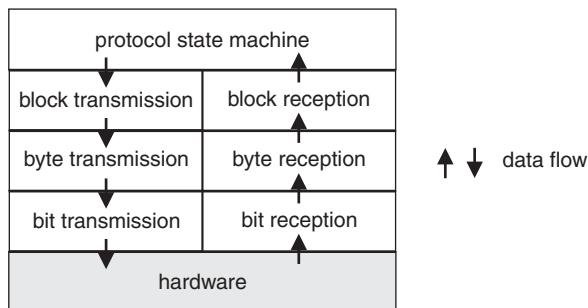
- transferring data to and from the smart card
- controlling command execution
- file management
- managing and executing cryptographic algorithms
- managing and executing program code

### 13.3 COMMAND PROCESSING

Command processing in smart cards that do not support downloadable program code is typically organized as shown in Figure 13.2. The smart card receives each command via the serial I/O interface. The I/O manager performs error detection and correction processes as necessary, fully independent of the other, higher-level layers. After a command has been received completely without errors, the secure messaging manager must decrypt the message or test its integrity. If secure data transmission is not used, this manager is completely transparent to both the command and the response. Command processing in the I/O manager is structured in hierarchical layers as shown in Figure 13.3.



**Figure 13.2** Command processing in a smart card operating system. The level of hardware abstraction increases from the bottom to the top of this figure. The program code interpreter is optional



**Figure 13.3** Command processing sequence in the I/O manager of a smart card operating system

After this processing, the next higher level (the command interpreter) attempts to decode the command. If this is not possible, the return code manager is called. It generates a suitable return code and sends it back to the terminal via the I/O manager. It may be necessary to design the return code manager in an application-specific fashion, since the return codes are not necessarily the same for all applications. If the command can be decoded, the logical channel manager determines which channel should be selected, switches to the state of this channel, and invokes the state machine if no error occurs.

The state machine checks whether the command, in combination with its accompanying parameters, is allowed in the current state of the smart card. If it is, the program code that actually performs the function of the application command is executed. If the command is prohibited in the current state or its parameter values are not allowed, the terminal receives a message to this effect via the return code manager and I/O manager.

If it is necessary to access a file while processing the command, this occurs only via the file manager, which converts all logical addresses into physical addresses within the chip. The file manager also monitors all addresses with regard to region boundaries and checks the access conditions for the file in question.

The file manager utilizes a lower-level memory manager, which performs all management functions for the physically addressed nonvolatile memory. This ensures that this program module is the only one that uses physical addresses, which significantly increases the portability and security of the entire operating system.

A central return code manager is responsible for generating return codes. It always produces the complete response sent to the calling routine. This layer is responsible for managing and generating the return codes used in all other parts of the operating system.

Since most smart card operating systems utilize cryptographic functions, in most cases there is also a dedicated library of cryptographic functions separate from the rest of the operating system. This cryptographic library serves all other modules as a central resource for cryptographic functions.

In addition to these layers, an interpreter or a verification module for executable files may be present in the region above the application command layer. It monitors the programs contained in these files and executes or interprets them. The exact design and implementation depends on whether there are actually any files with executable code present, and whether the stored code is machine code for the processor or code to be interpreted. This topic is discussed in detail in Section 13.14 on page 491.

## 13.4 DESIGN AND IMPLEMENTATION PRINCIPLES

As is well known, design defects first manifest themselves in the implementation stage, where they result in costs that are several times greater than with a better design having fewer defects. However, defects are a fact of life in all software projects. In order to minimize such defects, it is advisable to observe several principles in the design and implementation of a smart card operating system.

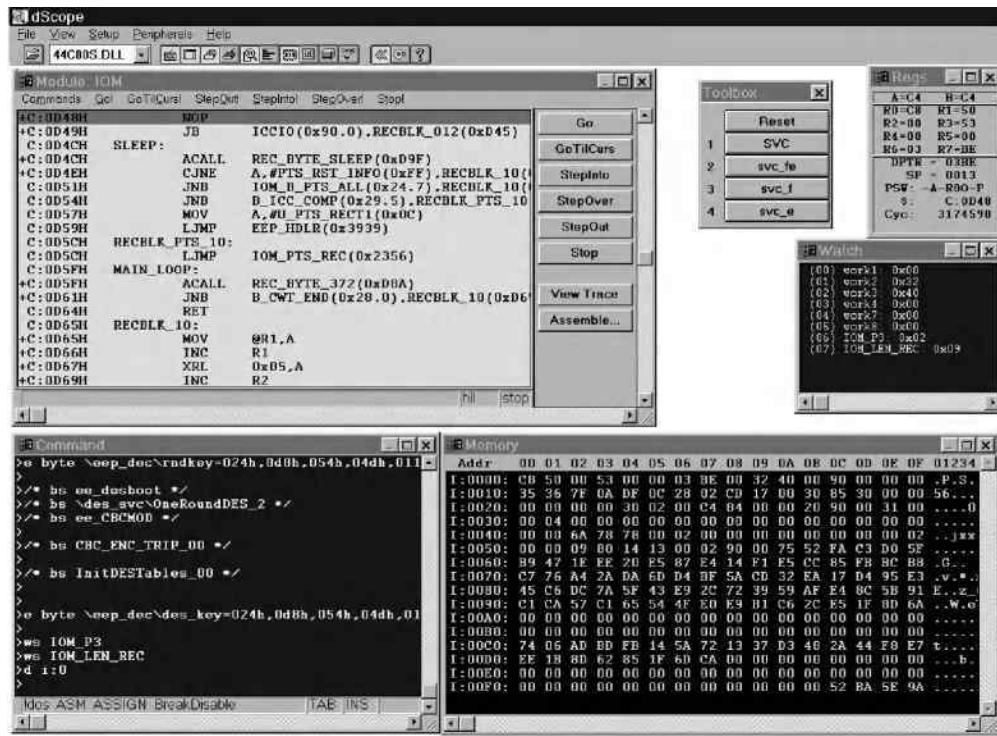
Due to its task profile, a smart card operating system is a secure operating system that must manage information and above all hold information confidential. In addition, it is normally not possible to make any changes at all to the software once it is in use, regardless of whether the operating system is located in ROM or flash memory. The first principle follows directly from these considerations: a smart card operating system must be extremely reliable, which means it must be virtually free of defects. Total absence of defects can never be achieved in practice, since even the smallest smart card operating system kernels are too large to allow all of the options of their internal processes to be fully tested.

However, strict modular design makes a decisive contribution to the discovery and elimination of any defects that may be present at the implementation stage. This modularity, which strongly enhances the reliability of the operating system, need not necessarily inflate the code size. An additional benefit of a modular design is that any system crashes that may occur generally do not affect the system's security as strongly as with highly optimized program code occupying less memory. With a modular design, the consequences of defects remain localized, and the operating system as a whole is more robust and more stable.

The fact that the software must often be implemented in assembly language or C, which is close to the hardware level, increases the risk of defects. A design based on individual, fully testable modules strongly contributes to detecting programming errors in a timely manner and limiting the scope of their effects, due to use of defined interfaces. The layered operating system architecture shown in Figure 13.2 on page 447 is a logical result of this modular approach. The increased amount of design and programming effort that this requires is offset, at least financially, by the fact that tests and reviews are significantly easier. As a result, almost all operating systems now have an architecture that is the same as or very similar to the one described here.

The approach usually taken to designing an operating system is the module/interface concept. In the design process, the tasks of the operating system and the application are broken down into functions as far as possible, and these functions are then packaged in modules. After the module interfaces have been defined precisely, programming of the individual modules (possibly by several persons) commences. In the ideal case, the first version is a platform-independent implementation, which means it does reflect the properties of a specific microcontroller. After this version has been fully reviewed and tested, the necessary adaptations to the appropriate microcontroller can be made.

As the amount of program code for a smart card operating system is still relatively small (see Table 13.4 for some typical examples with assembly language implementation), this very pragmatic approach can be used without any major problems. Its advantages, which are low planning effort and the ability to distribute the programming tasks over several people, along with easy reusability of the program code, yield their maximum benefits here. The accompanying drawbacks that must be accepted with this approach are the difficulty of proving the correctness of the system and the fact that changes to the system may strongly impact a large number of modules.



**Figure 13.4** An example of the simulation of a smart card microcontroller in a typical development environment for the C and assembly programming languages. The source code pane is at the top left, with various processor registers displayed at the top right. A memory map of the RAM is shown at the bottom right, while the command lines for controlling the simulator are shown at the bottom left. This simulator allows the software developer to monitor all the functions of the microcontroller and intervene at every stage of program execution (Reproduced with permission from Kiel)

**Table 13.4** Examples of the typical memory use of smart card operating system functions implemented in assembly language

Function	Required program memory (bytes)
CRC algorithm	≈ 50
File management (MF, two DF levels, EF, four EF structures)	≈ 1 200
DES algorithm (not SPA/DPA-resistant)	≈ 1 200
DES algorithm (SPA/DPA-resistant)	≈ 2 000
EEPROM write	≈ 150
RSA algorithm (with NPU)	≈ 300
T = 0 data transmission protocol	≈ 500
T = 1 data transmission protocol	≈ 1 200

Software development for smart card operating systems is moving away from programming entirely in assembly language. Almost all new projects are implemented in C, a high-level language that is nevertheless close to the hardware level, right from the start. However, the kernel of the operating system is still based on machine-dependent assembly language routines, while all higher-level modules, such as the file manager, the state machine and the command interpreter, are programmed in C. This significantly reduces the implementation time and improves the portability and reusability of the software, and the primary benefit of using a high-level language is improved testability of the program code. The improved, more easily understood program structure provided by a high-level language also yields a distinctly lower defect rate.

Unfortunately, the program code generated by a C compiler, even if it is highly optimized, occupies 10 to 30 % more space in ROM than equivalent assembly language code with the same functionality. In addition, the speed of a C implementation is slightly lower than an assembly language implementation. This is a sensitive issue for cryptographic algorithms and data transmission protocols, but the other routines of the smart card operating system normally do not include extremely time-critical processes. Smart cards with contactless interfaces form an exception here because they must have very short transaction times, for the simple reason that the users of these cards expect that they only have to be passed through the card reader field once for each transaction.

The greatest challenge in C programming is not necessarily the extra memory space needed in ROM or lower execution speed, but instead RAM usage. This type of memory is extremely limited in smart cards, and it has the further disadvantage of occupying the most space per bit on the chip compared with other types of memory. For this reason, careful attention must be given to using as little RAM as possible.

Smart cards are used in application areas in which security is a very important factor, so the card issuer and/or application provider must have considerable trust in the integrity of the producer of the operating system, who has every opportunity to take unfair advantage of the entire system by means of deliberately introduced security gaps. For example, consider an electronic purse in a smart card whose load command has been manipulated such that under certain conditions the purse can be reloaded without authorization.

Such scenarios are the reason why only a few operating system producers have become established up to now. The risk of purchasing a supposedly secure operating system that actually contains a Trojan horse is significantly greater if it is the product of a small, unknown producer than if it is a product of a well-known producer.

In order to achieve even better insight and security in this regard, for some time more effort has been put into evaluating smart card operating systems in accordance with the Common Criteria (CC).<sup>1</sup> This occurs either on the initiative of the operating system producer or in response to the demands of major card issuers, whose objective with such an evaluation is to achieve an increased degree of assurance that there are no significant errors in the program code. Checking for deliberately introduced Trojan horses during an evaluation would probably be of limited use, as there is practically no limit to the number of ways a Trojan horse can be incorporated in a program.

Up to now, the usual evaluation level for smart card operating systems has been EAL4+ as defined by Common Criteria.<sup>2</sup> Higher evaluation levels are sometimes requested or offered

<sup>1</sup> See also Section 15.5, ‘Evaluation of Hardware and Software’, on page 659

<sup>2</sup> See also Section 15.5, ‘Evaluation of Hardware and Software’, on page 659

in individual cases. It must be borne in mind that a CC evaluation of a complete smart card operating system at the EAL4+ level can cost around €500 000 or more. On top of this comes the obligation for re-evaluation after any modification of the program code, although a re-evaluation is considerably less costly than the initial evaluation because only the changes have to be examined. This is essentially why only relatively few smart card operating systems can boast CC evaluations.

In many cases, evaluations are performed by evaluation bodies without direct reference to the Common Criteria. Such evaluations are limited to thorough review of the design criteria, the source code and the documentation. For example, this is a fundamental requirement for operating systems used in German Eurocheque cards.<sup>3</sup>

## 13.5 OPERATING SYSTEM COMPLETION

In the case of ROM-based products, the life cycle of a smart card operating system consists of two major phases – before completion and after completion. In the phase before card completion, when the microcontroller comes from the semiconductor production facility with an empty EEPROM, all the routines run in ROM. No data is read from EEPROM, and no code is executed from EEPROM. If an error in the ROM code that makes completion impossible is discovered at this point, the entire batch of microcontrollers must be destroyed because the chips are unusable.

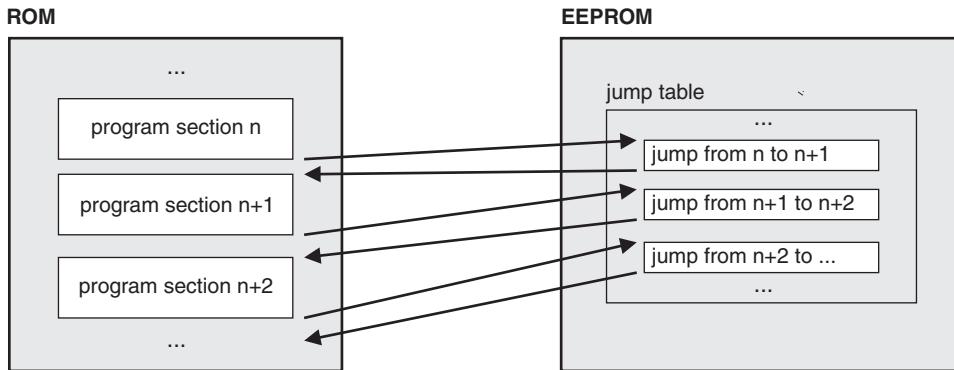
To minimize the likelihood of such a situation, it would be possible to have the ROM contain only a small routine for loading code into the EEPROM, and then load the actual operating system into the EEPROM. However, the chip area per bit is much larger with EEPROM than with ROM, which means that such an approach would excessively increase the cost of the chip. For purely economic reasons, therefore, as much of the code as possible must be located in ROM. Consequently, all the kernel operating system routines, as well as substantial parts of the rest of the operating system, are stored entirely in ROM. Only a few jumps to the EEPROM are provided for use in the completed version.

Some operating systems run completely in ROM even after completion, with only data being stored in the EEPROM, in order to keep the size of the relatively expensive EEPROM as small as possible. Of course, minimizing the area used by the memory comes at the price of major limitations on the flexibility of the operating system.

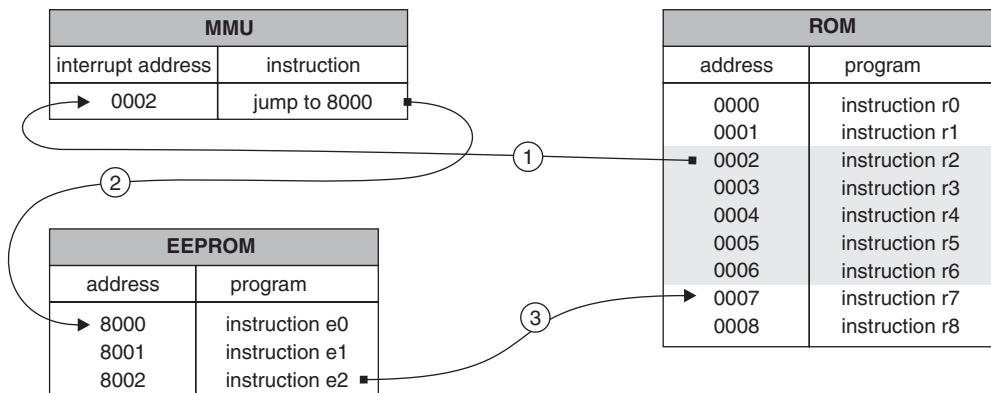
In the completion process, the code in the ROM is adapted to the actual application. The ROM code can be regarded as a large library that is linked and expanded by the code in the EEPROM to form a functional application, as illustrated in Figure 13.5 on the next page. In addition, almost all operating systems allow program code for additional commands or special cryptographic algorithms to be loaded into EEPROM during completion. This has nothing to do with any executable files that may be present, since the contents of such files can be downloaded at a later time, such as when the card is personalized. The routines that are loaded into EEPROM during completion are fully integrated into the operating system. This process can also be used to integrate patches for functional extensions or defect correction.

Figure 13.6 on the facing page shows an alternative to using a table of jumps to nonvolatile memory. However, this alternative requires a memory management unit (MMU), which generates an exception immediately before the start of the code to be skipped and sets the program

<sup>3</sup> See also Section 15.5.2, ‘ZKA criteria’, on page 663



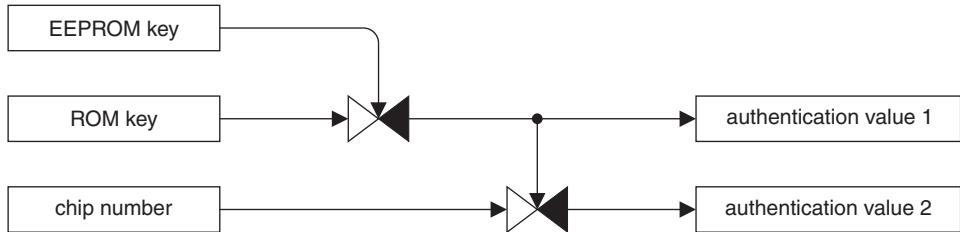
**Figure 13.5** Routines in ROM can be interconnected by a link table stored in the EEPROM when the operating system is completed



**Figure 13.6** Using a memory management unit (MMU) to patch out a section of code in ROM (shaded grey) with a routine stored in EEPROM. In step 1, the program counter reaches the address monitored by the MMU, which triggers an interrupt, and in step 2 the interrupt handler branches to a defined address in EEPROM where the new program code is stored. After this program code has been executed, a jump back to the appropriate location in ROM occurs in step 3

counter to the address of the new code in nonvolatile memory. When this portion of the program has run to completion, program execution is continued from the appropriate location in ROM by means of an absolute jump.

In order to complete a smart card operating system in EEPROM, unilateral or mutual authentication between the smart card and the outside world is necessary. There are many different methods that can be used for this. Figure 13.7 on the following page shows a typical example of a simple but relatively flexible approach. The producer of the smart card operating system incorporates a secret value in the ROM code of the microcontroller, which is thus specific to this ROM version of the operating system. During chip fabrication, the semiconductor manufacturer writes a secret key to the EEPROM, which may have a different value for each production batch. After this, if the ROM and EEPROM keys are known, a



**Figure 13.7** A possible scheme for distributing and deriving a secret key for authentication prior to completion of the smart card operating system. Authentication value 1 is specific to a batch of smart card microcontrollers, while authentication value 2 is specific to an individual microcontroller. Triple DES or AES can be used as the cryptographic algorithm

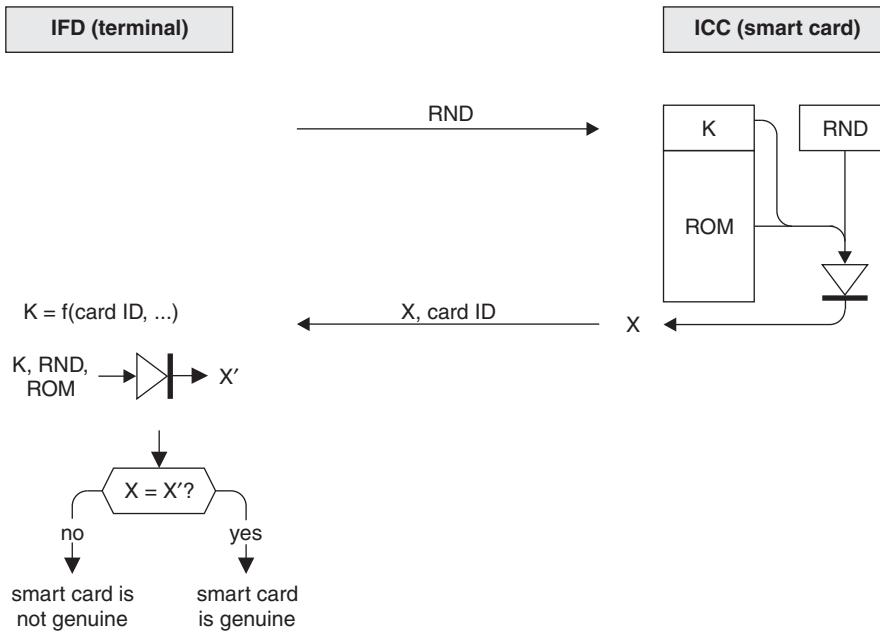
known cryptographic algorithm can be used to compute an authentication value for smart card completion. The operating system can be completed only after successful authentication.

This method allows batch-specific authentication to be performed. If an operating system producer provides only one authentication value to a card manufacturer, it can only be used to complete one particular batch of smart card microcontrollers. The advantage of this method is that the producer of the operating system only has to have one ROM mask, instead of generating a different version of the ROM for each card manufacturer.

This scheme can be further refined. For instance, a chip-specific authentication value can be generated using the microcontroller serial number. Not only does this increase the level of security if the chips fall into the wrong hands on their way to being completed, it also allows the producer of the operating system to control the completion process at the individual chip level. With such an approach, the card completer receives a list of authentication values and can only complete the operating system in the corresponding microcontrollers. This method, or a similar method, is commonly used in practice.

Conceivably, a special smart card containing a function for reading out the contents of the memory could be smuggled into the smart card manufacturing process and be completed, initialized and personalized, after which the secret data loaded into memory during these processes could be read out. Although this would be rather difficult in a typical smart card manufacturing process due to the strong security measures that are used, it is certainly not possible to fully exclude such a scenario. In order to provide inherent defense against this conceivable form of attack, the authenticity of the microcontroller can be verified prior to completion. This can be done in a large variety of ways, one of which is shown in Figure 13.8 on the next page.

This method requires the semiconductor manufacturer to store an individual key in each microcontroller. Prior to completion, a random number is sent to the smart card in order to prevent replay attacks. This random number, the individual card key and the ROM contents are then converted into a hash value by a hash function in the smart card, and this hash value is conveyed to the outside world along with a card identifier. There the card-specific key can be computed using the card identifier. Since the ROM contents and the random number are known to the outside world, the hash computation can be reproduced in the outside world, and the result can be compared with the hash value received from the smart card. If the two values match, the smart card is authentic. Although this method is simple, it is without doubt sufficiently effective. Certain aspects can be further refined, but our primary objective here is to present a conceivable approach to verifying authenticity prior to completion.



**Figure 13.8** A possible procedure for verifying the authenticity of the ROM portion of a smart card operation system. This sort of verification is especially worthwhile prior to completing the operating system in order to reliably prevent loading of genuine completion data into counterfeit cards

### 13.5.1 Operating system boot loader

With smart card microcontrollers that have only flash memory, the operating system must be loaded into memory after the microcontrollers have been produced. A boot loader is often used for this purpose. This small program can be used to load the operating system into the proper region of flash memory after suitable authentication between the smart card and the terminal. The boot loader is either located in a small ROM, which also contains the hardware tests for the chip, or it is loaded into the flash memory as the last step of the semiconductor manufacturing process on completion of testing. Its function is essentially the same as the routine that is used to load portions of the operating system into EEPROM with ROM-based microcontrollers.<sup>4</sup>

### 13.5.2 Hardware recognition

Most relatively recent operating systems can run on microcontrollers with various amounts of nonvolatile memory, although the sizes of the ROM (if present) and RAM must remain unchanged. This allows the card issuer to always use the least expensive version of the microcontroller that meets his needs. For example, a card issuer could start with an inexpensive

<sup>4</sup> See also Section 13.5, ‘Operating System Completion’, on page 452

single-application card with 1 KB of EEPROM, and then if necessary migrate to more expensive microcontrollers with 4, 16, 32 or 64 KB of EEPROM for its multiapplication cards. To the extent that the microcontroller manufacturer offers such a range of chips, the smart card operating system must have matching capability. This means that it must be able to automatically recognize the size of the nonvolatile memory and then configure its internal pointer structures for maximum free memory, file sizes and similar general parameters.

The technical implementation of this involves using an operating system routine to read the semiconductor manufacturer's fabrication data and calculate the size of the available EEPROM or flash memory on the chip from this data. Alternatively, the semiconductor manufacturer can store the memory size as data in the nonvolatile memory during fabrication. This technique can only handle variable EEPROM or flash memory sizes. Current smart card operating systems cannot adapt themselves to various sizes of RAM under software control.

The main advantage of this hardware recognition capability is that the producer of the smart card operating system does not have to match the program code to the variable size of the nonvolatile memory. This eliminates a possible source of defects, and above all it means that the operating system does not have to be re-evaluated for every new hardware platform. The hardware recognition capability of modern operating systems saves considerable time in software development and can thus reduce product development time by several weeks.

### 13.5.3 Soft and hard masks

The terms 'soft mask' and 'hard mask' are often used in connection with field trials and smart card operating systems. Strictly speaking and from a purely logical perspective, both terms are nonsensical because a ROM mask – which means the program code located in the ROM – is always unalterable and thus 'hard'. In the jargon of the smart card world, however, the term 'soft mask' means something similar to a ROM mask. This term is used when part or all of the program code for a smart card operating system, or the commands for an application, are located in EEPROM, which means that the code can be altered relatively easily, without the cost and time involved in generating a new ROM mask. This sort of mask is thus alterable, or 'soft'. Soft masks are primarily used in testing and field trials, since they allow defects to be corrected and programs to be modified quickly and inexpensively. The disadvantage of using a soft mask is that it requires using chips with large EEPROMs, which are more expensive than equivalent chips with program code in ROM. However, since field trials normally do not involve issuing millions of cards, the increased cost of using chips with large EEPROMs is entirely acceptable.

Once the test or field trial using a soft mask has been completed and the program code in the EEPROM is operational without any further modifications, it can be moved from the EEPROM to the ROM by generating a true ROM mask. This can be done with relatively little effort, and the result is called a 'hard mask' because it cannot be altered. Strictly speaking, the only advantage of a hard mask is that a given amount of memory occupies significantly less chip area in ROM than in EEPROM, thus allowing a smaller and less expensive microcontroller to be used for the same amount of program code.

This two-stage process using soft and hard masks for new smart card applications also provides flexibility and allows substantial changes to the program code to be made, even shortly before the cards are issued to users. With a traditional process using a pure ROM mask, it is not possible to make significant changes to the program code once the mask has been given

to the manufacturer. Since the two-stage process is superior to the traditional process in this regard, nowadays a soft mask is almost always used initially when introducing a new smart card application. Migration to a hard mask occurs only after any necessary modifications have been made to the program code.

However, the growing use of microcontrollers with flash memory instead of ROM and EEPROM is causing this method to be used less and less often, since flash-based operating systems essentially have the characteristics of soft masks by virtue of their design.

### 13.5.4 Operating system APIs

Originally, smart card operating systems did not allow third-party software to be loaded into the cards and run as needed. Consequently, at that time operating systems did not have published programming interfaces that could be used by third parties to call operating system functions. More recent developments in smart card operating systems, such as MULTOS and operating systems that support Java (Java Card), allow third parties to load their own program code into smart cards. In order to eliminate the need to again program routines already present in the operating system, such operating systems include carefully conceived application programming interfaces (APIs) that provide access to the most important functions of the operating system. Naturally, practically all operating systems have their own internal APIs, but these APIs are not designed for external use and are usually confidential.

As an exception to the usual situation in the smart card world, there are no general standards regarding APIs for smart card operating systems. Instead, two industry standards have come into predominant use. One of them consists of the various Java Card APIs, while the other is the Multos API. The APIs described in the related specifications provide access to the essential functions of the operating system.<sup>5</sup> These functions include access to the file management system, calls to the available cryptographic functions, and of course data transmitting and reception.

## 13.6 MEMORY ORGANIZATION AND MEMORY MANAGEMENT

The three types of memory used in smart card microcontrollers have entirely different properties. ROM can only be programmed as a single block using a mask during manufacturing, and its contents are static for the life of the chip. Given the structure of ROM, the chance of an undesired change in the ROM contents is practically zero.

In contrast to ROM, RAM retains its contents only as long as power is applied to the smart card. A power failure causes total loss of all data held in RAM. However, data can be written to RAM at the full working speed of the processor, and RAM can be erased an unlimited number of times.

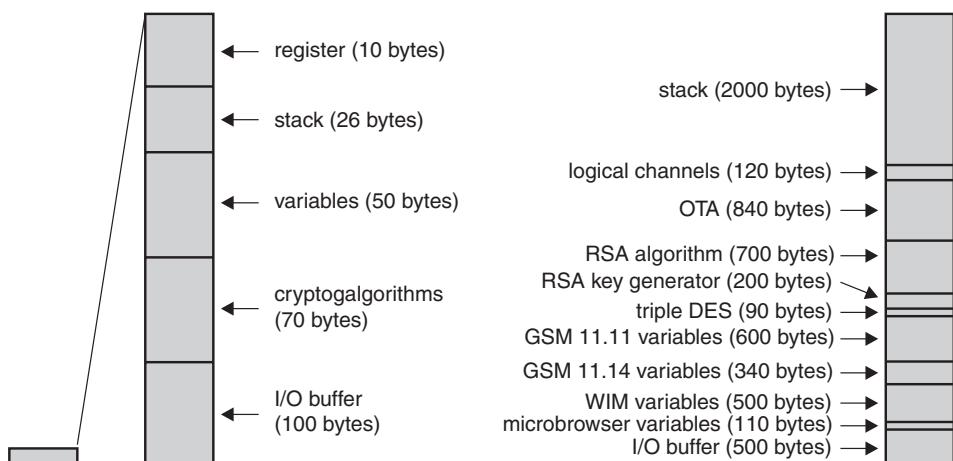
EEPROM, by contrast, can retain data without external power. However, it has three disadvantages, which are its limited lifetime, the fact that it is divided into pages, and its relatively long write and erase times (around 1 ms/byte).

<sup>5</sup> See also Chapter 25.4, ‘Directory of Standards and Specifications’, on page 999 and Section 13.16.3, ‘Multos’, on page 519

With the exception of the interrupt vectors, which are determined by the microcontroller, the operating system program code stored in ROM does not have to conform to any specific structure. The individual routines can thus be linked to each other in any desired sequence, although an attempt is made to limit the length of jumps in order to save memory space. It is important for the ROM to be protected by error detection codes (EDCs), since it is certainly possible for errors to occur occasionally in the ROM. A scratch in the ROM region of the microcontroller, for example, or a chip fracture during the wire bonding process, can cause the data in the ROM to be incorrect. Interestingly enough, this does not necessarily mean that the operating system can no longer function; instead, it is certainly possible that only specific routines will run incorrectly. In order to prevent problems that might arise in such situations, the ROM is checked when the operating system starts up to verify that it is fundamentally free of defects.

### 13.6.1 RAM memory management

Figure 13.9 shows the usual memory structure of a 256-byte RAM. It is divided into regions for the registers, the stack, general variables, workspace for cryptographic algorithms, and the I/O buffer. If a 256-byte I/O buffer is needed, for example, or if additional variables must be stored in RAM, the limits of the available memory can be reached very quickly. This problem is solved by having workspace in the nonvolatile memory, which is thus used like RAM. A disadvantage of this is that write operations with nonvolatile memory take around 10 000 times as long as with RAM. Another disadvantage is the limited lifetime of EEPROM and flash memory cells, which unlike RAM cells cannot be written an unlimited number of times. Nevertheless, relocating RAM contents to EEPROM or flash memory is often the only solution, such as when an I/O buffer is needed that is larger than the total available RAM. For



**Figure 13.9** The diagram on the left shows a typical memory structure of a 256-byte RAM with a simple smart card operating system programmed in assembly language. The diagram on the right shows the memory structure of a 6-KB RAM in a high-performance telecommunication smart card whose operating system is written in the C language

comparison, Figure 13.9 on the preceding page also shows the typical memory structure of a 6-KB RAM in a high-performance smart card for telecommunication applications.

### 13.6.2 EEPROM memory management

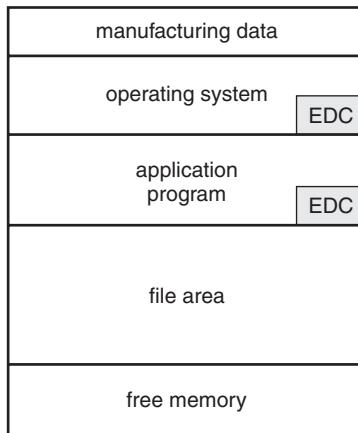
Data storage in nonvolatile memory is much more complicated and intricate than in the other two types of memory. With modern operating systems, the basic structure is roughly as follows. In a region at the start of nonvolatile memory with special hardware protection, which is available in many microcontrollers, special manufacturing data can be stored in the chip, such as a number that is used only once and is thus unique. Many semiconductor manufacturers also use this region to record the chip type and the amount of nonvolatile memory available to the operating system, as well as other types of data as listed in Table 13.5. This region is usually designed for write once, read multiple (WORM) access. This means that it can be written only once, after which it can only be read. Technically, this is usually achieved by using regular EEPROM cells that have been modified so that they cannot be electrically erased.

Above this region, which usually has a size of 16 or 32 bytes, come the operating system tables and pointers, which are loaded into the EEPROM when the card is completed. The combination of these tables and pointers and the routines stored in ROM yields the complete smart card operating system. To ensure that the operating system is always secure and stable in use, this region is protected by an error detection code (EDC) that is recomputed and checked before the first access or even before every access. If a memory error is detected during EDC checking, the affected portion of the EEPROM or flash memory must not be used afterward, as otherwise correct operation of the operating system cannot be assured.

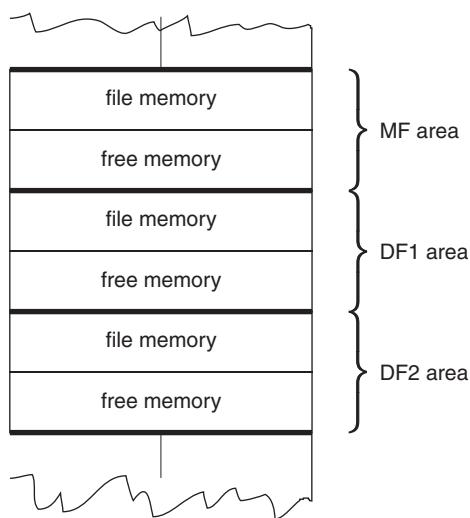
Above the protected operating system region, there is a region that contains additional application program code. If need be, this region may also be protected against changes by a checksum. Application-specific commands or algorithms that should not be located in the ROM or that are too large to fit in the ROM can be loaded in this region.

**Table 13.5** Sample manufacturing data written by some semiconductor manufacturers to a WORM region of the nonvolatile memory during chip fabrication. The first five manufacturing data elements, taken together, yield a unique 8-byte chip number. The major advantage of a chip number generated in this manner is that it does not require precise time synchronization over several production locations, but instead only data available to all production machines in every manufacturing facility

Data element	Size (bytes)
Semiconductor manufacturer	1
Production facility	1
Semiconductor processing batch number	2
X coordinate on the wafer	2
Y coordinate on the wafer	2
Microcontroller type	2
Optional hardware components	6
RAM size	2
EEPROM/flash size	3
ROM size	3



**Figure 13.10** Example EEPROM structure of a smart card operating system



**Figure 13.11** Example file region structure of a smart card operating system that supports several independent, physically separated applications. Although this arrangement yields extremely rigid separation of the individual applications, it is no longer used in operating systems with dynamic file management because it is too inflexible with regard to free memory management

The next region contains all the file structures, or in other words, the entire externally visible file tree. This region is not protected as a whole by a checksum, but instead usually has strong file-based protection. The internal structure of this region is shown in more detail in Figure 13.11.

A free memory region may be located at the top of nonvolatile memory, where it is managed by a dedicated memory manager. However, free memory is often assigned to individual applications in the file region, where it can be used within the applications for newly created

files. Otherwise it belongs to the general file region and is available for new applications that are loaded in their entirety.

When a file is deleted, the released memory space is returned to free memory. With the division into file memory and free memory described here, only the memory space of the most recently created file can be assigned to free memory when it is deleted. This severely restricts the entire memory management process, but no other solution is possible due to the limited program memory capacity of smart cards.

### 13.6.3 Flash memory management

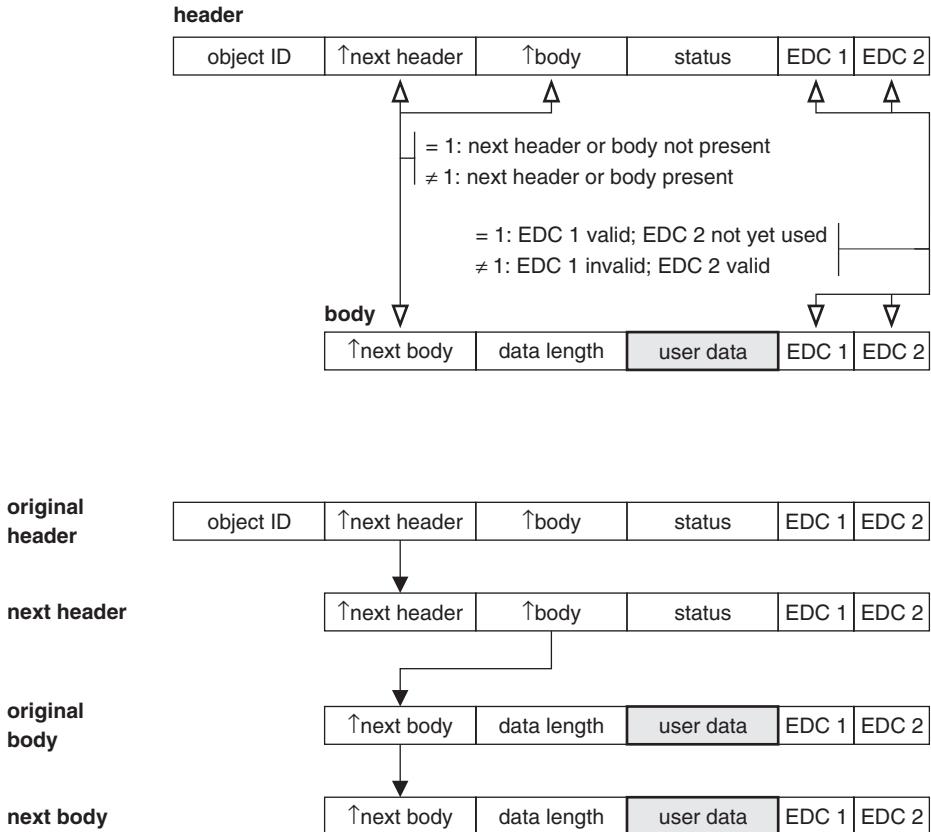
Flash memory is distinguished from EEPROM in that it allows data to be written bytewise relatively quickly, while erasing data involves an entire sector, which can have size of several kilobytes or even several tens of kilobytes with NAND flash. This disparity between the write and erase operations is necessary in order to minimize the chip area occupied by flash memory in smart card microcontrollers. Flash cells can be read or written individually, but they can be erased only relatively slowly in full sectors. By contrast, EEPROM has the same write and erase times and generally allows writing and erasing of individual bytes. As a result of these differences in the properties of EEPROM and flash memory, management functions specifically adapted to flash memory have been developed.

Particularly in the management of NAND flash memory, write accesses must be distributed over the relatively large memory pages as evenly as possible in order to spread the write wear uniformly over all of the memory. This technique is commonly called wear leveling. As this functionality is located between the actual hardware and file management, it is generally called the flash translation layer (FTL). The advantage of this layer is that it avoids the need to alter existing file management functions when they are used with NAND flash instead of EEPROM. Flash translation layers are also available as ready-made software components that only have to be adapted to the characteristics of the NAND flash concerned by means of parameter settings. A variety of commercial and open-source components are available. Typical examples include JFFS/JFFS2 (Journaling Flash File System), YAFFS/YAFFS2 (Yet Another Flash File System), TrueFFS (True Flash File System), and EFFS (Embedded Flash File System).

Memory management specifically adapted to the properties of flash memory is described below in detail. Here it is assumed that erased bits have state 1 and written bits have state 0.

The division into management data and user data shown in Figure 13.12 on the next page is similar to the proven file management structure used for many years with EEPROM. The management data is located in a header, while the associated user data is located in a body section linked to the header by a pointer. The header and the body are both protected against memory errors by EDC (error detection code) checksums. A special feature of the flash memory management scheme described here is the pointers in the header and the body. They are only used in case of changes to the data in the header or the body. In addition to the usual data elements, the header has supplementary status data that can be used to indicate whether the associated body is valid, invalid, erased, blocked, or has other special attributes.

When new user data must be written to the flash memory, the operating system creates a new header. This header contains an object identifier (ID) that can be used to identify the header (and thus the associated user data) uniquely in the entire memory. The second data element in the header is a pointer to a ‘next’ header, which is set to 0 when the header is



**Figure 13.12** Example of a possible flash memory management structure

created. This indicates that the next header does not yet exist. The third element is a pointer to the body, which holds the user data. The status of the data object is stored in the fourth data element. This is followed by the first error detection code (EDC 1), which is calculated over all of the previous data elements. The final data element of the header record is the second error detection code (EDC 2), which is set to 0 when the header is created in order to indicate that it is not yet used. A body linked to the header by the pointer is also created, and its first data element is a pointer to a ‘next’ body. This data element is initially set to 0, which indicates that the next body does not yet exist. The second data element indicates the length of the following user data. It is followed by the user data, which is protected by the first error detection code (EDC 1). The final data element is again a second error detection code (EDC 2), which is not used initially and is thus set 0 to indicate this fact.

When the smart card operating system starts up after a reset, the memory manager searches the appropriate part of the flash memory for all valid object IDs or determines them from a table in nonvolatile memory. This table with pointers to valid object IDs is held in RAM during run time in order to minimize the access times of the managed data objects.

The operating principle of this memory management scheme can be illustrated using two typical application cases. Naturally, all write operations must be protected by suitable atomic

operations to ensure that invalid values cannot occur in the memory management system in the event of a power failure.

The first scenario described here shows the actions that are necessary to alter the status data in the header. Memory management performs the following actions for this purpose. In a free region of nonvolatile memory, it first creates a ‘next’ header. This header is linked to the previous header by a pointer so that it can be found using the known and unchanged object ID. The next header receives the data elements of the original header with the exception of the object ID. The new status data is then written to the next header. After the pointer to the next header has been written to the original header, the second error detection code (EDC 2) is updated accordingly. The file manager can recognize a valid EDC 2 value by the fact that its bits are no longer all set to 1. When EDC 2 is used, EDC 1 is no longer valid and is no longer used.

The second application scenario illustrates modifying user data, which occurs quite frequently. In order to achieve the highest possible transaction speed, it is important to perform as few erase operations as possible. For this reason, when the user data is changed a ‘next’ body is created instead of erasing the original body and updating it with new data. The next body has the same structure as the original body, with the only difference being that it contains the modified user data. The original body is linked to the next body by a pointer, and the second error detection code (EDC 2) is used in place of the first error detection code, which effectively marks the first EDC as invalid.

This example of a data object management scheme for flash memory illustrates one of the mechanisms that is commonly used now to avoid erase operations. This considerably reduces access times. However, it must be borne in mind that repeated creation of new headers and bodies slowly but surely consumes the available free memory. Consequently, it is necessary to use a garbage collector in addition to this data object management scheme to periodically search the entire data object storage space for data that is no longer used. Such garbage collectors typically clean up the memory by consolidating strings of linked headers and bodies into single headers with associated bodies.

## 13.7 FILE MANAGEMENT

All files in a smart card are stored in nonvolatile memory. This is the only type of memory in the smart card that can retain stored data without power and that allows data to be altered if necessary when power is available. It also provides the only means to save information from one session to the next, since the contents of the RAM are lost when the smart card is deactivated, and the contents of the ROM cannot be altered after the chip has been manufactured.

In earlier smart cards, files were directly accessed using physical addresses. Actually, there were no files in the true sense of the word. Instead, the entire memory was linearly addressable from the outside and could be accessed using write and read commands. This is not allowed in modern operating systems for reasons related to security and applications. Object-oriented file management, with access condition data located directly in the files, is currently the standard. The organization and management of these files is the task of the file manager portion of the operating system.

With an object-oriented structure, every file must have a file descriptor that contains all the information relevant to the actual file. In smart card technology, the file descriptor is called the file header. The data content of a file, or in other words the user data, is located in the body of the file.

The data in the file descriptor depends strongly on the capabilities of the file manager. However, the file descriptor must contain at least the following items:

- file name (e.g. FID = ‘0001’)
- file type (e.g. EF)
- file structure (e.g. linear fixed)
- file size (e.g. 3 records with 5 bytes each)
- access conditions (e.g. READ = after PIN code entry)
- attribute (e.g. WORM)
- position in the file tree (e.g. directly below the MF)

With an EF or the MF, the file name is the 2-byte file identifier (FID). With a DF, the application identifier (AID) also forms part of the file name. The file type, which may be MF, DF or EF, must also be indicated.

Depending on the file type, there may be an element in the header that describes the internal structure of the file (transparent, linear fixed, linear variable, cyclic, or TLV). All information regarding the length of the transparent data portion or the number and length of the records is affected by the file structure.

In addition to the basic file properties described above, the operating system needs even more detailed information about the access conditions, which means which commands are allowed to access the file in which states and what types of access are allowed. The access conditions must be specified individually for every possible command. Special file attributes such as high update activity, WORM, or EDC protection can also be indicated if they are supported by the file manager.

All of the above information relates to the file as an isolated object. In order to define the location of the file in the file tree, an additional pointer is needed to specify the exact location of the file in the MF or DF.

### 13.7.1 Pointer-based file management

In simple operating systems, the file headers have fixed lengths that depend on the file type (MF, DF or EF). This reduces the amount of management and computational overhead for internal file management. However, such an arrangement has the disadvantage of being relatively inflexible with regard to extension. Another drawback is that it does not allow an unlimited number of different access conditions to be implemented for any given EF, or even a very large number, since the amount of memory that would have to be reserved to accommodate a large number of access conditions in the header would not be fully used by most applications. Consequently, variable-length headers are very popular in smart card file management systems. The information in such headers can be automatically adapted to the needs of specific applications by the smart card operating system.

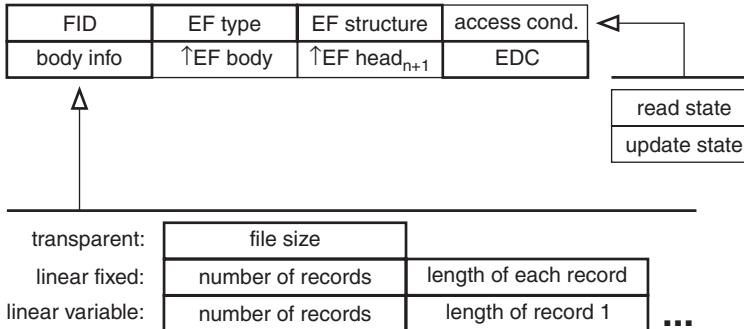
Figures 13.13, 13.14 and 13.15 illustrate a possible implementation of a pointer-based file structure of this sort.

**MF header:**

FID	$\uparrow$ EF head <sub>1</sub>	$\uparrow$ EF key head	$\uparrow$ DF <sub>1</sub>	EDC
-----	---------------------------------	------------------------	----------------------------	-----

**DF header:**

FID	DF name	$\uparrow$ EF head <sub>1</sub>	$\uparrow$ EF key head	$\uparrow$ DF key head <sub>n+1</sub>	EDC
-----	---------	---------------------------------	------------------------	---------------------------------------	-----

**EF header:**

**Figure 13.13** A possible file header structure for MF, DF and EF files (internal and working) in a pointer-based smart card file management system. Heavy borders indicate mandatory TLV-coded data objects, while light borders indicate optional data objects. The numbering is valid only within the directory in which the file is located, rather than globally for all files. This is based on the requirements for a simple file management system specified for Small-OS (see Section 13.17 on page 521). Here ‘access cond.’ stands for ‘access condition’ and ‘head’ for ‘header’

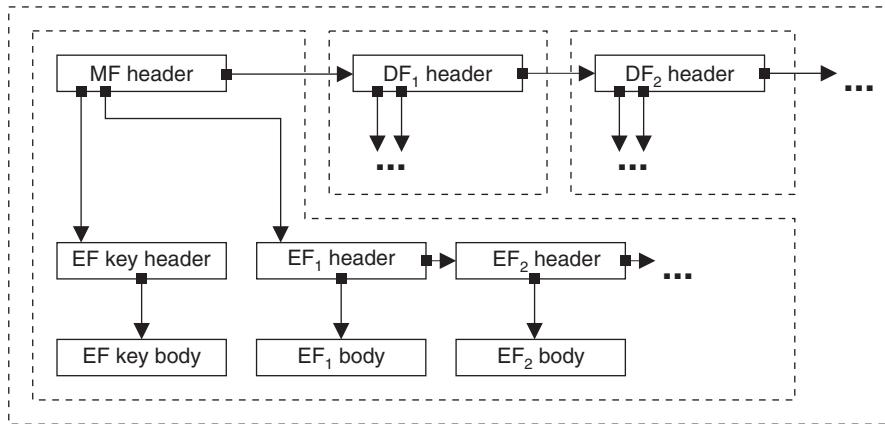
**EF body**

data	EDC
------	-----

**EF key body** (structure of a single record)

key number		
initial state	result state OK	result state NOK
retry counter	maximum retry count	
purpose	PIN / key	
EDC		

**Figure 13.14** A possible file body structure for internal and working EFs in a smart card file management system. All TLV-coded data objects shown here are mandatory. This structure is based on the requirements for a simple file management system specified for Small-OS. Here ‘EDC’ stands for ‘error detection code’



**Figure 13.15** Outline of a possible arrangement of the pointer and data structures in a smart card file management system. This is based on the requirements for a simple file management system specified for Small-OS. The dashed outlines demarcate the memory available to each directory

### 13.7.2 File management with a file allocation table (FAT)

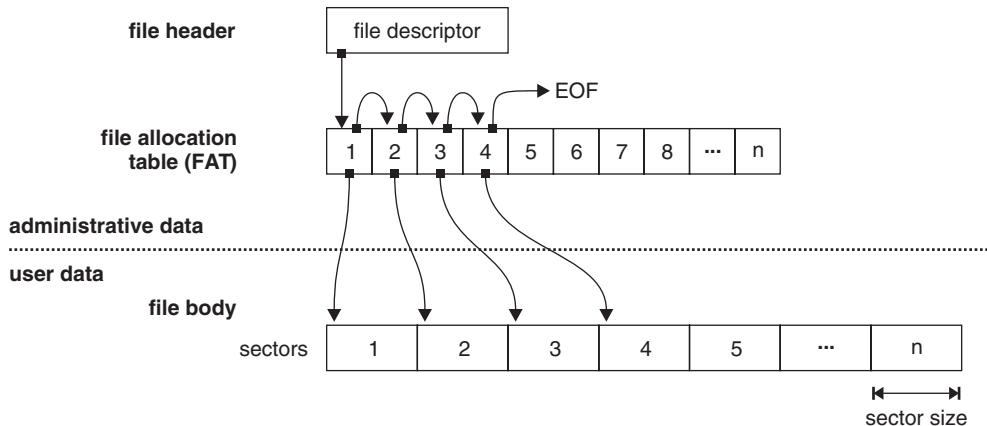
A file management method that is widely used for PC hard disk drives is based on a file allocation table (FAT). This method can be adopted for smart card memory management without any significant changes. With this method, the nonvolatile memory to be managed is divided into many equal-sized pieces called sectors. Ideally, the sizes and start addresses of the sectors correspond to the EEPROM or flash memory pages. This allows write and erase operations to be performed on integral EEPROM pages.

The FAT contains pointers to the individual memory sectors, with the FAT entries also being linked to each other by pointers. Vacant FAT locations and defective sectors are marked by special entries. The amount of memory occupied by the FAT can be significantly reduced if there is a direct relationship between the FAT entries and the sectors. In this case, sector pointers in the FAT are unnecessary.

Figure 13.16 on the facing page shows a possible implementation of a FAT for file management in a smart card. The file descriptor, which contains the essential information about the file structure, includes a pointer to the initial entry in the FAT. In the FAT, a number of sectors corresponding to the size of the file are linked using internal FAT pointers. The final entry always contains an end-of-file (EOF) marker. There is a one-to-one relationship between the FAT entries and the memory sectors that hold the user data belonging to the file descriptor.

The question of whether file management is implemented using file headers and file bodies linked by pointers or using a FAT largely depends on a variety of technical considerations and constraints. Both approaches have advantages as well as disadvantages. FAT-based file management requires memory for the FAT itself, and particularly with small files this is disproportionately larger than the amount of memory needed for a pointer-based system. However, memory fragmentation does not occur in a FAT-based system because it fundamentally cannot occur in such a system.

Memory allocation and deallocation in a smart card memory management system are implemented using a variety of services. At the lowest level, these services include requesting and



**Figure 13.16** Basic structure of a file allocation table (FAT) for file management in a smart card. The end of the user data is indicated in the FAT by an EOF (end-of-file) marker

releasing memory, increasing memory, reading and writing data, and writing data as an atomic operation. The file management interface built on top of these services usually includes services for creating and deleting files, reading data from files, writing data to files, selecting the MF, selecting a higher-level DF, selecting a file using its FID, and selecting a file using its DF name.

### 13.7.3 Memory partitioning into pages

The limited number of write/erase cycles of nonvolatile memory and the partitioning of nonvolatile memory into pages create a general problem for all file management activities. This has a significant effect on the entire design of the file manager and the internal file structures. The sizes of file headers and file bodies must be matched to the predefined memory page size to prevent them from being split by page boundaries. File management data in memory must also be strictly separated from the actual data content of the files. Otherwise undesired interactions could occur between the management data in the header and the user data in the file body. These interactions could destroy the entire internal security structure of the smart card operating system. This is briefly illustrated by the following example.

Suppose the access conditions for a file containing secret, nonreadable keys are stored on the same memory page as the public, writable data of another file. If a write operation to this file is interrupted, for example by pulling the card out of the terminal, this will affect the access conditions stored on the same page. In the worst case, no access conditions at all will remain, and the file containing the secret keys can be read by everybody. It is thus fundamentally important to store the internal file structures for file management and the user data on separate memory pages.

### 13.7.4 DF separation

One way of understanding the role of a DF is to regard it as representing all of the memory available to a particular application. Within this region, the application operator is fully

responsible for his application and can essentially do and permit whatever he wishes. However, he must under no circumstances be able to access a memory region assigned to a different application from his own application, nor should he be able to read or alter data stored in another region. Consequently, some smart card operating systems have special mechanisms that always test every memory access to see whether the physical address is located within the limits of the current DF. If it is not, the process is terminated and a severe internal error is reported.

This address monitoring is currently performed by suitable software routines in the operating system, due to a general lack of hardware support. The security of this solution is naturally significantly lower than what could be achieved with suitable hardware, since it is more easily bypassed. In the future, high-performance smart card microcontrollers will also have memory management units (MMUs), as do all current PC CPUs.<sup>6</sup> Such units can be used to achieve secure control over memory accesses within a DF. Until then, the only option is to employ suitable operating system routines to monitor the address boundaries of the DFs. This principle of memory organization – storing all components of a DF in a single contiguous memory region – had to be abandoned in the development of recent operating systems because memory management would otherwise have been too inflexible.

### 13.7.5 Free memory management mechanisms

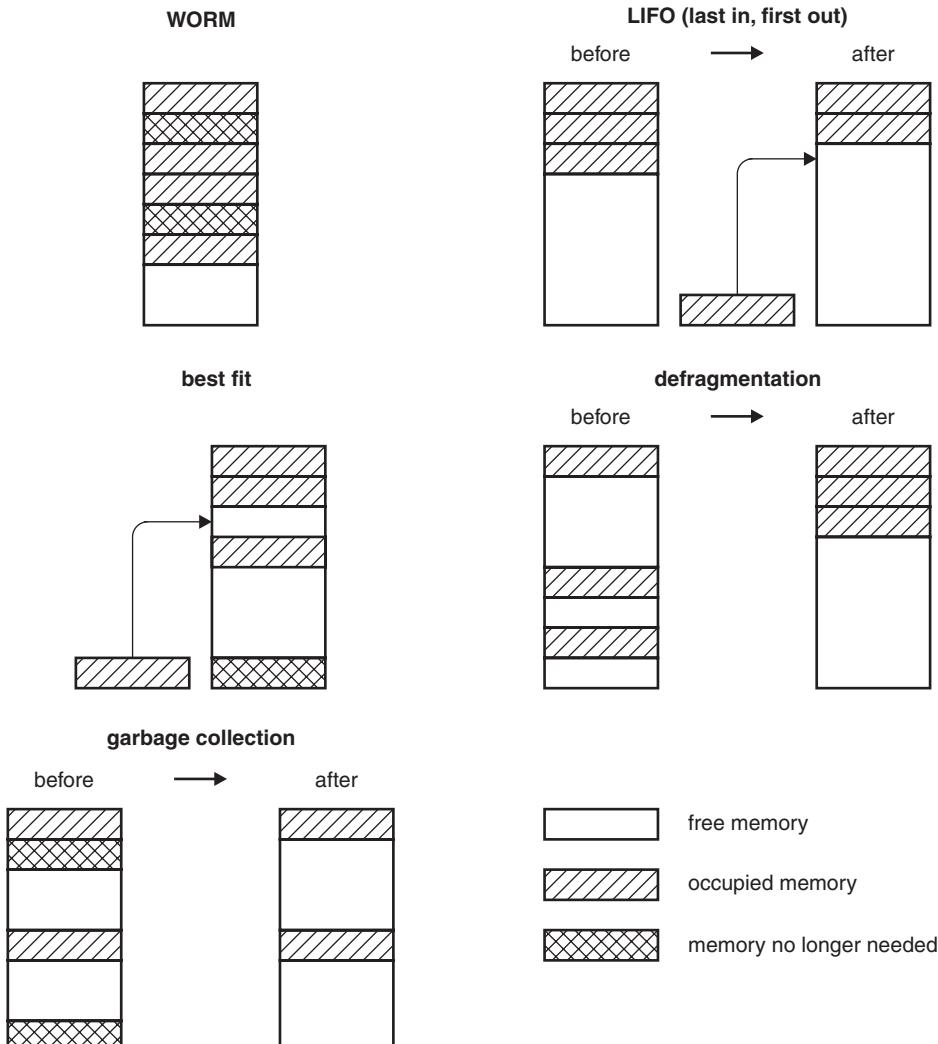
The small memory capacity of smart cards imposes major restrictions on smart card operating systems with regard to free memory management of the EEPROM. Only since the mid-1990s have operating systems been available that can create and subsequently delete files (DFs and EFs) after the card has been personalized. Given the secure nature of a smart card, this must naturally be protected in a cryptographically flawless manner. The ideal solution is to execute the appropriate commands in secure messaging mode after initial mutual authentication.

Free memory management must also take into account the fact that in the event of a sudden loss of power, which can for example occur if the card is pulled out of the terminal, the entire file tree must remain in a well-defined state. Particularly in such circumstances, the security of a smart card can completely collapse if file pointers suddenly become undefined. Satisfactory approaches to solving this problem once again involve atomic operations, although in this case such processes are relatively slow and require large memory buffers due to the large data volumes.

There are various strategies for realizing memory management and file management in smart cards, and they differ significantly in terms of their software implementation. This is described below using smart card memory management as an example.

Figure 13.17 on the next page shows the free memory management mechanisms described here. The simplest form of memory management is a sort of write once, read multiple (WORM) function. With this approach, memory space that has been allocated to a file remains occupied even after the file has been (logically) deleted. The management overhead is minimal with this approach. A somewhat more elaborate method from a software engineering point of view is memory management using the last-in, first-out (LIFO) principle. With this approach, the most recently created file can always be deleted, accompanied by the release of the space it occupied.

<sup>6</sup> See also Section 5.4, ‘Supplementary Hardware’, on page 93



**Figure 13.17** Typical memory management mechanisms for smart card operating systems. They are described in detail in the text

This method is often used in simple smart card operating systems. With the somewhat more sophisticated best-fit algorithm, the operating system always attempts to find the smallest suitable region of free memory when it creates a new file. If the file is deleted, this region again becomes free and can be used by other files. However, strong memory fragmentation occurs relatively quickly if files of various sizes are frequently created and deleted. As a result, large files can no longer be created because no contiguous region of free memory is large enough to hold the entire file, even if the total amount of free memory is sufficient.

This is precisely where defragmentation comes into play. If the memory becomes severely fragmented, this memory management process (which is relatively complex by smart card standards) repeatedly relocates files in memory until it arrives at a situation in which all of

the free memory forms a single contiguous block. The difficulty here comes from the fact that such an algorithm runs relatively slowly in smart cards because it must perform many time-intensive EEPROM write operations.

Garbage collection can in theory be regarded as independent of the processes just described. Operating either on demand or at regular intervals, garbage collection searches the entire memory for memory blocks that are no longer used. If a block that is no longer needed is found, garbage collection automatically allocates it to the free memory pool. A subsequent defragmentation process can be used to combine all of these small memory blocks into a large, contiguous region of free memory.

### 13.7.6 Quota mechanism

The basic idea of multiapplication smart cards is to combine several applications from different application providers in a smart card so the card user can use them for a wide variety of purposes. To achieve maximum flexibility, it must actually be possible for these applications to be loaded into the smart card by different entities and at different times. This is also desirable from the user's perspective because it allows the user to personally decide which applications he or she wants to load or delete at any given time. This requirement, which is by no means easy to achieve technically, must be implemented by the smart card operating system.

The functions defined in the Global Platform (GP) specification have become established worldwide as a standardized mechanism for loading and deleting arbitrary code-based applications.<sup>7</sup> To meet the requirements described above, they must be extended to include memory management functions. It must be possible for the card issuer to specify the maximum amounts of volatile and nonvolatile memory that can be allocated to each downloaded application and to adjust these figures as necessary. This function is called the quota mechanism, based on the corresponding Unix function. It is specified in ISO/IEC 7816-13, and in this regard we should mention that operating system producers and card issuers have also developed a number of special solutions to this issue that deviate from this standard, but have nevertheless become established.

The quota principle works as follows. As an extension to Global Platform functionality, the current usage and maximum permitted usage of volatile memory (RAM) and nonvolatile memory (EEPROM or flash) are maintained separately in the operating system. This includes memory for the program code (typically applets) and files of each of the applications. This is usually based on net data volumes, which means ignoring memory used for management data, in order to maintain the comparability of individual operating systems.

With applets, the quota mechanism typically comes into play with the Global Platform INSTALL command. Applets can be installed as long as there is enough memory available in the appropriate security domain for the appropriate area; otherwise installation is terminated with an error message. When an applet that has been installed in the smart card uses the new method to request additional memory, the quota mechanism monitors this request and either approves or denies the memory request.

For files, the quota mechanisms monitor the CREATE and DELETE commands and all commands that can request additional memory space (such as APPEND RECORD) or release

<sup>7</sup> See also Section 13.13, 'Application Management with Global Platform', on page 485

memory space (such as RESIZE). If execution of one of these commands in the application DF concerned would cause the maximum permitted amount of nonvolatile to be exceeded, the quota mechanism terminates command processing and informs the terminal of the reason for termination by means of a suitable return code.

Within the quota function, it is also necessary to communicate the corresponding memory usage limits and the amount of free memory still available to the outside world. This information is normally needed by the card management system. The GET DATA command and the tags of the memory values concerned can be used for this purpose. In principle, the PUT DATA command could be used here to adapt the memory size settings to new requirements, but in practice the INSTALL command with the option for changing registry settings is often used for this purpose instead. This is because the Global Platform registry is the preferred storage location for this memory restriction data. Naturally, all changes to quota values must be preceded by suitable authentication to prevent arbitrary (unauthorized) changes to the settings.

With quota control, it is certainly possible for the total amount of memory allocated to the various applications to be larger than the amount of memory actually available on the card. This is also a fairly realistic scenario, since not every application intended for multiapplication smart cards has to be present in a given card in its full version. The maximum memory values in the quota system are not guaranteed values that can always be used by the application concerned under all circumstances, but instead the maximum amount of memory the application can use if it is actually available. This means that after an application has been installed and put into use, it should request all the memory it needs as quickly as possible in order to avoid discovering much later that the smart card no longer has sufficient memory for the full scope of the application.

The quota mechanism, which imposes fairly severe technical demands and is only available in top-end smart card operating systems, gives issuers of multiapplication smart cards a powerful tool that enables them to sell memory space to other parties without having to fear that they might take over the entire memory capacity of the card for their own purposes. For precisely this reason, some form of memory control is an indispensable requirement for smart cards used in a multiapplication environment.

### 13.7.7 Data integrity

Another important consideration is ensuring data integrity. The file manager should always be able to test whether the data in the memory has accidentally changed, which could occur due to factors such as aging. To minimize the management overhead for this function, the degree of data redundancy and/or the extent of the supervisory protective functions should match the importance of the data. There is no need to protect all data with checksums as a matter of principle. Several data elements, such as a complete file header, can be protected as a group, or particularly important data elements can be protected individually. This primarily depends on how often the data elements are modified in nonvolatile memory and how much memory space the operating system designer is willing to sacrifice to ensure data integrity.

Error detection codes are used to ensure data integrity, and in particular to protect critical data elements such as data access conditions and file body pointers in file headers. Checksums based on CRCs are often used for this purpose because they can be computed relatively quickly and require relatively little program code. However, Reed–Solomon codes are often used to provide better protection against the typical failure characteristics of EEPROM and

flash memory cells. These codes are significantly better than CRC checksums for detecting the burst errors that typically occur when changes occur in an entire memory page.

### 13.7.8 Cross-application access

Certain smart card functions are only enabled after the card user has entered a PIN code. The memory capacity of the average person has certain limits, so it has become common practice to use only one PIN per smart card, even with multiapplication cards. Every application in the card thus uses this common PIN. It could be stored separately for each application in an internal EF, but this would require each of the stored PINs to have its own retry counter. If there are five applications, for example, and each application allows three attempts, a total of fifteen attempts to guess the PIN will be allowed. In many cases, this is not tolerable in terms of application design or security. Consequently, some operating systems allow cross-application access to PINs and keys.

This utilization of shared resources is in principle implemented in a manner similar to the alias mechanisms commonly used in PC operating systems. The main difference is that smart card operating systems only allow pointers to higher-level DFs, with the MF being the highest-level entity. It is thus not possible to access a PIN located in an arbitrary DF, but only one located in a higher level of the file hierarchy, such as the MF.

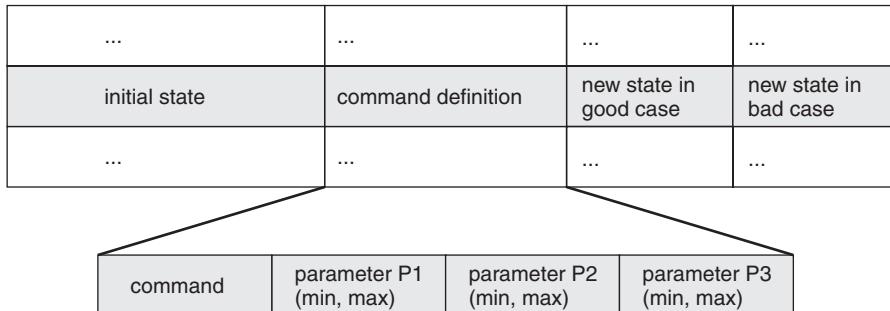
In the case of the above example of a single PIN and retry counter shared by several applications, a possible implementation is as follows. The PIN is stored in an internal EF located immediately below the MF. In the application, which is located in a DF below the MF, a reference to the location of the PIN is stored in an internal EF. The states resulting from successful and unsuccessful PIN comparisons are naturally stored in the PIN record in this DF, since they apply to only one particular application. If a PIN comparison is triggered, the VERIFY command first accesses the PIN record in the current DF, from which it sees that the PIN and its associated retry counter are located at a different level. It then uses the indicated PIN for the PIN comparison. The state of the currently selected DF stored in the internal EF is set according to the result of the comparison.

This method is presently supported by many smart card operating systems in various forms. Particularly for utilization of data that is shared by two or more applications, it provides a very elegant and cryptographically faultless solution. In addition to allowing PINs and keys to be used across several applications, some operating systems also offer an equivalent mechanism for EFs. This makes it possible to access global data directly in EFs located immediately below the MF without first deselecting the current DF.

## 13.8 SEQUENCE CONTROL

If a state machine must be implemented in an operating system, there are various ways in which it can be constructed. However, certain basic principles must be observed, independent of the operating system and its producer.

In the previously described layered operating system model, the state machine must be located after the command interpreter and before the actual execution of the command. The task of the state machine is to determine whether the received command may be executed in the present state. It does this by using a table. A basic principle here, as is usual with smart



**Figure 13.18** Example data structure elements for implementing a command sequence state machine

cards, is to use as little memory as possible to provide the state information. In addition, this information must be structured such that the actual state machine can be built using as little memory as possible.

The state machine needs a certain amount of information to analyze the command held in the I/O buffer. Figure 13.18 shows a possible structure of a smart card state table.

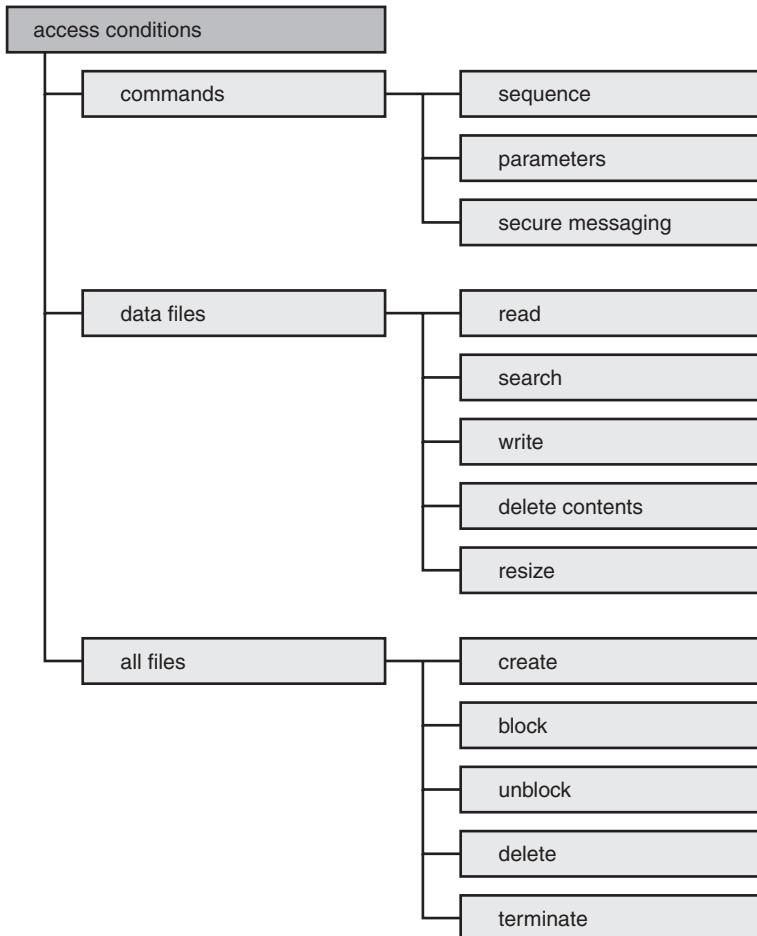
The first data element (initial state) contains the state for which the rest of data in the data structure must be processed. This data element could contain a number that directly defines the state in which all other information must be taken into account. This is followed by a subtable of all commands that are allowed in the initial state. In each state, it must be possible to allow individual commands, groups of commands, all commands, or no commands.

The allowed parameters of each command follow the command definition in the table structure. In these data elements, it must be possible to define individual values and ranges of values for the parameters. For example, ID code of the READ BINARY command could be located in the command field, the minimum and maximum offsets for read access to transparent data in the P1 and P2 parameter fields, and the lower and upper length limits in the P3 parameter field. As several entries may be present in this subtable for each state, additional commands and their parameters could be defined after READ BINARY.

Each table entry concludes with the new state to be assumed if the command is successfully executed, which means if command execution completes without any errors. The data structure in the example also allows a state to be defined that is to be assumed if command execution is not successful. In order to maintain a high degree of flexibility in the state machine, it must be possible to specify subsequent states either absolutely or relatively. Here ‘relatively’ means that the new state is determined by adding a value to the initial state value or subtracting a value from the initial state value, while ‘absolutely’ means that the value of the new state is determined directly without reference to the value of the initial state.

In principle, there are no limits to how a state machine can be constructed. The data structure illustrated here is quite suitable for use in relatively sophisticated smart card operating systems.

In principle, every possible state machine diagram can be represented in a smart card using the described data structure and a corresponding state machine. Naturally, individual files also have their own supplementary protection against unauthorized reading or writing in the form of access conditions for commands. Nevertheless, sequential control for commands can provide an additional higher-level mechanism that complements this object-oriented protection and thus increases the security of the system. This is actually the primary benefit of using state machines in smart cards.

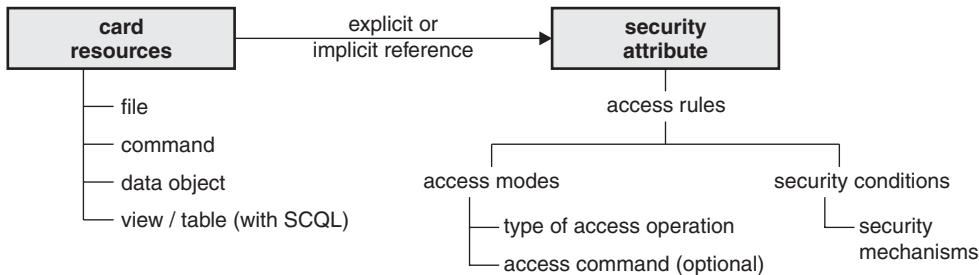


**Figure 13.19** Classification of command and file access conditions according to ISO/IEC 7816-9

## 13.9 ISO/IEC 7816-9 RESOURCE ACCESS

ISO/IEC 7816-9 defines access conditions for commands and files as shown in Figure 13.19. The allowed types of access to files can be specified using state-oriented or command-oriented access conditions. With state-oriented conditions, the current security state is compared with the corresponding access condition of the file using a definable logical comparison. There are two types of current security state, which are the global security state (the security state of the smart card as a whole) and the local security state (the security state of the currently selected directory). By contrast, with command-oriented access conditions the access table in the file contains information about the commands that must be successfully executed prior to each type of access.

Both types of access conditions (state-oriented and command-oriented) have been and will continue to be supported in various forms by commercial smart card operating systems. Until



**Figure 13.20** Classification of card resources and associated security attributes according to ISO/IEC 7816-9

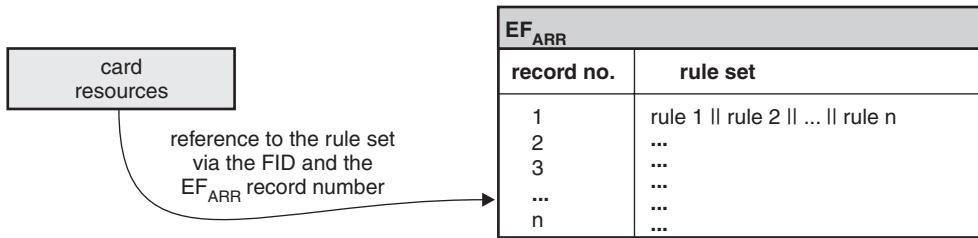
recently, the biggest problem was the large variety of implementations and approaches that have been used. The objective of the ISO/IEC 7816-9 standard is to define a uniform approach to accessing resources in smart cards, and it includes a section specifically devoted to this subject that specifies a very powerful model for access conditions of files as well as commands and data objects. Unfortunately, this model is also complicated. This universal access model unifies both state- and command-oriented access conditions, and to this it adds the possibility of specifying specific command sequences. Furthermore, ISO/IEC 7816-9 allows specific data object tags to be used to define an access control state machine. The concept is based entirely on TLV-coded data objects, which are known for their ability to enable generation of very flexible and elegant IT structures.

As illustrated in Figure 13.20, ISO/IEC 7816-9 defines security attributes (SAs) that can be used to control granting or denying access and attaining specific security states in the smart card. These security attributes control access to card resources such as files, commands and data objects, as well as SCQL tables and views.

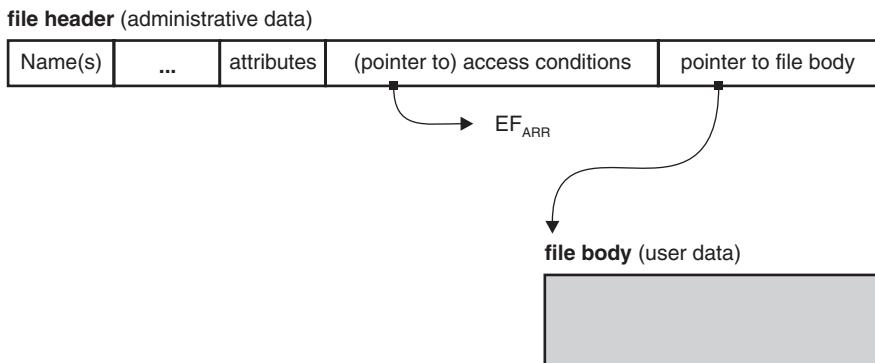
The access control principle is relatively simple. The resource to be protected is assigned a reference (which may be explicit or implicit) to one or more security attributes. These attributes consist of one or more access rules (ARs), which in turn are composed of access modes (AMs) and security conditions (SCs). Each access mode specifies a type of access, such as read or write, while the security conditions specify the security mechanisms (SMs) needed to allow the access conditions to be satisfied. An additional element that may be incorporated in the security rules is the current security environment (SE).

All ISO/IEC 7816-9 security attributes are stored either in compact format to save memory space or as regular TLV-coded data objects in expanded format. Both formats provide similar access protection functionality, but the expanded format offers significantly more flexibility. For instance, with this format it is possible to generate a detailed specification of commands and associated parameters for accessing resources.

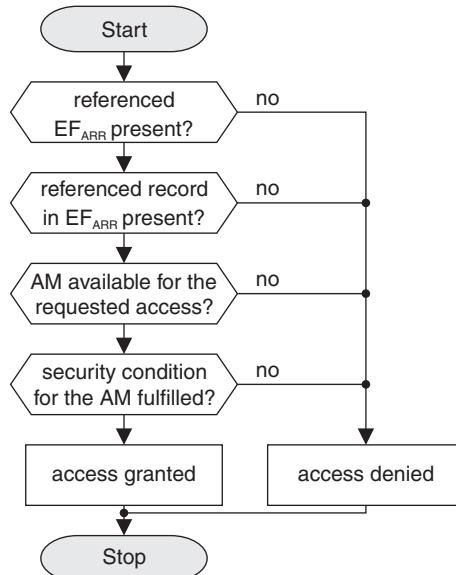
The access rules are stored in one or more EFs with linear variable structure. These can be internal EFs (EFIs) or working EFs (EFWs), and they are given the name EF<sub>ARR</sub> (access rule reference). The EF file identifier (FID) can be freely chosen. If an EFI is used to store access rules, this EF is accessed implicitly via the operating system each time an access condition occurs. If an EFW is used, the card resource to be protected has a reference to the FID of the EF<sub>ARR</sub>, as depicted in Figures 3.21 and 13.22. In both cases, the card resource to be protected references the number of the record in the EF<sub>ARR</sub> that holds the appropriate access rule. The main advantage of a selectable EF<sub>ARR</sub> (that is, an EFW) is that its contents can be modified



**Figure 13.21** Linking arbitrary resources of a smart card to associated access rules stored in an EF<sub>ARR</sub> file



**Figure 13.22** Linking an arbitrary file header in the file management system of a smart card to the associated access rules stored in an EF<sub>ARR</sub> file



**Figure 13.23** Flow chart of the essential queries for checking file access according to the ISO/IEC 7816-9 access model

**Table 13.6** DF access mode (AM) byte codes defined by ISO/IEC 7816-9

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	...	...	...	...	...	...	...	b7 ... b1 according to this table
1	...	...	...	...	...	...	...	b3 ... b1 according to this table and b7 ... b4 proprietary
...	1	...	...	...	...	...	...	DELETE FILE (current file)
...	...	1	...	...	...	...	...	TERMINATE CARD USAGE (MF), TERMINATE DF
...	...	...	1	...	...	...	...	ACTIVATE FILE
...	...	...	...	1	...	...	...	DEACTIVATE FILE
...	...	...	...	...	1	...	...	CREATE FILE (create a DF)
...	...	...	...	...	...	1	...	CREATE FILE (create an EF)
...	...	...	...	...	...	...	1	DELETE FILE (subordinate file)

**Table 13.7** Access mode data object (AM DO) codes defined by ISO/IEC 7816-9.  
These codes are used in the extended format

Code	Length	Meaning
'80'	1	AM byte from Tables 13.6 and 13.8 on the following page
'81' ... '8F'	x	Description of CLA    INS    P1    P2 for specifying the parameters of access commands
'9C'	x	Proprietary description of a state machine

using standard commands with suitable access conditions. This creates enormous flexibility, since the access conditions for the smart card resources can be modified whenever desired.

Here it should be noted that it is not necessary to store a specific record in the EF<sub>ARR</sub> for every EF. It is fully sufficient to store a single record for all EFs with identical access conditions and reference this record from these EFs. This considerably reduces the number of records needed in the EF<sub>ARR</sub>.

The link between the EF and the EF<sub>ARR</sub> works in only one direction; it cannot be used both ways. It is not possible to determine from a specific record in the EF<sub>ARR</sub> which EF or EFs reference this record. This impacts file management in the sense that a record in the EF<sub>ARR</sub> linked to an EF cannot be deleted when the EF is deleted, since it may also be referenced by one or more other EFs.

The following procedure is used when a command needs to access a file. First the operating system checks whether the explicitly or implicitly referenced EF<sub>ARR</sub> and the corresponding record are present. If they are not, access is denied. Next, the EF<sub>ARR</sub> is searched for an access mode (AM) data object for the requested access. If this data object is found, the specified security condition (SC) is checked; otherwise access is again denied. If the security condition is met, access to the file with the corresponding command is granted.

To help clarify this explanation, some typical examples of EF<sub>ARR</sub> entries are shown in Tables 13.6 through 13.16 on page 477–481 with full coding in both expanded format and compact format. In summary, it can be noted with regard to access rules compliant with ISO/IEC 7816-9 that, although this system is very powerful and highly flexible, it also imposes

**Table 13.8** EF access mode (AM) byte codes defined by ISO/IEC 7816-9

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	...	...	...	...	...	...	...	b7 ... b1 according to this table
1	...	...	...	...	...	...	...	b3 ... b1 according to this table and b7 ... b4 proprietary
...	1	...	...	...	...	...	...	DELETE FILE (current file)
...	...	1	...	...	...	...	...	TERMINATE EF
...	...	...	1	...	...	...	...	ACTIVATE FILE
...	...	...	...	1	...	...	...	DEACTIVATE FILE
...	...	...	...	...	1	...	...	WRITE BINARY, WRITE RECORD, APPEND RECORD
...	...	...	...	...	...	1	...	UPDATE BINARY, UPDATE RECORD, ERASE BINARY
...	...	...	...	...	...	...	1	READ BINARY, READ RECORD, SEARCH BINARY, SEARCH RECORD

**Table 13.9** Security condition (SC) codes defined by ISO/IEC 7816-9

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	0	0	0	0	0	0	Access is always allowed
1	1	1	1	1	1	1	1	Access is never allowed
...	...	...	...	0	0	0	0	No security environment reference
...	...	...	...	...	0001 ... 1110	...	...	Security environment number
...	...	...	...	1	1	1	1	RFU
0	...	...	...	...	...	...	...	At least one condition must be satisfied
1	...	...	...	...	...	...	...	All conditions must be satisfied
...	1	...	...	...	...	...	...	Secure messaging
...	...	1	...	...	...	...	...	External authentication
...	...	...	1	...	...	...	...	User authentication (e.g. PIN entry)

**Table 13.10** Security condition data object (SC DO) codes defined by ISO/IEC 7816-9. These codes are used in the extended format

Code	Length	Meaning
'90'	0	Access is always allowed
'97'	0	Access is never allowed
'A4'	x	Control reference template (CRT) for authentication (external authentication or user authentication)
'B4', 'B6', 'B8'	x	Control reference template (CRT) for a command and/or response with secure messaging
'9E'	x	Security condition according to Table 13.9
'A0'	x	The stated security conditions are to be combined in a logical OR manner
'AF'	x	The stated security conditions are to be combined in a logical AND manner

**Table 13.11** Usage qualifier date object codes for use in a control reference template (CRT) defined by ISO/IEC 7816-9. These codes are used in the extended format. The usage qualifier tag is ‘95’

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	...	...	...	...	...	...	...	Verification, encryption and external authentication
...	1	...	...	...	...	...	...	Computation, decryption, and internal authentication
...	...	1	...	...	...	...	...	Secure messaging response
...	...	...	1	...	...	...	...	Secure messaging command
...	...	...	...	1	...	...	...	Knowledge-based user authentication (e.g. PIN)
...	...	...	...	...	1	...	...	Biometric user authentication
...	...	...	...	...	...	x	x	RFU

**Table 13.12** File control parameter (FCP) codes defined by ISO/IEC 7816-4 and ISO/IEC 7816-9. The FCP tag is ‘62’

Code	Length	Meaning
‘80’	2	Number of data bytes (excluding structure data) in a transparent EF
‘81’	2	Number of data bytes including structure data
‘83’	2	FID
‘84’	1 ... 16	DF name
‘88’	1	SFI coded in bits 8–4; bits 3–1 set to 0
‘8A’	1	Life cycle status integer (LCSI)
‘8C’	variable	Security attribute in compact format
‘AB’	variable	Security attribute in expanded format

**Table 13.13** Control reference data object (CR DO) codes defined by ISO/IEC 7816-4

Code	Meaning
‘80’	Algorithm reference
‘81’	File reference: FID or file path
‘82’	File reference: DF name
‘83’	Key reference: for direct use
‘84’	Key reference: for computing a session key

**Table 13.14** Example of the content of an EF<sub>ARR</sub> record in compact format. This record specifies the access conditions for UPDATE BINARY and READ BINARY for a file (EF). All other types of file access are automatically prohibited

Data item	Designation	Meaning
‘8C’	Tag	The tag ‘8C’ identifies an access rule in compact format
‘03’	Length	The length of the following data is 3 bytes.
‘03’	AM	The following security conditions refer to UPDATE BINARY and READ BINARY because ‘03’ = 0000 0011
‘00’	SC	No security condition is specified for UPDATE BINARY, which means that the EF may always be written
‘00’	SC	No security condition is specified for READ BINARY, which means that the EF may always be read

**Table 13.15** Example of the content of an EF<sub>ARR</sub> record in compact format. This record specifies the access conditions for ACTIVATE FILE, DEACTIVATE FILE and READ BINARY for a file (EF). All other types of file access are automatically prohibited

Data item	Designation	Meaning
'8C'	Tag	The tag '8C' identifies an access rule in compact format
'04'	Length	The length of the following data is 4 bytes.
'19'	AM	The following security conditions refer to ACTIVATE FILE, DEACTIVATE FILE, and READ BINARY, ... because '19' = 0001 1001
'90'	SC	Prior user authentication, such as PIN entry, is necessary as a security condition for ACTIVATE FILE because '90' = 1001 0000. The PIN needed for this is implicitly known to the operating system
'90'	SC	Prior user authentication, such as PIN entry, is necessary as a security condition for DEACTIVATE FILE because '90' = 1001 0000. The PIN needed for this is implicitly known to the operating system
'00'	SC	No security condition is specified for READ BINARY, ..., which means that the EF may always be read

a corresponding toll on the smart card operating system in the form of additional memory space. Furthermore, in practice it is nearly impossible to manually code access conditions based on this rule-based approach, even for simple smart card applications, without the assistance of suitable software tools. Nevertheless, this concept for controlling access to smart card resources will become established worldwide because the advantages of flexibility and standardization outweigh the drawbacks.

## 13.10 ATOMIC OPERATIONS

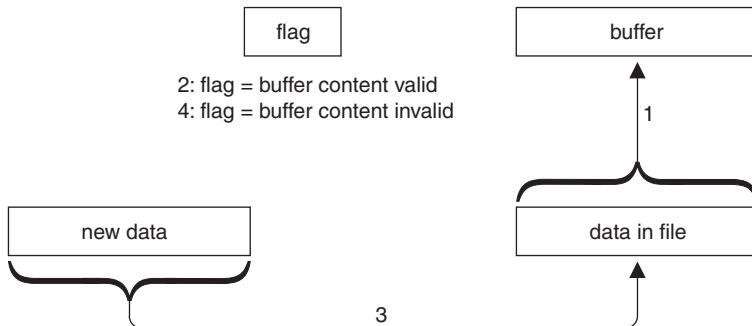
A requirement that is frequently imposed on smart card microcontroller software is that certain parts of it must execute either completely or not at all. Processes that are indivisible and thus fulfill this requirement are called atomic operations. They always occur in connection with EEPROM write routines.

Atomic operations are based on the idea of ensuring that when an EEPROM write operation occurs, the data in question must never be written only partially. This could happen if, for example, the user pulls the card out of the terminal at the wrong time or there is a sudden power failure. Since the smart card has no buffer for electrical power, the software in the card would immediately lose its ability to do anything at all in such cases.

Particularly in the case of electronic purses in smart cards, it is essential to ensure that file contents are complete and correct at all times. For example, failure to fully update the balance of an electronic purse if the card is suddenly pulled out of the terminal would have fatal consequences. Corresponding entries in log files must also always be complete. As the hardware of smart cards does not support atomic operations, they must be implemented in software. The methods used for this are in principle not new. They have been used for a long time with databases and hard disk drives. The basic procedure of a method that is used in smart card operating systems is described here. This error recovery method is transparent to the outside world and thus does not require any changes to existing applications.

**Table 13.16** Example of the content of an EF<sub>ARR</sub> record coded in expanded format in accordance with ISO/IEC 7816-9. This specification allows access to the corresponding EF using READ BINARY, etc. at all times. UPDATE BINARY, . . . are only allowed after prior successful verification of PIN 1 or PIN 2. All other types of file access are automatically prohibited

Data item	Designation	Meaning
'AB'	Tag	The tag 'AB' identifies an access rule in expanded format
'1A'	Length	The length of the following data is 26 bytes
'80'	AM DO	The code '80' indicates that the data object for the AM DO access conditions contains a byte with the access conditions (AM byte)
'01'	Length	The length of the following data is 1 byte
'02'	AM	The following security conditions refer to UPDATE BINARY, . . . because '02' = 0000 0010
'A0'	SC	The following SCs are combined in a logical OR fashion
'10'	Length	The length of the following data is 16 bytes ('10')
'A4'	CRT	The tag 'A4' indicates that the following data contains information about the necessary authorizations. This data forms a control reference template (CRT)
'06'	Length	The length of the following data is 6 bytes
'83'	Key reference	The code '83' indicates that this is a CRT data object for referencing a key
'01'	Length	The length of the following data is 1 byte
'01'	Key number	Key number 1 is to be used
'95'	Usage qualifier DO	The tag '95' (1001 0000) indicates that this is a CRT data object for a usage qualifier
'01'	Length	The length of the following data is 1 byte
'08'	Usage qualifier	Knowledge-based user authentication (i.e. a PIN) is specified as a usage qualifier because '08' = 0000 1000
'A4'	CRT DO	The tag 'A4' indicates that the following data contains information about the necessary authorizations. This data forms a control reference template (CRT)
'06'	Length	The length of the following data is 6 bytes
'83'	Key reference	The tag '83' indicates that this is a CRT data object for referencing a key
'01'	Length	The length of the following data is 1 byte
'02'	Key number	Key number 2 is to be used
'95'	Usage qualifier DO	The tag '95' (1001 0000) indicates that this is a CRT data object for a usage qualifier
'01'	Length	The length of the following data is 1 byte
'08'	Usage qualifier	Knowledge-based user authentication (i.e. a PIN) is specified as a usage qualifier because '08' = 0000 1000
'80'	AM DO	The tag '80' indicates that this is an access mode (AM) data object
'01'	Length	The length of the following data is 1 byte
'01'	AM	The following security conditions refer to UPDATE BINARY, . . . because '01' = 0000 0001
'90'	SC DO	The tag '90' indicates that accesses are always allowed
'00'	Length	The length of the following data is 0 bytes. This indicates that a security condition for READ BINARY, . . . is not necessary, which means that the EF may always be read



**Figure 13.24** Example of a possible implementation of atomic operations in a smart card operating system. This method can of course be cascaded to process multiple data elements in parallel

The operation of this method can be described using an example in which data destined for a particular file is sent to a smart card via its interface. This is a typical operation with an UPDATE BINARY command, for example. The process is explained below and illustrated in Figure 13.24. The operating system creates a buffer in its EEPROM or flash memory that is large enough to hold all the necessary data. This buffer has a status flag, which is also stored in the EEPROM or flash memory. The flag can be set to two states: ‘data in buffer valid’ or ‘data in buffer not valid’. In addition to the buffer, there must also be suitable locations in memory for the target address and the current amount of buffered data.

The specific sequence of events is as follows. In the first step, the data in the region starting at the target address, which may for example be located in a file, is copied to the buffer along with its physical address and length. The buffer flag is then set to ‘data in buffer valid’. In the next step, the operating system copies the new data to the intended address and then sets the buffer flag back to ‘data in buffer not valid’. When the operating system starts up, it checks the buffer flag before sending the ATR. If the flag is set to ‘data in buffer valid’, the data in the buffer is automatically written to the memory region specified by the stored address and length data.

This mechanism ensures that the data in the file is valid under all conditions. If a routine is aborted at any time during program execution, the data in the nonvolatile memory of the smart card can always be restored. For example, if the cardholder pulls the card out of the terminal at the third step of the process – while new data is being written to the EEPROM or flash memory – only part of the new data will be present in the file. When the card is again activated in a subsequent session, the operating system notices that there is valid data in the buffer and copies it to the appropriate location. This restores the original status of the file, so the contents of all files in the nonvolatile memory are consistent. The initial waiting time between the individual bytes of the ATR provides an excellent opportunity to make this correction.<sup>8</sup>

The procedure described above has two serious drawbacks. The first is that the buffer will have the heaviest write/erase wear of all regions in the EEPROM or flash memory. As the number of write/erase cycles for any given region of the nonvolatile memory is limited, there is a high probability that this important buffer region will be the first part of the EEPROM or

<sup>8</sup> See also Section 8.1, ‘Answer to Reset’, on page 203

flash memory to start showing write errors. Such errors would mean that the smart card could no longer be used, since the integrity of the data would no longer be assured. This problem can be remedied by creating a buffer with a cyclic structure so the same region is not written every time. Unfortunately, this means that a rather large amount of nonvolatile memory must be allocated to the buffer. Another disadvantage of this implementation of atomic operations is prolonged program execution time due to the required write accesses to the buffer. In the worst case, access can take three times as long with this mechanism as it would with direct writing to the EEPROM or flash memory. It is thus common practice to restrict this buffering of write accesses to specific files or data elements instead of buffering all EEPROM or flash memory accesses. In the case of files, this can be specified by an attribute in the header of each file.

This method can very easily be extended to enable writing several data elements to the buffer instead of only one. If this is done, it is even possible to ensure that write accesses to several different files or data elements are performed either completely or not at all. Almost all modern smart cards operating systems, such as Java Card, support atomic operations, and they may also allow relatively long sequences in the program flow to be marked as atomic in order to protect them against power interruptions.

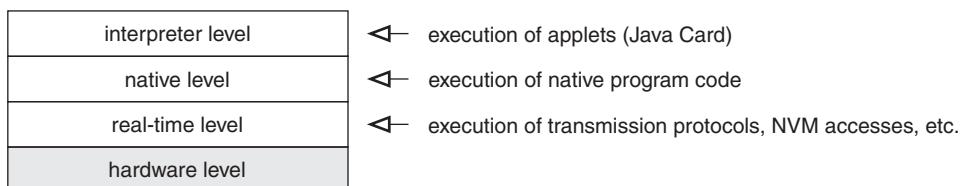
## 13.11 MULTITASKING

As long as smart card operating systems only had a single interface to the outside world, the process of exchanging commands and responses was strictly sequential. There was no need to be able to process several tasks in parallel, since a response always followed each command after the completion of command processing. This situation changed around 2007 with the introduction of additional interfaces such as Single-Wire Protocol (SWP) and USB. It was suddenly necessary to support at least two interfaces simultaneously, along with their associated applications, and there was no choice but to introduce a form of multitasking in the operating systems.

Multitasking had always been avoided in the past because there was no hard technical requirement for it, and in part because it requires significantly more processing power and RAM. In a smart card with two or more interfaces, the operating system must support these interfaces with very short response times and communicate with the associated applications via these interfaces.

Multitasking, which means quasi-concurrent execution of several different programs, is often structured in three layers in smart card operating systems, as illustrated in Figure 13.25.

The multitasking layer closest to the hardware consists of the interface drivers. Particularly with high communication data rates, they must meet stringent real-time requirements. This



**Figure 13.25** Typical layer structure for multitasking in a smart card operating system

can only be achieved by using interrupts from the corresponding hardware components; it is not possible using time-slicing methods. These interrupts have the highest priority in the system, as otherwise there would be a risk of communication failure.

The layer above this consists of the executable native program code in the smart card. This is usually part of the operating system, and it is processed using a time-slicing method. This means that each of the tasks obtains access to the processor for a certain length of time at regular intervals. This naturally requires suitable hardware support, since among other things the contents of all CPU registers must be saved each time a task swap occurs. Task swapping can be implemented using a timer with a suitable interrupt, although it is desirable to use a memory management unit (MMU) as well to ensure clear separation of the memory regions of the individual tasks. This also increases the robustness of the operating system, since using an MMU prevents the tasks from penetrating the working memory of other tasks.

Hardware support is also necessary in another area, which is controlling write access to nonvolatile memory. This must be possible without any processor involvement, as otherwise conflicts could arise during task swapping. Naturally, it must also be possible to execute program code from nonvolatile memory while data is being written to the same memory in parallel. Quite a few smart card operating systems cannot fulfill this requirement.

The top layer of the smart card multitasking hierarchy is occupied by interpreted programs, such as Java bytecode in a Java card. Quasi-parallel processing of several such applications by the Java virtual machine (VM) can be controlled directly by the VM software. The real-time requirements in this layer are relatively relaxed, and they can certainly be met in a satisfactory manner with a software solution of this sort. Alternatively, several instances of the Java VM can be launched and provided with processing time by the middle layer using time slicing.

If tasks with differing priorities must be processed, a simple time-slicing scheme is not adequate. In such cases it must be possible to process the tasks concerned using a priority-driven scheme. This means that the scheduler allocates processing time to the individual tasks according to their priority.

A topic that is closely related to multitasking, but nevertheless separate, is memory protection. This involves confining each task to its own, individually allocated memory space. This results in complete task partitioning, which distinctly increases the robustness and security of the applications. However, this sort of memory protection must be supported by the hardware of the smart card microcontroller. The appropriate component for this is the memory management unit (MMU).

## 13.12 PERFORMANCE

The performance of smart card operating systems is rarely a significant consideration in the native code domain because it usually lies in a range that has no negative effect on users. Applications that use asymmetric cryptographic algorithms form an exception to this rule, as it is certainly possible for their execution times to be noticeable to users, depending on the key length that is used. Incidentally, this is one of the reasons for exercising prudence when increasing key lengths. Performance problems can certainly arise when smart cards using contactless interfaces are used. In this connection, the activation time of the smart card after it enters the terminal field is a key issue.

In the Java card domain, adequate performance can certainly be an issue in practice because interpretation of program code in applets is naturally slower than direct execution of native

code by a processor. In addition, modern Java cards often have functions with significantly larger scope than the functions of native-code smart cards and are thus more time-intensive.

A dual approach is usually taken to optimizing the performance of a smart card operating system. The byte code execution speed of the Java VM is analyzed by performing measurements. If the measured values are compared with the frequency of execution of specific byte codes, it is easy to determine which byte codes need speed optimization.<sup>9</sup>

In addition, the execution speeds of typical commands and command sequences are measured. Here again the most attention is given to the commands that occur most often in a particular application. With a SIM or USIM card, these are the SELECT, READ BINARY and READ RECORD commands.<sup>10</sup> The activation sequence immediately after a mobile telephone is switched on serves as an example of a large variety of typical command sequences. A large volume of data is read from the smart card during this process, which consequently takes a considerable length of time. If the analysis reveals unusually long processing times in certain places, they must be improved.

Of course, optimizing the performance of a smart card operating system does not come for free; it often costs a considerable amount of development time, and it usually costs memory space. There are many different approaches that can be taken to this problem, most of which lie at the level of software development for embedded systems. One approach is caching the NVM data for both reading and writing.<sup>11</sup> However, this increases RAM usage, and great care must be taken to ensure that the cached data is always consistent with the data in the files. Consideration must also be given to the fact that power to the smart card can be interrupted at any time, which must never be allowed to result in corrupted or inconsistent data. This simple example clearly illustrates the difficulties that must be dealt with in performance optimization.

In practice, performance optimization is more often triggered by the subjective impressions of smart card users than by objective measurements. As long as the execution times of certain transactions remain below a certain threshold, optimization is not regarded as necessary. However, an urgent need to optimize an existing application, or even an existing operating system, arises as soon as this threshold is exceeded. The difficulties here are increased by the fact that the threshold is almost always purely subjective and can rarely be expressed in numbers in practice.

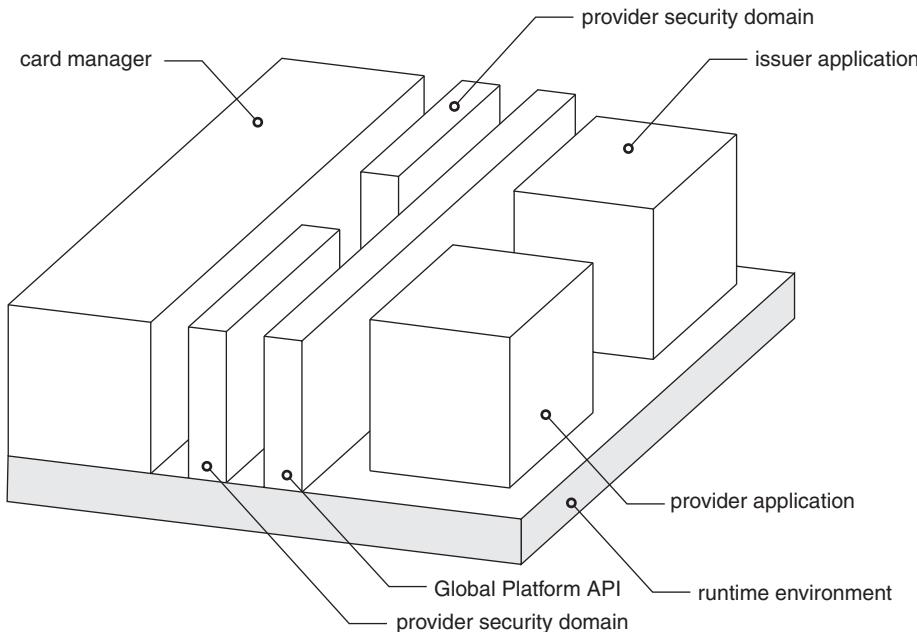
## 13.13 APPLICATION MANAGEMENT WITH GLOBAL PLATFORM

As a result of its activities as one of the largest card issuers, Visa International was confronted relatively early with the problems associated with managing multiple applications from a wide variety of sources in multiapplication smart cards. This led to the generation of the Visa Open Platform (VOP) specification, which defines an interface in smart card operating systems that is intended to be used to manage smart card applications. The publisher of this specification since 1999 is the Global Platform Committee [Global Platform], whose task is to standardize technologies for multiapplication smart cards. The name of the specification was also changed to Global Platform (GP) at that time. The GP specification is the most important

<sup>9</sup> In this connection, see also [Douin 04]

<sup>10</sup> See also Section 11.17, ‘Command Processing Times’, on page 407

<sup>11</sup> The time advantage of a read cache arises from the fact that for certain data the operating system can avoid the time-intensive use of complex file management system structures



**Figure 13.26** Basic architecture and components of Global Platform

international specification for application management in multiapplication smart cards, and it can be obtained free of charge from the Web server of the Global Platform Committee.

The GP specification is intentionally independent of any particular operating system, which allows it to be supported by all types of smart card operating systems, both proprietary and open (such as Multos and Java Card). In practice, however, the GP specification has primarily become the *de facto* standard for loading and managing Java-based applications with the Java Card operating system. For instance, the ETSI TS 43.019 standard regards Global Platform as the standard interface for downloading applications.

The purpose of the GP specification is to provide card issuers with mechanisms for securely managing third-party applications in the smart cards they issue. To this end, Global Platform defines the basic architecture of a multiapplication smart card. This is shown in Figure 13.26, and it is very important for understanding these mechanisms.

The runtime environment of the card's operating system forms the foundation for all applications. It provides application partitioning, a hardware-independent interface (API), and memory space for the data and program code of the various applications. The card manager is built on this foundation. It is the central component of Global Platform and consists of the Global Platform environment (also called OPEN); the issuer security domain, which can be selected using a freely definable AID; and Cardholder verification management (CVM). CVM provides a PIN that can be used by all applications.

The Global Platform environment, which is also called OPEN, manages the applications on the card (card content management) and handles selection of the applications on the logical channels of the card and dispatching of APDUs coming from the outside world to their associated applications. In addition, the Global Platform environment provides the applications

with interfaces to the system's oncard services and interfaces to the installed security domain applications.

In order to manage the data related to the various security domains, applications, component life cycle stages, component privileges, and their links on the smart card, the card manager has a data region called the card registry. This is the central location in the smart card for managing all data related to Global Platform.

### 13.13.1 Security domains

A security domain is a privileged application that manages keys and provides cryptographic services. This privilege, which is stored in the card registry, determines the rights and duties of the security domain. Each security domain is the representative of an application provider for matters related to informatics and security.

A security domain implements a secure communication function that fulfills several tasks. During mutual authentication of the user and the card, the communication keys are computed using random numbers and a derivation rule. After this, commands and responses can be protected by a cryptographic checksum and the data they contain can be encrypted. This secure communication can also be provided to applications as a service. This relieves the applications of the need to implement their own secure communication and enables several applications to use the keys of a single security domain.

An application with the security domain privilege is generally called a security domain. It provides a communication service using a secure channel (secure messaging) to applications and manages the keys necessary for this. A nonprivileged application that wishes to use this service and thus the keys must be linked to the security domain concerned. This link is managed in the card registry. An application linked to a security domain is called an associated security domain.

An application that is associated with a security domain can be personalized via the security domain. For this purpose, the application must implement the application interface used by the security domain to transfer the data to the application.

An application with the data authentication pattern (DAP) security domain privilege checks the cryptographic checksums of downloaded applications for the card manager's card content manager and manages the associated keys. If one or more DAPs are presented when an application is loaded in the card, the card content manager calls these security domains and they verify the correctness of the loaded data. If a DAP cannot be verified, loading is terminated with an error. If no DAP is presented when an application is loaded, the application data is not checked. The mandated DAP privilege is an extension to this. If one or more security domains are installed with this privilege, the card content manager compels the presentation of the necessary DAPs. The security domain privilege and the DAP security domain privilege are mutually independent, which means that an application can offer a secure channel service, a DAP service, or both.

An application with the delegated management security domain privilege is authorized to perform content management tasks on behalf of the card issuer; the card issuer delegates its management authority to this security domain. This means that a security domain with the delegated management privilege is authorized to load, install, and delete applications with the help of the card content manager. However, this authorization is restricted because every command from the card issuer must be signed with a cryptographic checksum (called a token)

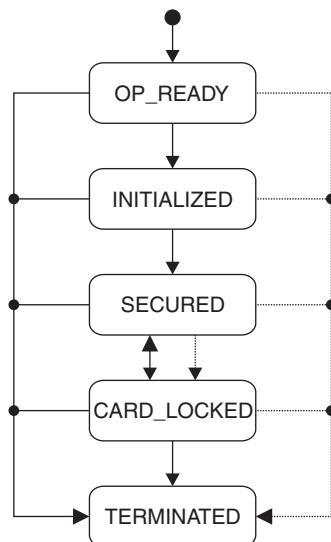
and installation and deleting are only allowed for autoloaded applications. Application linking is managed in the registry with an associated security domain link. The token is checked by the Card Content Manager and a confirmation (called a receipt) is issued after successful processing of the command. The receipt is a cryptographic checksum computed over certain parts of the processed command. The keys used for token verification and receipt generation are managed by the card manager.

As applications that are loaded by a delegated management security domain are linked to this security domain, the security domain must support secure channels for these applications. Consequently, every application with the delegated management security domain privilege also has the security domain privilege.

### 13.13.2 Issuer security domain

The issuer security domain is the first security domain in a Global Platform card, and it is defined as mandatory by the specification. The issuer security domain is the IT representative of the card issuer in the multiapplication smart card. The issuer security domain has the delegated management security domain privilege. However, as a representative of the issuer it does not need a token and does not generate a receipt.

The issuer security domain also manages the card status as illustrated in Figure 13.27. After the card is activated for the first time, the issuer security domain has initial keys for secure communication and is in the OP\_READY state, in which it can receive Global Platform commands. After this it initializes data and other keys and attains the state INITIALIZED.



**Figure 13.27** The states in the life cycle of a smart card compliant with Global Platform. The solid lines indicate state transitions controlled by the issuer security domain, while the dotted lines indicate state transitions that can be initiated by an application

After applications have been loaded, installed and personalized, the issuer security domain is put in the secure state, which is called SECURED. In this state, secure communication is used, the user must be authenticated, and cryptographic checksums (MACs) must be used to protect all commands. The card can now be issued to the user. The card can be blocked by a command or by a privileged application, in which case its status is LOCKED. In this state only the issuer security domain can be selected, and its command set is severely restricted. Card blocking can be rescinded by a command. At the end of the card life cycle, the card is terminated and attains the TERMINATED state. Here again, only the issuer security domain can be selected, and it only allows reading of certain data using the GET DATA command. All other commands are no longer available. Additional security domains can be downloaded via the issuer security domain.

### 13.13.3 Global Platform API

The Global Platform API (GP API) gives the applications access to their own status, the card status, the services of the associated security domain, and the services of the card manager (CVM). The interface that must be supported by the service provider is also defined in the API. For some functions, the Global Platform environment checks whether the application has the necessary privileges. These application privileges are stored in the card registry. In addition to the previously described privileges, there are privileges and API functions for blocking the card (card lock privilege), terminating the card (card terminate privilege), and altering the historical bytes in the ATR (default applet privilege).

### 13.13.4 Global Platform commands

Several special commands necessary for the Global Platform functions are defined in the GP specification. Table 13.17 on the next page lists the commands defined in the Global Platform specification and explains what they mean.

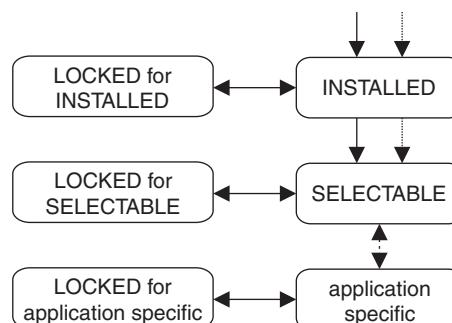
When application data is loaded, the data to be transferred is TLV-coded and consists of the actual load data of the application (the load file) preceded by an optional data authentication pattern (DAP) block for cryptographic protection of the data and the loading process.

The first stage of the application's life cycle, which is also defined by the GP specification, starts after it is loaded into the smart card. This is illustrated by the state diagram in Figure 13.28 on the following page. The first stage is called 'installed', which means that the application has been stored in the memory allocated to it and properly linked to the operating system, so that it can be run. However, the application cannot yet be selected from outside the smart card in this state. This is possible in the next state, which is called 'selectable'. All further states are application-specific and are not monitored by the operating system. However, the card manager can block applications by setting them to the 'locked' state. This state can only be reset by the system. If an application is deleted but it cannot be physically deleted, for example because it is stored in ROM, it enters the 'logically deleted' state.

Security domains have the same life cycle, but there is only one application-specific state, which is attained when the security domain has received all the data and keys needed for its services, which means that it has been personalized.

**Table 13.17** Global Platform commands. The card content manager must have the delegated management privilege in order to be the processing entity

Command	Processing entity	Function
SELECT	Dispatcher	Subset of the SELECT command as defined in ISO/IEC 7816
MANAGE CHANNEL	Dispatcher	Subset of the MANAGE CHANNEL command as defined in ISO/IEC 7816
INITIALIZE UPDATE	Security domain	Initialize a secure communication channel
INSTALL FOR PERSONALIZATION	Security domain	Personalize an application via the security domain
EXTERNAL AUTHENTICATE	Security domain	Command for authentication of the CAD. This command completes the establishment of a secure communication channel
PUT KEY	Security domain	Stores or alters a key in the security domain
DELETE KEY	Security domain	Deletes a key in the security domain
GET DATA	Security domain	Read TLV-coded data from the security domain
STORE DATA	Security domain	Store TLV-coded data in the security domain
SET STATUS	Card content manager	Set the card status or application status
GET STATUS	Card content manager	Read the AID, privileges, and status of the issuer security domain
INSTALL FOR LOAD	Card content manager	Start loading an application program
LOAD	Card content manager	Load a new application program block in the card
INSTALL FOR INSTALL	Card content manager	Install an application
INSTALL FOR EXTRADITION	Card content manager	Alter the linking of an application to a security domain.
DELETE	Card Content Manager	Delete an application from the card



**Figure 13.28** The possible states of an application in a multiapplication smart card that complies with the Global Platform specification. The entities responsible for each of the state transitions are indicated. The solid lines indicate the state transitions controlled by the issuer security domain, while the dotted lines indicate the state transitions that can be initiated by an application. The state transitions shown as dashed lines are performed by the application

Terminal (IFD)		Smart card (ICC)
SELECT card manager with AID	→	Return code := selection result
IF (return code = OK)	←	<i>Response</i> [return code]
THEN card manager successfully selected		
ELSE abort		
Authentication of the outside world with respect	→	...
to the issuer security domain (e.g. using		
EXTERNAL AUTHENTICATE)		
IF (authentication = OK)	←	...
THEN continue sequence		
ELSE abort		
INSTALL with parameter 'load an application'	→	...
REPEAT {		
LOAD using the data in the load file		...
IF (return code = OK)	←	<i>Response</i> [return code]
THEN loading successful		
ELSE abort		
}		
UNTIL (load file fully transferred)		
INSTALL with parameter 'install an application'	→	...
INSTALL with parameter 'make application selectable'	→	...

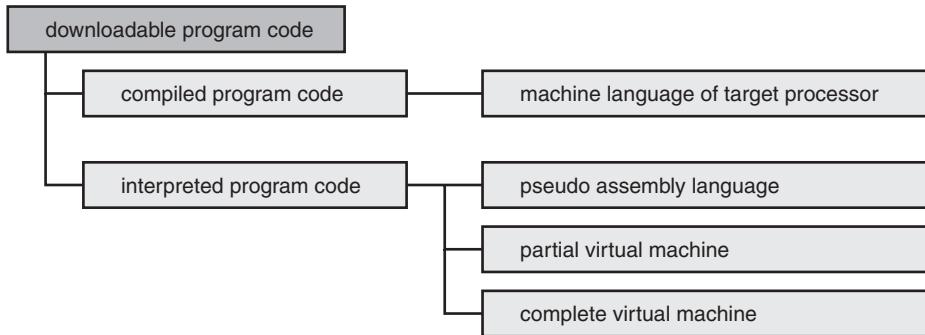
**Sequence Diagram 13.1** Basic command sequence for loading a new application in a smart card with Global Platform

## 13.14 DOWNLOADABLE PROGRAM CODE

In the first German edition of this book published in 1995, the section titled 'Downloadable Program Code' occupied approximately one and a half pages. The amount of text devoted to this subject has increased by a factor of 20 in the present edition. This alone indicates how important this subject has become. There is no risk of exaggeration in saying that a full paradigm change with regard to downloadable program code in smart cards occurred within one year (1997). Downloadable program code in smart cards is now regarded as the rule rather than the exception.

The reasons for the sudden increase in the importance of downloading executable program code cannot be clearly ascertained, even in retrospect. One trigger may have been the floating-point error in the FDIV instruction of the then widely used Pentium processor, which became well known in 1994. This was a true hardware error that could not be corrected by downloading new software. However, software patches to circumvent the problem were available for many applications.

It is likely that this error is the reason why some large system operators, shortly after it became generally known, suddenly made plans to allow executable code to be downloaded to smart cards. One of the largest applications that can accept executable program code is the German Eurocheque smart card. However, this capability is not presently used and is effectively only a 'sheet anchor' to be used if serious program errors are discovered. In the GSM system, there are also smart card operating systems for SIMs that allow program code for special applications to be downloaded via the air interface.



**Figure 13.29** Various methods for downloading and running executable program code in a smart card operating system

However, in contrast to all other computer operating systems, with smart cards it is not common practice to load programs into the cards after they have been issued and then run these programs as desired, despite the fact that this (along with storing data) is actually a primary function of every operating system. There are naturally good reasons why this particular capability has been largely absent in smart cards up to now.

From a technical and functional perspective, executable program code (stored in EFs, for example) does not present any problems at all. Modern operating systems can manage files containing executable code, and they allow executable code to be downloaded after the card has been personalized. This makes it possible for an application provider to execute program code in the smart card that is not known to the producer of the operating system. An application provider could thus load a private encryption algorithm into the card and use it there. This would allow the knowledge of the security features of the system to be divided between several parties, which is one of the basic requirements for secure systems.

Another good reason for the mechanism of downloadable program code is that it creates opportunities for correcting program errors in fully personalized cards (bug fix). Known errors in the operating system can be corrected or at least rendered less severe by using downloaded code.

As shown in Figure 13.29, there are two basic ways to execute program code in a smart card. The first and technically simplest way is to load native code (code compiled into the machine language of the target processor) into files in the smart card. This program code must of course be relocatable, since the memory addresses are not known outside the card. In addition to its technical simplicity, this solution has the advantage that the program code can be processed at the full working speed of the processor, which makes it especially attractive for downloadable algorithms. In addition, there is no need for extra program code in the smart card for an interpreter. The main problem with this approach is that the downloaded program can also access the memory regions of other applications if the microcontroller does not have a memory management unit (MMU).

The second way to execute downloadable program code in smart cards is to interpret the code. In this case, the interpreter can check the memory regions that are addressed when the program is running. However, the interpretation must run quickly because program code that runs slowly is of little use. The implementation of the interpreter should also occupy as little memory as possible, since only a very limited amount of memory is available. Presently, the

best known versions of this approach are the Java Card specification [JFC], the MEL (Multos Executable Language) C interpreter from Multos [Maosco], and Windows for Smart Cards (formerly discontinued). A Basic interpreter for smart cards has also been available for several years [Zeitcontrol]. Interpreters make a specific region of protected memory available to an application program, which means they are not suitable for correcting errors in smart card operating systems because they do not have access to the regions where the operating system routines and data are stored.

The basic problem with interpreters is their slowness, which is an inherent property of interpreted code. There are several approaches that can be used to minimize this drawback and keep the program code of the interpreter as small as possible. This simplest approach is to interpret pseudocode, which ideally is as close as possible to the machine instructions of the target hardware. With machine-level pseudocode, the processing speed of the interpreter is relatively high and machine-independent program code can be used. Memory accesses during interpretation can be monitored, but this is not mandatory. A slower solution that is somewhat more complicated in terms of programming logistics is to split the interpreter into an offcard part (the ‘offcard virtual machine’) and an oncard part (the ‘oncard virtual machine’). This approach is taken by many current Java Card implementations. Its main advantages are reliable memory protection and complete hardware independence. However, dividing of the interpreter into oncard and offcard parts has drawbacks. It makes cryptographic protection mandatory for transferring programs between the two parts of the interpreter, since otherwise the oncard part of the interpreter could be deliberately caused to misbehave by using manipulated program code.

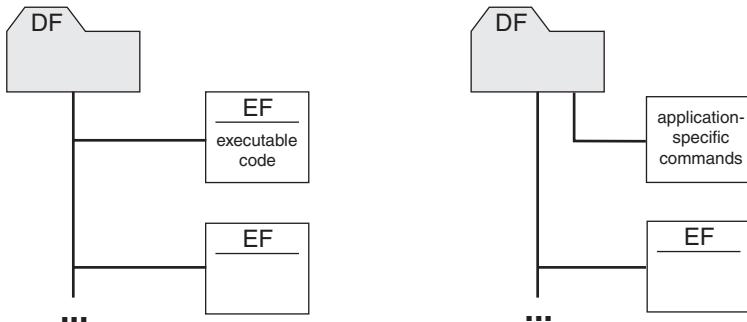
The optimum technical solution is to have a complete interpreter in the smart card. This makes it possible to load any desired program into the card and run it without any risk to other applications in the card. However, the code size of a full interpreter is so large that it will certainly take several years and several generations of smart card microcontrollers before this solution becomes widely established in the smart card world.

## 13.15 EXECUTABLE NATIVE CODE

Microcontrollers currently used for smart cards still have processors that lack any form of memory protection mechanism or monitoring capability. If the program counter enters a region containing ‘foreign’ executable code, this code assumes control of all of the memory and all functionality. At this point, it is no longer possible to restrict the functions of the executable program. Every addressable memory location can be read, bypassing any memory manager or handler that may be present, and memory locations in RAM, EEPROM or flash memory can also be written. The entire contents of the memory could thus easily be sent to a terminal via the card interface.

This is precisely the weak point of downloadable and executable programs. If everyone were allowed to download programs, or if programs could be downloaded by circumventing protective mechanisms, the security of any secret keys or other confidential information in the entire memory region could no longer be assured. This would be an ideal form of attack on a smart card. The card would still behave in the same way as an unmanipulated card with respect to the outside world, but special commands could be used to read out its entire memory or write data to portions of the memory.

This problem cannot be eliminated by simply restricting downloading to a few application providers, which would certainly be feasible with mutual authentication before the code is



**Figure 13.30** The two options for adding executable program code to a conventional smart card operating system: as an executable file (left) or as application-specific commands (ASC) (right)

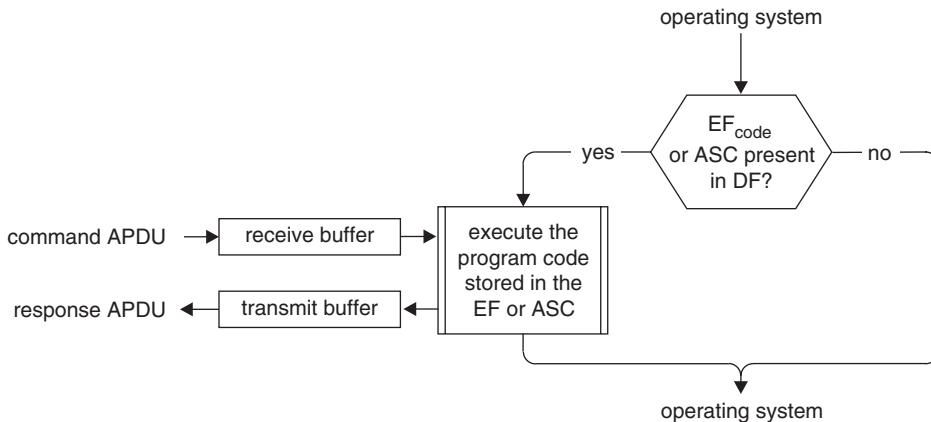
downloaded. This is because there is nothing to prevent an application provider from exceeding the boundaries of the DFs allocated to his application in order to access secret information belonging to other applications present in the card. This would compromise the security of the system.

There is yet another compelling argument against downloadable third-party program code. The producer of the downloadable file must know all the entry points (jump addresses) and calling parameters of the operating system routines in order to use important operating system functions. However, in the interest of security some operating system producers reveal as little information as possible about the internal processes and addresses of their program code. In addition, it would be necessary to ensure that the downloaded code does exactly what it is supposed to do without any errors and does not harbor a Trojan horse. This can only be done by an independent entity.

The most elegant and probably the most promising solution is to use a hardware memory management unit (MMU) in the smart card alongside the actual processor. The MMU hardware monitors the program code while it is executing to ensure that it remains within its assigned boundaries. Only with this approach is it possible to allow program code from application operators to be downloaded without prior examination by the card issuer, while still preserving the security of the card. Each such application is assigned a physically contiguous memory region for its DF. The MMU monitors the assigned memory boundaries when the program downloaded into the DF is running. If it exceeds these boundaries, program execution can be halted immediately by an interrupt, and the application is blocked pending further action.<sup>12</sup>

There are two implementation options for downloading native code into smart cards, as indicated in Figure 13.30. The first is to place the program code in an EF with ‘executable’ structure. After this file has been selected, its contents can be executed by issuing the EXECUTE command. Depending on the application, prior authentication may be required. The parameters for running the program are supplied to the smart card in the EXECUTE command, and the command response generated by the program in the EF is sent back to the terminal. Figure 13.31 illustrates the process for executing downloaded native code using this mechanism.

<sup>12</sup> See also Section 5.4, ‘Supplementary Hardware’, on page 93



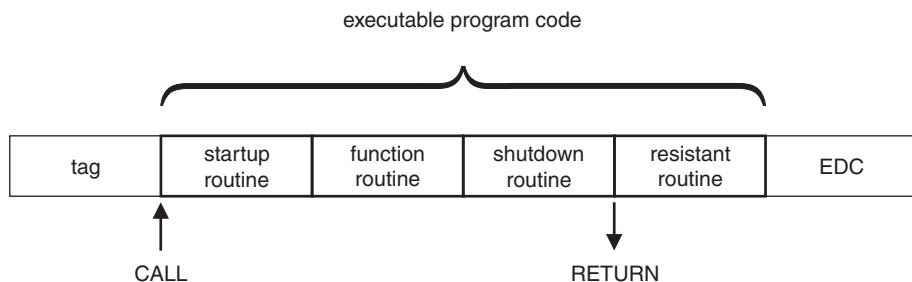
**Figure 13.31** Basic calling procedure for executable program code stored in an EF or programs based on ASCs with a conventional multiapplication smart card operating system

The second approach takes a somewhat different form, since it is based on the principles of object-oriented design. This option is described in the EN 726-3 standard (and elsewhere) as ‘application-specific commands’ (ASC). This standard stipulates that the entire application with all of its files and application-specific commands is contained in a single DF. Program code can be downloaded to a memory region in this DF that is managed by the operating system. This is done using a special command that transfers all the necessary data to the smart card. If the DF in question is selected and a command is sent to the card, the operating system checks whether the command is one of the downloaded commands. If it is, the operating system immediately begins execution of the program code in the DF. If a different DF is selected, the downloaded commands effectively do not exist in the current context.

There are several large smart card applications whose specific operating systems allow executable code to be downloaded after the card has been personalized. However, the specifications for this capability are almost always confidential, and in some cases even the fact that this capability exists is confidential. Consequently, we can only describe the general principles of this capability here, independent of any actual operating system, and present a possible implementation in more detail.

Certain basic requirements must be satisfied before downloadable program code can be regarded as eligible for use in a smart card. It may sound obvious, but the most important requirement is that the processor type must be known (for example, 8051 or ARM). Particularly in a heterogeneous environment with many different types of smart card microcontrollers, satisfying this requirement can easily involve a certain amount of effort. Along with this comes the requirement that the smart card operating system and its application programming interface (API) must be known, including all entry points and the parameters supplied to and returned from its routines.

The downloadable program code, which is always native code (machine code of the target processor), must be programmed such that it is relocatable, or the smart card must relocate the code on the fly while it is downloading. The relocatability requirement, which means that the code can be shifted in memory, arises from the fact that the memory addresses where the code will be stored are known only to the smart card operating system and not to the outside



**Figure 13.32** A possible structure for native code that can be downloaded to and run from an EF.

world. Program code is usually made relocatable as part of software development. In specific terms, among other things relocatability means that only jumps relative to the address of the jump instruction are allowed; jumps to absolute physical addresses are not allowed.

If the program code satisfies all of these requirements, it can in principle be loaded into the memory of a smart card and run there. The program code can of course be structured as desired. Figure 13.32 shows a possible structure, but the actual structure can be completely different, depending on the operating system. The first data element in this example is a unique label that identifies the data element to the smart card operating system as program code. Such a label is commonly called a magic number. For example, with Java Class files the label consists of the four bytes ‘CAFEBABE’.

The program code starts after the label. In this example, it is divided into four sections. The first section contains all the necessary initializations, data saving and the like. Following this start-up routine comes the actual function routine, which contains the program code for the desired task. This is followed by the shutdown routine, which is the counterpart to the start-up routine. The shutdown routine ensures that the program is correctly terminated, and if necessary it restores any saved data and adjusts the stack.

The fourth section of the program shown in the figure is optional. It can include program code for resistant integration into the smart card software. Bug fixes for the smart card operating system are typically located here. In this case, the three prior routines would change pointers or handles in order to permanently integrate the routines in this section into the software of the operating system. All of this is very similar to the well-known TSR (terminate and stay resident) routines of the DOS era. When these routines are called the first time, they anchor themselves in the operating system and remain there until the next reset. In the smart card situation, these resistant routines would be installed permanently after being called, rather than only for the duration of a single session.

Here we assume that the downloaded program is called using a Call instruction and that it returns control to the calling program with a Return instruction. In principle, a direct jump to the first machine instruction (using a Jump instruction) is also possible, but this has the disadvantage that the called routine does not know which routine called it.

For insurance against unintentional changes, the entire data block should be protected by an error detection code (EDC). Alternatively, a digital signature could be used to provide additional protection. In this case the smart card would hold the public key and the producer of the program code would hold the associated private key. This would provide binding assurance that the program code to be run in the smart card is authentic.

---

Terminal (IFD)	Smart card (ICC)
Select an EF with execute structure	→ ...
Mutual authentication of the smart card and the background system	→ ...
Enable secure messaging	
Send $n$ UPDATE BINARY commands with executable code in the data segment, protected by secure messaging	

---

**Sequence Diagram 13.2** A possible basic procedure for loading executable program code in an existing EF with ‘execute’ structure. The access conditions for UPDATE BINARY stipulate mutual authentication of the smart card and the terminal and data transmission using secure messaging

---

Terminal (IFD)	Smart card (ICC)
SELECT <i>Command</i> [EF with executable program code]	→ Select the requested file
	IF (file found) THEN return code = OK ELSE return code = file not found
EXECUTE <i>Command</i> [data]	← Response [return code] Check the label (magic number) of the program Check the EDC of the program Run the first machine instruction using CALL Provide response data
IF (return code = OK) THEN file executed ELSE error during file execution	← Response [data    return code]

---

**Sequence Diagram 13.3** A possible procedure for running executable program code, which in this example is stored in an ‘execute’ EF

The downloaded program code can be stored in an EF or in a program memory region in a DF that is not visible to the outside world. The first option is described in some detail below, since it is encountered significantly more often in actual practice.

EFs with ‘transparent’ structure are ideal for storing program code because they can be effectively written in several sections using the UPDATE BINARY command with offsets. Also, their maximum length of more than 65 KB is more than adequate, even for extensive programs. These EFs can have the ‘executable’ attribute so that the program code stored in them can be invoked directly by an EXECUTE command.

However, some operating systems also support a file structure called ‘execute’, which is derived from the transparent structure. This is not especially important to the outside world, especially since both types can usually be accessed by the UPDATE BINARY and EXECUTE commands. The EF can be selected by its FID or an SFI. The read access condition is always set to ‘never’. Writing data is normally allowed after prior authentication, using secure messaging.

Sequence Diagram 13.2 shows the basic elements of a procedure for loading program code into an EF in a smart card in a secure manner. If a suitable EF is not already available, it must first be created. Sequence Diagram 13.3 shows the basic process of EF selection followed by

an EXECUTE command that initiates execution of the code. Data can optionally be transferred in the body of the command, and data can be returned to the terminal in the response in a similar manner if necessary. Naturally, the called program must retrieve any data that is sent to it by reading it from the receive buffer, generate its response, and place its response in the transmit buffer.

Due to the stringent requirements for unambiguous identification of the microcontroller, the operating system and the internal software interfaces, as well as system management requirements, program downloading is usually limited to online downloads from a background system. The databases located in the background system either hold all the necessary data keyed to the unique chip number, or they receive this information online via a direct end-to-end link with the smart card. Using this information, a program with the desired functionality is selected from those available in the system and transferred to the smart card using the specified security mechanisms. The secret keys for this are normally managed and used in the background system exclusively within a security module. The usual procedure for this process is shown in Sequence Diagram 13.2.

The method for loading native code into a smart card described above has several attractive practical advantages. The method is simple and robust, and it can be implemented in a smart card operating system at relatively low cost. The program code does not have to be interpreted, but instead can be executed directly by the processor. This yields high processing speed, which makes this method suitable for downloading complex algorithms such as AES, IDEA and the like.

Due to their low processing speeds, interpreter-based systems cannot achieve this performance now or in the foreseeable future. In the absence of hardware-based memory management (MMU) to restrict free memory access, this method provides an excellent way to correct errors in the smart card software after the card has been issued. It provides a sort of back door for use in emergencies, which can only be achieved with this form of software downloading because other technologies, such as Java for smart cards, implement strict and unconditional memory partitioning. If a MMU is present, it is still possible to use administrator mode to temporarily disable memory supervision if necessary.

This brings us to the drawbacks. Downloading executable native code presupposes a high level of knowledge of the hardware and/or the operating system of the smart card. It may be necessary to have a separate program available for each type of smart card used in the system, even if all of these programs have the same functionality. The second major drawback of this method is that for security reasons the program must be developed by the card issuer or on order of the card issuer. Loading unknown or third-party programs into the smart card must be strictly prohibited, since the downloaded program assumes control of the microcontroller when it is started and cannot be governed in any manner. Such a program could for example read out the secret keys of other applications in the card and send them to the terminal via the I/O interface.

Evaluation of the program code by the card issuer provides only weak protection against attacks of this sort. Better security can be achieved by using hardware-based memory management, which makes only certain regions in RAM and nonvolatile memory available to the downloaded program and immediately terminates the program if it tries to exceed the boundaries of these regions. This makes it possible to fully partition the applications in the smart card. Due to the lack of suitable MMUs, the only available expedient at present is to carefully review the downloadable program code.

## 13.16 OPEN PLATFORMS

With the widespread use of Java Card, Multos and Windows for Smart Cards, the term ‘open platform’ has come into general use. It refers to smart card operating systems that allow third parties to load applications and programs into smart cards without any involvement of the operating system producer. In most cases, the specifications of open platforms are public and are generated by a consortium of companies, such as Java Card Forum. Most open smart card operating systems are available from several producers, who provide systems with similar or mutually compatible functions.

The opposite of an open platform is a ‘proprietary’ platform. The term ‘proprietary’ is also often applied to company-specific solution in a deprecatory sense. In many cases, the specifications for such platforms are not fully published or are the property of a single company.

However, both terms – ‘open’ and ‘proprietary’ – are used in ways that are by no means unambiguous or objective, and which are often strongly driven by marketing interests. Objectively, many so-called ‘open’ smart card operating systems are rather proprietary and dependent on a particular company. Truly open platforms such as Linux, with free access to the source code, no licensing restrictions and independence from specific companies or organizations, are presently not available in the world of smart card operating systems.

### 13.16.1 ISO/IEC 7816 compatible platforms

ISO/IEC 7816 essentially describes smart card operating systems for file-based applications. Virtually all major card producers have at least one product line in their portfolio with functionality that complies with the essential aspects of this family of standards. However, the focus here is always on file-based applications. This means that third-party program code cannot be run on smart cards based on this platform.

### 13.16.2 Java Card

In 1996, Europay presented an article that described and largely specified an open terminal architecture (OTA) for terminals, based on a Forth interpreter. The objective was to establish a uniform software architecture for terminals in order to create a basis for hardware-independent terminal programming. With this architecture, a specific application such as paying with a credit card would only have to be programmed once, and the software could be without any modifications on all terminals made by various manufacturers. Although the proposed model was never fully implemented, it stimulated extensive discussion in the smart card world.

Consequently, no one was especially surprised when it became known in the fall of 1996 that Schlumberger was developing a smart card that could run platform-independent programs written in the Java language. The idea of combining an interpreter with a memory-poor microcontroller was already well known from the OTA proposal. The published specification (Java Card 1.0) described an application programming interface (API) for integrating Java into an ISO/IEC 7816-4 operating system so that Java could access the usual smart card file system with its MF, DFs and EFs.

Many producers of smart card operating systems were initially skeptical of the idea of using a language such as Java, which normally requires well over a megabyte of memory, in smart card applications. However, nearly all major smart card manufacturers were represented at the first meeting with Sun, the company that developed and promoted Java, in the spring of 1997.

This was the first meeting of what has since become known as Java Card Forum (JCF), which acts as the international standardization body for Java in smart cards. The task of the technical group of Java Card Forum is to define a subset of Java for smart cards, to specify the framework of the Java interpreter (the Java virtual machine or JVM), and to define a general-purpose API as well as application-specific APIs for application areas such as telecommunication and payment systems. The task of the marketing group of JCF is to promote Java technology for smart cards.

The current versions of the specifications are the Java Card Virtual Machine (JCVM) Specification, the Java Card Runtime Environment (JCRE) Specification, and the Java Card Application Programming Interface. They are available free of charge from the Java Card Forum web server [JCF]. Approximately 3.5 billion Java cards were issued up to December 2007, including around 1.2 billion in 2007 alone.

### ***13.16.2.1 The Java programming language***

In 1990, a development group at Sun lead by James Gosling started developing a new programming language. The objective was to create a hardware-independent, modern and secure language that could be used for microcontrollers in consumer products such as toasters and espresso machines. A large variety of microcontroller types with various architectures are used in such products. The combination of this diversity and frequent hardware changes makes it rather difficult for software developers to write portable program code. Remarkably enough, smart cards exactly match the characteristics of the original target application area.

The language was initially called Oak after the oak tree in front of James Gosling's office, but the name was changed to Java in 1995 and the objectives were redefined.<sup>13</sup> In the summer of 1995, Sun began to intensively promote Java as the hardware-independent language for the heterogeneous World Wide Web on the Internet. Sun's slogan 'Write Once – Run Anywhere', which was often quoted at that time, probably provides the clearest indication of the intended degree of hardware independence.

The beginning of the widespread use of Java coincides with the beginning of the enormous growth of the World Wide Web (WWW).<sup>14</sup> For a variety of nontechnical reasons arising primarily from business policies and worldviews, the new language was welcomed enthusiastically by researchers, universities and software companies throughout the world [Franz 98]. As a result, Java very quickly became the *de facto* standard for Internet applications. Naturally, the properties of this new language encouraged this development.

The Java programming language is a fully object-oriented and strongly typed language. Programmers find it easy to learn because it has much in common with C and C++. Java is also a robust language, which means it does not permit the tricks and popular but dubious techniques possible with C and C++ (among other languages). For example, no pointers are

<sup>13</sup> 'Java' is American slang for coffee, and in this context it has no connection with the Indonesian island or the French biscuits with the same name

<sup>14</sup> In 1993, there were only three WWW servers in the entire world

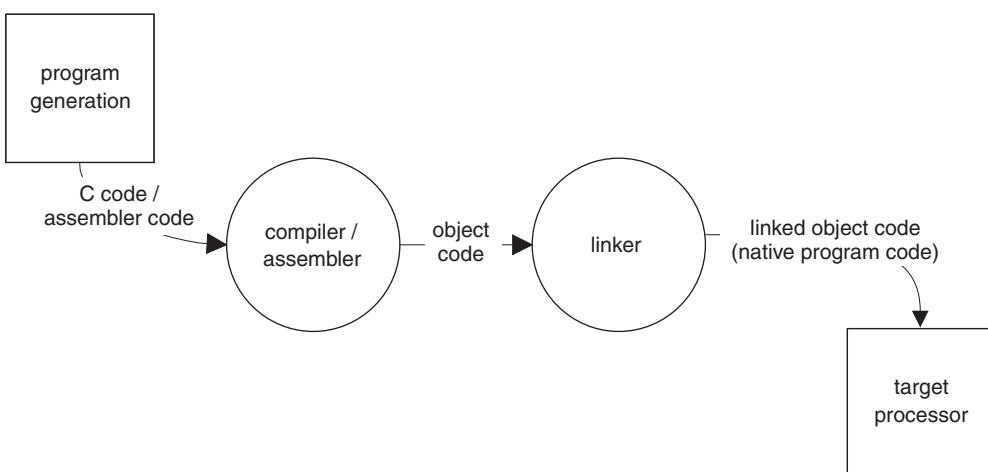
used in Java, field boundaries are monitored at run time, and there is strict type checking. In addition, memory management is handled by Java and an associated garbage collector, so memory leaks (a much-feared phenomenon with C and C++) are impossible by design. Java is also a secure programming language, which means that when a program is run the functions that it wants to perform are monitored while it is running, so the runtime environment can stop the program if necessary. This is one of several possible reasons for generating an exception call. The main advantage of exceptions is that they must be handled and the place where they are handled can be freely chosen. If an exception occurs, the program's own functions and all callable functions are searched for a routine (catch block) that can handle this sort of exception. The appropriate response can be specified in the catch block. If a suitable catch block for the exception is not found, the Java virtual machine terminates processing of the program and issues a suitable error message.

Most of these properties are only possible because Java is an interpreted programming language that is not executed directly by the processor. Java also has other properties, such as multithreading capability and support for distributed processing.

Sun's intention was to have Java standardized in the form of an ISO standard. However, for a number of reasons this did not happen. One of the reasons is doubtless that ISO's five-year review and revision cycle is too long for a new programming language such as Java because it delays and hampers the implementation of changes made necessary by practical experience. Another conceivable reason could be the requirement for a licensing agreement with Sun, with significant associated costs, before Java technology can be used in products (such as smart cards). This is contrary to the usual conventions for ISO standards.

#### 13.16.2.2 The properties of Java

As illustrated in Figure 13.33, programs written in Java are translated into Java bytecode by a compiler. Java bytecode is essentially a sort of processor-independent object code. Bytecode



**Figure 13.33** Top-level data flow diagram of the typical process used to convert source code in C or assembly language into executable machine code for a target processor

**Table 13.18** Basic structure of a class file

Data elements of a class file
Label ( <i>magic number</i> )
Version number
Constants pool
Methods
Attributes of classes, fields and methods

can be regarded as program code consisting of machine instructions for a virtual Java processor. This processor does not actually exist, but is instead simulated by the actual target processor. This simulation takes place in the Java virtual machine (JVM or VM), which is the actual interpreter. Seen from a different perspective, the JVM is a simulation of the Java processor on an arbitrary target system. Of course, the target processor works with native code. The main advantage of this arrangement is that only the JVM, which is programmed in native code, has to be ported to a particular target processor. Once this is done, Java bytecode can run on the new processor.

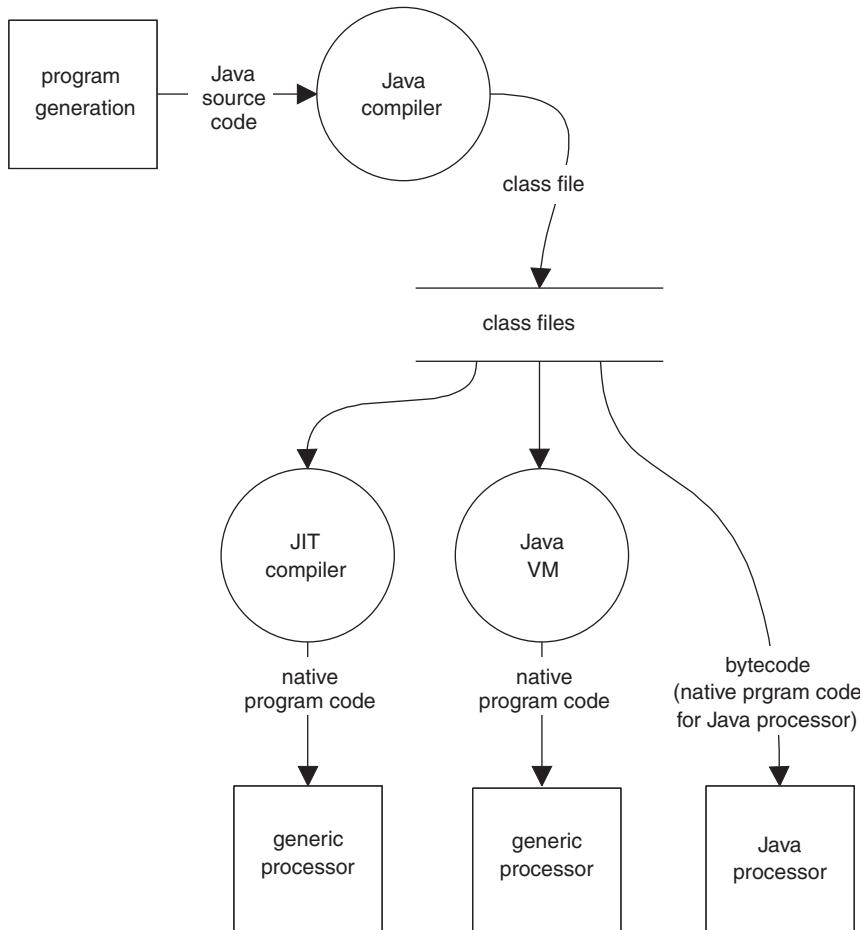
The runtime task of the VM involves more than just blindly interpreting bytecode and includes activities such as type checking and monitoring access to objects, for which reason the VM is also called the sandbox. This name graphically illustrates the fact that a Java program is only allowed to act within its own environment (sandbox) and is not allowed to leave this environment, as the VM will otherwise put a stop to its activities.

A compiled Java program, which means one that has been translated into bytecode and provided with certain supplementary information, is stored in a class file with the structure described in Table 13.18. The program is executed by the Java virtual machine after it has been loaded. An application consists of one or more class files. Figure 13.34 shows three options for the processes of converting Java source code into executable machine code.

The hardware independence of Java naturally has its price, which primarily consists of its low execution speed relative to other common programming languages. Although this problem has been resolved in a satisfactory manner in the PC domain, it remains unresolved in the embedded systems environment. One step that has already been taken is to use a just-in-time compiler (JIT compiler), which analyzes the program flow and translates portions of the Java bytecode into the processor's machine language. Although this causes slower execution the first time the program is run, subsequent runs are significantly faster.

### 13.16.2.3 Java virtual machine (JVM)

The Java virtual machine is the essential element of Java technology. It simulates a Java processor, and it can be implemented in software on any sufficiently powerful processor. If it is desired to run Java bytecode on a new type of processor, the Java virtual machine must be ported to this processor. It is usually written in the C programming language, which means that in some cases the actual porting of the virtual machine may not require anything more than a few minor changes and recompilation of the source code. The size of a Java virtual machine on a PC ranges from 100 to 200 KB.



**Figure 13.34** Top-level data flow diagram of possible processes for converting Java source code into executable machine code for a target processor. Although the path on the left is provided by some compiler producers, it does not correspond to the original Java philosophy because it does not maintain hardware independence

A Java virtual machine has all the elements of a real processor. It has its own instruction set in the form of the bytecode, and it has a program counter and a stack. The data to be processed is supplied to the virtual machine as a class file, which contains the fixed constants, the bytecode to be executed in the form of methods, and various supplementary data.

Java bytecode takes up very little space and is almost as compact as machine code. The only thing that degrades the net memory space requirement compared with native machine code is the obligatory virtual machine. The difference naturally depends on the size of the program code relative to the size of the virtual machine code.

Bytecode is essentially very similar to the machine instructions of a real processor. For example, it has stack manipulation instructions, logical and arithmetic instructions, and instructions that access the fields of objects and local variables. The Java virtual machine and bytecode are extensively described in a book by Tim Lindholm and Frank Yellin [Lindholm 97].

**Table 13.19** Comparison of the functional scope of Java Card and full Java

Functionality	Java Card VM	Java VM
Operators	all	all
Sequence control functions	yes	yes
Exception handling	yes	yes
Data types: boolean, byte, short	yes	yes
Data type: int	optional	yes
Data types: long, float, double, char	no	yes
Fields	one-dimensional	multidimensional
Object fields	one-dimensional	multidimensional
Cloning of classes	no	yes
Cloning of objects	no	yes
Dynamic object creation	yes	yes
Static methods	yes	yes
Virtual methods	yes	yes
Dynamic downloading of classes	no	yes
Load unit	Package	Class
Interfaces	yes	yes
Dynamic memory management (garbage collection)	optional	yes
Threads	no	yes

#### 13.16.2.4 Java Card virtual machine (JCVM)

Due to the significantly restricted system resources of smart card microcontrollers, certain restrictions are placed on the Java Card VM compared with the original Java VM for PCs. There are fewer data types available in smart cards, and the bytecode is reduced from 149 instructions to 76. Support for class files is also restricted.

The functional restrictions of Java for smart cards relative to regular Java are listed in Table 13.19, and the limitations of Java for smart cards are described in Table 13.20. The data types supported by Java Card and their characteristics are listed in Table 13.21.

It was also necessary to split the Java Card VM into an oncard part and an offcard part. Static checks can easily be performed outside the smart card in the offcard VM without reducing performance or security. The link between the two parts of the VM is formed by data in CAP format. For full security, this data must be cryptographically protected, ideally using digital signatures, so it cannot be manipulated during transmission. It would otherwise present an attacker with a promising starting point, since the security mechanisms of the offcard VM could be circumvented using manipulated data.

After bytecode verification is finished, the loader takes the checked data and sends it in CAP format to the actual interpreter in the smart card. For security reasons, the data should be provided with a digital signature so it cannot be manipulated on its way from the loader to the oncard part of the Java VM. The actual loading process is independent of the Java Card specification. Here the Global Platform specification<sup>15</sup> has become established as an industry standard.

<sup>15</sup> See Section 13.13, ‘Application Management with Global Platform’, on page 485

**Table 13.20** Summary of the limitations of Java for smart cards. These broadly defined limitations currently do not impose any restrictions on the development of software for smart cards

Language element	Limitation
Class	An instance of a class may have at most 255 fields. A class may have at most 256 static methods. A class may have at most 256 static fields. A class may have at most 15 interfaces
Package ( <i>package</i> )	A package may have at most 255 public classes and interfaces
Method	A method may have at most 255 local variables. A method may have at most 32 767 bytecodes
Array	Arrays may have at most 32 767 fields
Switch instruction	A switch instruction may support at most 65 536 branches. If the ‘int’ data type is supported, the maximum number of branches in the switch instruction depends on the value range of the selected data type (char, byte, short or int), as with Java for PCs

**Table 13.21** Data types supported in Java Card with their memory requirements and value ranges. The ‘int’ data type is optional

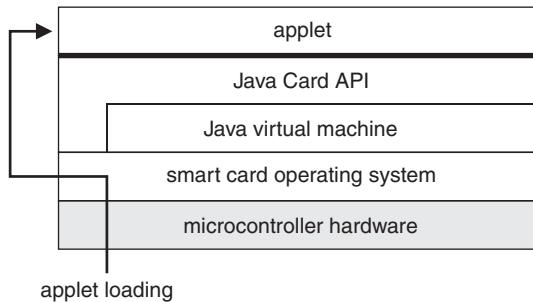
Data type	Size of static variable (bytes)	Size of instance variable (bytes)	Size of local variables (bytes)	Value range
boolean	1	2	2	true, false
byte	1	2	2	-128 ... 127
short	2	2	2	-32 768 ... 32 767
object reference	2	2	2	-32 768 ... 32 767
int	4	4	4	-2147 483 648 ... 2147 483 647

After being loaded, the executable bytecode is located in the memory of the smart card along with various supplementary data, where it can be executed by the oncard part of the Java VM. The interpreter reads the bytecodes one at time with their associated arguments and converts them into the native machine instructions of the target processor. The security manager operates in parallel with the bytecode interpreter. Among other things, it is responsible for checking compliance with field, stack, and heap boundaries. If it detects a violation of the defined security rules, it is authorized to immediately initiate an exception and stop the processing of the bytecode that caused the problem.

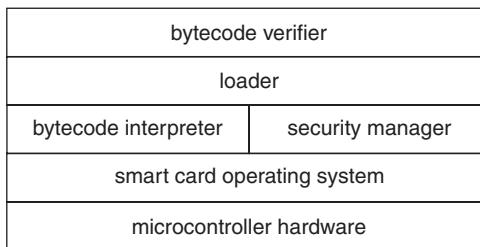
The basic process for loading a Java applet in a smart card is depicted in Figure 13.35, and the general structure of the Java Card virtual machine is shown in Figure 13.36.

### 13.16.2.5 Memory sizes in Java cards

The oncard Java VM for smart cards has a program code size of around 40 KB of 8051 machine code when written in C, with the pure interpreter code accounting for approximately 12 KB of this. It also needs approximately 400 bytes of RAM. The API with the javacard.framework and



**Figure 13.35** Basic process for loading an applet into a smart card operating system with Java



**Figure 13.36** Basic components of the Java Card virtual machine

`javacardx.framework` classes occupies 3 to 4 KB of memory; most of the API is programmed in Java. In addition, the Java VM requires at least a rudimentary operating system with data transmission protocols, cryptographic algorithms, and many hardware-level functions. The code for this occupies 6 to 8 KB if it is programmed in assembly language [Baentsch 99]. With regard to these memory sizes, it must be noted that they are highly dependent on the supported functions and other complex considerations. Consequently, the values stated above can vary considerably from one implementation to the next. As a general rule, the memory space requirements depend on the number of API functions that are provided.

As a stack-oriented language, Java naturally needs a stack and a heap. They are created and managed separately for each applet in the smart card. The stack is primarily used for passing data when calling methods, while the heap serves as a storage area for objects. Typical sizes are approximately 700 bytes in RAM for the stack and approximately 5 to 8 KB in EEPROM for the heap. However, relatively small applets can generally manage with 50 to 60 bytes of stack space and a few hundred bytes of heap space.

The basic functions of the Java bytecode interpreter are described in pseudocode form in Table 13.22. The correctness of this relatively small program is extraordinarily important, since an error or security gap in the Java VM could undermine the whole security concept of Java for smart cards. The design and implementation must therefore be essentially error-free. Common Criteria evaluations are normally used to verify this property.<sup>16</sup> The small size of the Java VM program code considerably simplifies the evaluation process, especially because it allows the full functionality of the VM to be described formally.

<sup>16</sup> See also Section 15.5, ‘Evaluation of Hardware and Software’, on page 659

**Table 13.22** Basic functions of the main program loop of the Java bytecode interpreter

DO (	Interpreter main loop
fetch and save program counter	Fetch and save the program counter for later comparison.
fetch opcode	
fetch operands	
execute machine instruction	Execute a virtual Java processor machine instruction, which consists of an opcode and its operands.
IF (machine code did not - alter program counter)	If the Java machine instruction just executed did not alter the program counter (which means it was not a GOTO bytecode, for example), set the program counter to the next opcode.
THEN (increment - program counter)	
) WHILE (opcodes available)	Repeat until all opcodes have been processed.

### 13.16.2.6 Performance in Java cards

Given the relatively small memory sizes of smart cards, implementation of a JIT compiler in a smart card will not be possible in the near future due to the considerable software complexity (and thus size) of such compilers. Unlike the situation in the PC world, direct compilation of Java programs into the machine language of the target processor would not be worthwhile in the heterogeneous world of smart cards.

A conceivable approach would be to integrate a special Java processor in the semiconductor device alongside the normal processor. This would provide the advantage of allowing time-critical routines (such as those used for data transmission or cryptographic algorithms) to still be programmed in assembly language, while allowing a high-level language such as Java to be used for all higher software layers.

Another possible solution to the speed problem would be hardware extension of the instruction sets of smart card processors. With this approach, approximately 80 % of the machine instructions of the Java VM could be handled by the processor. This would not be particularly difficult in technical terms, and it would yield an enormous increase in processing speed. This approach has not been taken because current high-performance 16-bit and 32-bit processors provide sufficient computing power, and they have the major advantage of being usable for a wide variety of applications.

The processing speed disadvantage of interpreted program code relative to compiled program code can be dramatically reduced by providing suitable programming interfaces for the interpreter. These applications programming interfaces (APIs) allow interpreted program code to call routines coded in the machine language of the host processor. The native routines called via the API run at the full working speed of the host processor.

Although this may initially appear to be an ideal way to achieve increased processing speed, it has certain drawbacks. First, the API for native code must be carefully designed to support a wide range of applications instead of only a few special cases. Nevertheless, this can be achieved with reasonable deliberation. The second drawback is more serious. Compatibility and the hardware independence of a programming language such as Java are only possible if all APIs are the same. There is no advantage to using a standardized programming language if it has to work with a variety of APIs that have different interfaces or functionalities. This would require making various adjustments to the source code for each platform with its own

API. This is also why Java Card Forum invests so much effort in the standardization of APIs, which is a fundamental requirement for achieving platform independence.

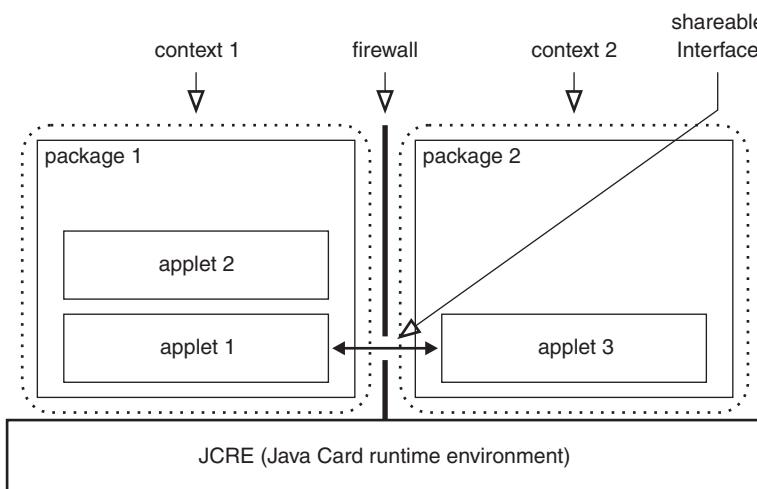
The speed of an application on a Java card is also highly dependent on the implementation of the application. With unfavorable programming, a Java Card application is very slow, but with favorable programming an application can certainly achieve the same speed as native programming.

### 13.16.2.7 Java Card runtime environment

The runtime environment of a Java card includes a transaction mechanism, management of logical channels and the applications selected via these channels, package loading and applet installation, deletion of applets and packages, and management of transient memory.

### 13.16.2.8 Application partitioning (firewalls)

From an IT perspective, individual applets in a smart card are fully isolated from each other as shown in Figure 13.37. Interactions are prevented by the security manager of the Java virtual machine. With Java Card, all applets of a package are automatically located in the same security context. Applets within a context can access each others' objects. For this purpose, the owner of each object (the applet that generated the object) is stored on the system heap. In addition, objects can have special attributes that are examined by the firewall during access checks. If the class of an object implements an interface that is directly or indirectly derived from the shareable interface, the object concerned is a shareable interface object. All other applications are allowed to access the interface methods of this object. However, it must be borne in mind that access to variables or methods that do not belong to the interface is not allowed. In addition, Java Card supports JCRC entry point objects and global arrays as



**Figure 13.37** Schematic representation of the relationship between two packages with one or two applets and the Java Card firewalls and associated security context. Applets 1 and 3 can use the shareable interface to exchange data through the firewall

**Table 13.23** JCRE entry point objects and global arrays in Java cards

Name	Type	Comment
APDU	JCRE entry point object	Object for application communication
AID	JCRE entry point object	Registry AID object
APDU Buffer	Global array	Buffer for command and response data

described in Table 13.23. These objects can only be created by the operating system, and they allow all applications to access their data.

If access to the methods or variables of an object is desired, the firewall checks whether the currently running application is allowed to access the object. The following pseudocode shows the process in simplified form.

```
void firewallCheck(object) {
    if (object.owner.package) == currentapplet.package) {
        return;
    }
    if (object.isJCREEEntryPoint()) {
        return;
    }
    if (object.isGlobalArray()) {
        return;
    }
    if (object.isSharable()) {
        return;
    }
    // Access denied
    throw new SecurityException();
}
```

In order for an object to access another application, certain supplementary data is necessary regardless of the firewall. The first thing that is needed is the reference to the object to be accessed. In addition, information about the interfaces, methods and variables of the object is necessary.

### 13.16.2.9 Command dispatching and application selection (*dispatcher*)

The dispatcher handles command dispatching and application selection. Logical channels compliant with ISO/IEC 7816 are supported in Java Card 2.2 and later versions. The management function maintains the state of each logical channel (open or closed) and which application is active on each channel. Each incoming command is first assigned to a logical channel as specified in ISO/IEC 7816. If the channel is open and an applet is active on the channel, the process() method of the applet is invoked with this command. If the channel is not open or no application is selected, a command response with an error status is returned.

In order to open or close channels and select applications on logical channels, the dispatcher supports the SELECT by Name and MANAGE CHANNEL commands specified in

ISO/IEC 7816. They are the only commands that are supported by a Java Card operating system. All other commands are forwarded via the appropriate logical channel to the selected applets, which means that the actions of these commands and the associated responses are determined by the applets.

The MANAGE CHANNEL command can be used to open and close logical channels. In this regard, it must be borne in mind that the base channel (channel 0) is always open and an attempt to close this channel will result in an error message. If a channel with an actively selected applet is closed, the applet is informed of this by calling its deselect() method. When a channel is opened, an applet can be implicitly selected on the newly opened channel. This is also reported to the applet by calling its select() method.

Only the SELECT by Name variant of the SELECT command is supported here. Other variants of the SELECT command, like all other commands, are forwarded to the active application on the appropriate logical channel. The entire AID of the requested object, or the first part of the AID, is supplied in the data portion of the SELECT command. If the AID is not found, the SELECT command is again forwarded to the active application on the logical channel. If an applet with a matching AID is found, it is initialized by calling its select() method, the applet is activated on this channel, and the SELECT command is forwarded to the process() method of the applet that has just been selected.

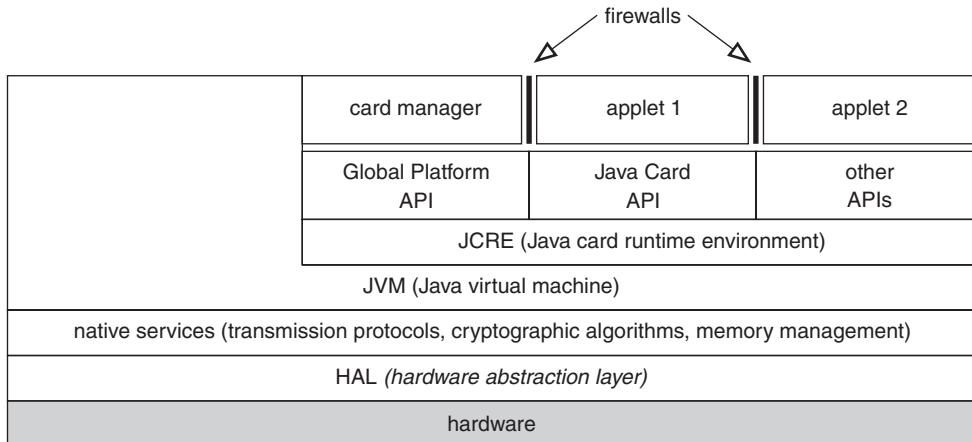
#### ***13.16.2.10 Transaction integrity (atomic operations)***

In case of changes to an object made implicitly by the virtual machine or the operating system, the system ensures that a sudden loss of power during a session will not cause the data of an applet to assume an undefined state. Every write access is performed as an atomic operation, which means that the operating system ensures that the values of variables are changed either completely or not at all, even in the event of a power failure. A system service call can be used to combine several write accesses into a transaction. This makes it possible to ensure unconditional integrity across several objects or processes. The programmer can close a transaction on completion or roll it back. In case of a power failure, the operating system ensures that all changes can be reversed. If an application opens a transaction but does not close it, the operating system automatically terminates the transaction and rolls it back.

#### ***13.16.2.11 Persistent and transient objects***

All objects in a Java card are stored in nonvolatile memory as persistent objects, which means the metadata of the operating system as well as the actual object data are stored in nonvolatile memory. Persistence is the ability of an object to exist beyond the execution time of a process. Persistent objects thus continue to exist after the end of a session or after a sudden loss of power without any consequent loss of data or data inconsistency. Any object exists as long as there is a reference that points to it. If the reference is deleted, the object is effectively no longer present, although it still occupies memory. This can be remedied by a file manager with garbage collection, which is provided by the specification of Java Card version 2.2 and later.

Java Card allows application programmers to create transient arrays. In this case the metadata of the array object is located in nonvolatile memory, but the actual array data is in volatile memory (RAM). The type of the transient data store must be stated when it is created. Clear on reset (CoR) Transaction integrity RAM arrays are preloaded with zeros when the Java card is reset. Clear on deselect RAM can only be used if the applet is active on a logical channel.



**Figure 13.38** Schematic representation of the basic architecture of a Java Card system. This example includes several typical APIs as well as application management for the Global Platform mechanisms. The location of the applications is shown in the form of two applets

When the applet is deselected, the data in this array is overwritten with zeros. Before a clear on deselect (CoD) RAM array can be accessed, the card must check whether the application to which the object belongs is active on a logical channel. If the application is not active, the array is not available, and the access causes a security exception.

### 13.16.2.12 Java Card application programming interface

To make the programming of smart cards in Java as easy as possible, there are packages available that provide standardized application programming interfaces (APIs) with functions that are useful for smart cards and fit into the structure of a Java Card system as shown in Figure 13.38. Three of these packages are mandatory for all Java cards that do not necessarily need certain key lengths or cryptographic algorithms. Packages with an 'x' (for 'extension') in their name are optional. They can be included if necessary. A variety of other application-specific packages are also available, such as packages with functions for GSM/USIM or Global Platform. The principal packages used in smart card applications are described in Tables 13.24 to 13.29.

**Table 13.24** The most important classes of the Java Card API packages java.io, java.lang, and java.rmi

Class	Description
java.io.IOException	Java Card I/O exception handling class; used for remote method invocation (RMI)
java.lang.Exception	Java Card exception handling class
java.lang.Object	Class for all Java Card classes
java.lang.Throwable	Class of all Java Card exceptions and errors
java.rmi.Remote	Remote interface; objects that implement this interface can be accessed via RMI
java.rmi.RemoteException	Class for handling exceptions in RMI processing

**Table 13.25** The most important classes of the Java Card API javacardx.apdu, javacardx.biometry, javacardx.crypto, and javacardx.external packages

Class	Description
javacardx.apdu.ExtendedLength	This interface serves as a marker for applets that can process commands with more than 255 bytes of data
javacardx.biometry.BioBuilder	Contains methods for generating biometric access controls
javacardx.biometry.BioTemplate	Basic interface for all biometric algorithms
javacardx.biometry.OwnerBioTemplate	Interface for biometric algorithms
javacardx.biometry.SharedBioTemplate	Interface for biometric algorithms that can be used by other applets
javacardx.crypto.KeyEncryption	This interface provides methods for decrypting keys that are transmitted in encrypted form
javacardx.crypto.Cipher	Basic class for all encryption algorithms.
javacardx.external.MemoryAccess	Interface for reading and writing external memory, such as Mifare memory
javacardx.external.Memory	Class for generating objects that enable access to external memory

**Table 13.26** The most important classes of the Java Card API javacard.framework package

Class	Description
javacard.framework.AID	Encapsulates the 5–16 byte application identifier (AID) specified by ISO/IEC 7816-5
javacard.framework.APDU	Provides methods for exchanging data between the smart card and the terminal at the APDU level
javacard.framework.Applet	Basic class for all Java Card classes; defines the interfaces used to integrate JCER with applets
javacard.framework.ISO7816	This interface encapsulates various constants defined in ISO/IEC 7816-3 and ISO/IEC 7816-4, such as offsets to various data elements in an APDU and a variety of return codes
javacard.framework.JCSystem	An important class for using transactions, creating transient arrays, access to the system, and access to the services of other applets on the card
javacard.framework.OwnerPIN	Provides a secure PIN implementation; implements the javacard.framework.PIN interface
javacard.framework.PIN	This interface defines the interfaces of a PIN with associated retry counter and validity status
javacard.framework.Util	Contains methods for filling, copying and comparing byte arrays and type conversion of short variables
javacardx.framework.Sharable	Shareable interface; methods of objects that implement this interface can be invoked by other applets

**Table 13.27** The most important classes of the Java Card API javacard.framework.service package

Class	Description
javacard.framework.service. BasicService	Basic class for all command-based services, such as RMI
javacard.framework.service. Dispatcher	Class for dispatching incoming commands to registered services
javacard.framework.service. Service	Interface to oncard services
javacard.framework.service. RemoteService	Interface to services that can be used from outside the card
javacard.framework.service. RMIService	Implementation of the service for accessing the methods of remote objects
javacard.framework.service. SecurityService	Extends the Service interface with methods for secure communication

**Table 13.28** The most important classes of the Java Card API javacard.security package

Class	Description
javacard.security.AESKey	This interface specifies access to AES
javacard.security.DESKey	This interface defines and provides methods for accessing DES and triple DES with two or three keys
javacard.security.DSAPrivateKey	This interface provides access to the private DSA key generating signatures
javacard.security.DSAPublicKey	This interface defines and provides the access methods for the public DSA key for verifying signatures
javacard.security.ECPrivateKey	This interface provides access to the private elliptic curve key generating signatures
javacard.security.ECPublicKey	This interface defines and provides the access methods for the public elliptic curve key for verifying signatures
javacard.security. InitializedMessageDigest	Basic class for pre-initialized hash algorithms
javacard.security.KeyAgreement	Basis for key exchange processes, such as the Diffie–Hellman algorithm [IEEE P1363].
javacard.security.Key	Basic interface for all keys.
javacard.security.KeyBuilder	Class for generating keys in the card
javacard.security.KeyPair	Container class for a key pair (public and private keys)
javacard.security.KoreanSEEDKey	This interface specifies the access key for the Korean SEED algorithm
javacard.security.MessageDigest	Basic class for all hash algorithms.
javacard.security.PrivateKey	Interface for private keys for asymmetric cryptographic algorithms
javacard.security.PublicKey	Interface for public keys for asymmetric cryptographic algorithms
javacard.security.RandomData	Basic class for generating random numbers
javacard.security. RSAPrivateCrtKey	Interface for generating private RSA keys using the Chinese remainder theorem (CRT)
javacard.security.RSAPrivateKey	This interface provides access to the private RSA key
javacard.security.RSAPublicKey	This interface provides access to the public RSA key
javacard.security.SecretKey	Interface for all private keys
javacard.security.Signature	Basic class for all signature algorithms
javacard.security. SignatureMessageRecovery	Extension of the Signature class for signatures that support message recovery

**Table 13.29** The most important classes of the Java Card API javacardx.framework.math, javacardx.framework.tlv, javacardx.framework.util, and javacardx.framework.util.intx packages

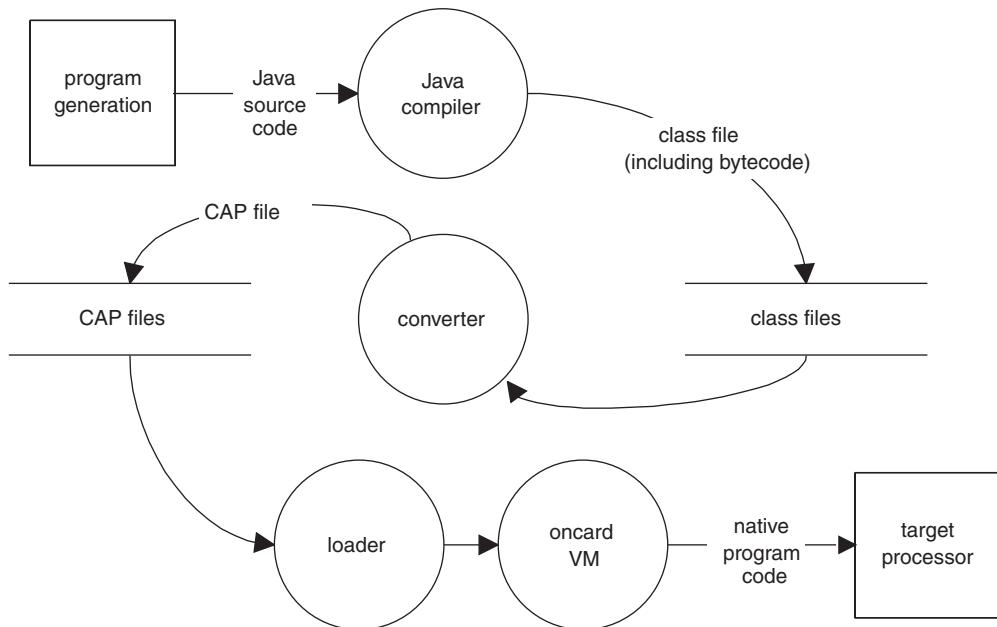
Class	Description
javacardx.framework.math. BCDUtil	Utility for converting and checking BCD numbers
javacardx.framework.math. BigNumber	Class for storing and calculating large numbers in hexadecimal or BCD format
javacardx.framework.math. ParityBit	Class for calculating parity bits (e.g. in DES keys)
javacardx.framework.util. BERTag	Basic class for managing tags according to the BER convention
javacardx.framework.util. BERTLV	Basic class for managing BER TLV structures
javacardx.framework.util. ConstructedBERTag	Class for constructed BER tags
javacardx.framework.util. ConstructedBERTLV	Class for constructed BER TLV structures
javacardx.framework.util. PrimitivBERTag	Class for primitive BER tags
javacardx.framework.util. PrimitivBERTLV	Class for primitive BER TLV structures
javacardx.framework.util. ArrayLogic	Utility for filling, copying and comparing arrays of various data types
javacardx.framework.util. intx.JCInt	Utility for generating a transient integer array and writing or reading integer values in arrays

As the program code behind the API can be generated in the machine language of the target processor, this approach not only provides a standard interface but also yields an enormous increase in processing speed.

The obligatory ‘java.lang’ package forms the basis for Java in smart cards. It defines the elementary classes for exceptions. It is complemented by the package ‘javacard.framework’, which defines the core functions for Java Card applets such as elementary classes for applet management, data exchange with the terminal, and various constants in the context of ISO/IEC 7816-4. The cryptographic functions package ‘javacard.security’ provides interfaces to various cryptographic algorithms. For export control reasons, this package is constructed such that it does not allow the smart card to be used as a general-purpose encryption and decryption tool. The optional package ‘javacardx.crypto’, which contains the interfaces to the associated decryption methods, is needed for this capability. Recent versions of the specification have defined other packages, which are described in the following tables.

#### 13.16.2.13 Software development for Java in smart cards

The basic process of developing and executing Java software for smart cards is depicted in Figure 13.39 in the form of a data flow diagram. The first thing the programmer does is to generate the Java source code using a text editor. A number of rules must be observed in this

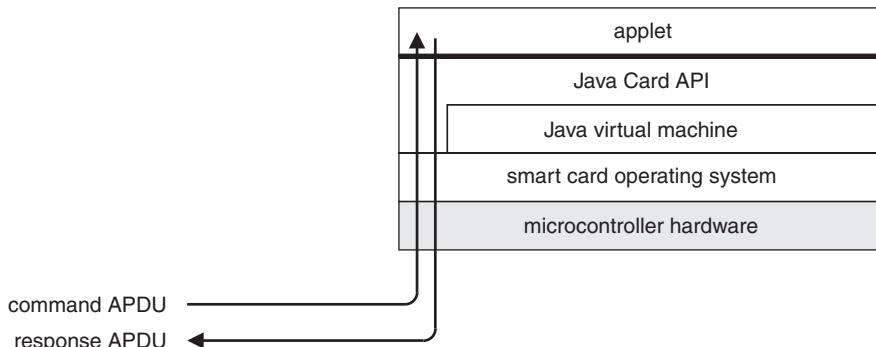


**Figure 13.39** The usual program development process up to execution of a program by the Java virtual machine running in the smart card microcontroller

process. The applet must be derived from the abstract class `javacard.framework.Applet`, and the missing methods – especially the `process()` method – must be implemented. In addition, the static method `install()` must be implemented. This method generates an instance of the applet, and it must invoke the `register()` method of the instance. Of course, the application can only use the classes and methods of Java Card. The `vector` class, which is present in normal Java, is not available. After generating the applet and its utility classes, the programmer compiles them with his or her favorite Java compiler and receives class files containing machine-independent bytecode. Up to this point, the process is identical to Java programming for PCs.

The class files of a package are then sent to the Java Card converter (which is actually the offcard portion of the Java virtual machine), which checks the format, syntax, field references, and so on. If all these checks are passed successfully, the Java Card converter generates a card application file (CAP file). This means that a CAP file always contains the routines and applets of a Java package. If necessary and depending on the application, a digital signature is added to the CAP file to provide the assurance that the CAP file has been checked by the offcard VM and is authentic. In the absence of a verifiable signature, the security of the oncard VM could be bypassed by using a manipulated applet, since the oncard VM cannot perform all the checks due to memory space limitations. After this, the applet is loaded into the smart card in the form of a CAP file. This is usually done using the Global Platform mechanism.<sup>17</sup> It first verifies the digital signature, which is usually present, and then passes the applet to the oncard VM after it has been checked. Now that the package is in the card, the applet can be installed using a command and the selected with the `SELECT` command. With this command and every

<sup>17</sup> See also Section 13.13, ‘Application Management with Global Platform’, on page 485



**Figure 13.40** Data flow from the command APDU to the applet and the corresponding response APDU, with reference to the layer model of a Java smart card

other command, the process method of the applet is invoked, which causes the bytecode of the applet to be interpreted by the Java Card virtual machine and processed. In normal operation, command APDUs pass upward through the hierarchical layers of the smart card system to the applet as illustrated in Figure 13.40, while responses pass downward through the layers on their way to the terminal.

In actual practice, the process is naturally somewhat more complicated. It is to be hoped that the developer does not immediately start writing Java code after receiving the task assignment, but instead uses analysis and design methods to determine the actual requirements before starting to program.

In order to quickly locate errors during and after the coding process, the developer uses a Java smart card simulator. This allows the developer to follow the execution of the code step by step, examine variables, and make any necessary corrections quickly and easily.

Besides this, a suite of tests is run for relatively large projects and those that are critical for security. These tests check all good cases and the most important bad cases of commands and responses. Source code inspection by an independent party may also be included.

As you can see from this example, Java for smart cards significantly reduces development time, and it reduces possible error sources as a secondary benefit. However, coding by itself is only one of many aspects of developing a smart card application. The primary advantage of Java for smart cards is that it allows a large number of developers to generate executable programs for smart cards, rather than just a few software developers employed by card manufacturers. If a particular applet must be loaded into a very large number of smart cards in identical form, some implementations of Java Card also allow portions of the applet to be defined as part of the ROM mask, although the portions of the applet that contain modifiable data must remain in EEPROM. Such an applet is often called a ROMable applet.

In generating Java applets for smart cards, several properties of the current Java Card specification should be taken into account in addition to the particular features of the operating system being used.<sup>18</sup> They are listed and briefly described below.

<sup>18</sup> Good summaries of software development for smart cards are provided by Zhiqun Chen [Chen 00] and Wolfgang Rankl [Rankl 06].

### 13.16.2.14 Execution speed

Aside from its memory demands, the major point of criticism of Java for smart cards is doubtless its low execution speed. However, it is relatively difficult to make fair comparisons between assembly language programs and Java. This is primarily because it is not essential to create the same program flow in Java as in assembly language, as long as the programs behave the same way at the interface to the terminal. For example, a file system is not always necessary with a Java program, and probably nobody would ever program a cryptographic algorithm in Java.

Another general consideration in programming is that the methods of the Java Card API should be used as much as possible because they are in part coded in the native language of the target processor. This can yield a considerable increase in the processing speed of the interpreted Java code.

When typical smart card commands are implemented in Java with intensive utilization of interfaces to native library routines, it can be assumed that the execution time (excluding data transmission time) will be approximately 30 % longer than with a comparable implementation in C or assembly language. With unfavorable programming, a Java program can easily be a factor of 2 to 3 slower than a corresponding native program.

Another point that applet programmers should always bear in mind is that most Java variables are located in persistent memory. As the transaction mechanism secures the writing of the variables, up to three write operations to EEPROM are performed for each write access. A write operation typically takes 3 to 4 ms, so writing to a variable can consume up to 12 ms. You can easily imagine the speed nightmare that occurs when such a variable is used as a loop counter.

Consequently, applet developers should always be aware of which type of memory they use for their variables. The following types of variables are always located in EEPROM or flash memory: member variables, static variables, and array data. By contrast, local variables and transient array data is held in RAM.

### 13.16.2.15 File system

It is not mandatory for an applet to have its own file system. For some applications, it can be fully adequate to create file-independent data objects that can be accessed using either standard commands or commands defined by the programmer (private-use commands). The advantage of applets without file systems arises from the ever-present basic need to conserve memory usage in smart cards. In addition, the object-oriented nature of Java allows data objects to be accessed by calling methods with the suitable calling conditions, which makes it possible to implement access mechanisms that fulfill very specific requirements in certain applications. For example, an application or user could be allowed to bequeath their access privileges to other entities.

However, examination of typical applications in the smart card domain and the PC domain reveals that data is very often stored and managed in file-oriented structures. Although this option is not excluded by the Java Card specification, it is unfortunately not directly supported. File management, for example in accordance with ISO/IEC 7816-4, has to be implemented by the programmer in Java using classes designed for this purpose. This is often necessary

in order to provide a basis for issuing Java smart cards that are compatible with previous applications that use standard file structures.

#### ***13.16.2.16 Cryptography and export restrictions***

In many countries, smart cards with general-purpose operating systems having free access to data encryption and decryption functions via internal interfaces are subject to export restrictions. As a result, such cards cannot be exported at all in some countries, or the exporter must wait several months for an export permit to be issued by the responsible authorities.

Consequently, the classes for cryptographic functions are structured in Java Card such that they can be used without restriction for general data decryption and MAC computation but not for encryption. This is fully adequate for many applications, and in many countries it allows a simplified export permit procedure to be used.

If an application does require data encryption, the card manufacturer can incorporate the `...cryptoEnc.DES3.EncKey` and `...cryptoEnc.DES.EncKey` classes to make encryption possible. From a cryptographic perspective, however, it is certainly possible to devise easily implemented methods that can be freely used for data encryption and decryption without using these two ‘encryption’ classes.

#### ***13.16.2.17 Future generations of Java cards***

The development of Java Card has not reached an end with the current version. The next generation bears the name ‘Java Card 3.0’ and incorporates a host of technical innovations. Sun published the definitive specification for this version in the spring of 2008. With regard to backward compatibility, there is a compromise that was not present in past versions in this form. There are two versions of Java Card 3: classic and connected. The classic version corresponds to version 2.x of the Java Card specification and is thus compatible with billions of cards already in use. The real innovations are contained in the connected version, which is not fully compatible with the 2.x specification. However, applets that have been developed according to the previous specification only have to be recompiled in order to make them suitable for loading and running in Java Card 3.0.

The new functions include substantial extensions, such as a 32-bit virtual machine (VM), direct loading of class files, and oncard byte code verification. The language elements now include the variable types `char`, `int`, `long` and `string`, and multidimensional arrays are possible. In addition, there is support for multithreading, the USB and MMC interfaces, and contactless interfaces compliant with ISO/IEC 14443. It is also possible to use TCP/IP to communicate with the card via the above-mentioned interfaces. An HTTP server (SCWS)<sup>19</sup> is also defined in the new specification.

This sizable functional expansion relative to the previous versions can be expected to require major revisions to corresponding operating systems because fundamental changes to the software architecture are necessary. The demands on the hardware of the smart card microcontroller are also substantial. Initial estimates indicate that a Java Card 3.0 implementation with the previously mentioned functionality would require approximately 500 KB of program

<sup>19</sup> See also Section 23.3, ‘Smart Card Web Server, on page 912

code and 25 KB of RAM. From this, it can be fairly easily concluded that at least in the initial years, the connected version of Java Card 3.0 will be used only at the upper end of the smart card performance scale.

#### 13.16.2.18 Summary and future prospects

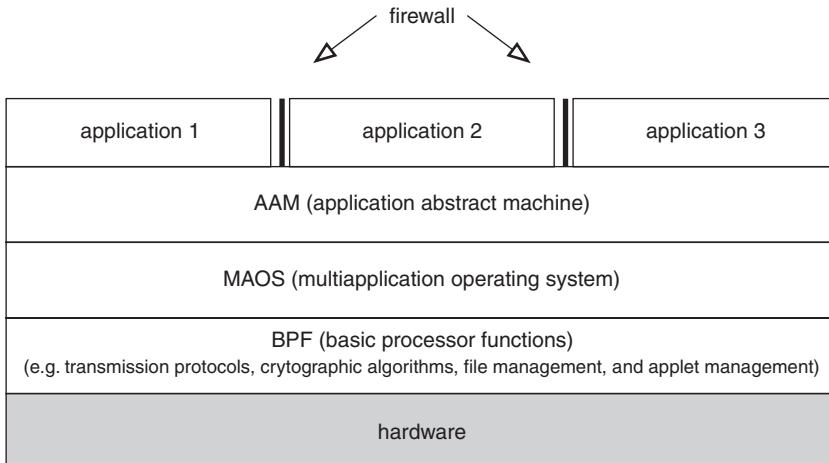
Despite the unrelenting hype surrounding Java, it should not be forgotten that it surely will not prove to be the solution to every informatics problem of past and future years and that it may not always meet everyone's expectations. You only have to consider the fate of previous programming languages that are no longer in fashion, such as Pascal ('modular'), Lisp ('AI for everyone'), C ('portable'), and C++ ('reusable program code'). Although these languages have advanced information technology by orders of magnitude, many of their predicted benefits failed to materialize. Nevertheless, a new era in the history of smart cards began with the introduction of Java, since it is the first language that allows third parties to run executable program code in smart cards in a straightforward manner. Java has now become the standard programming language for smart card applications. Version 3 will also lead to closer correspondence to regular Java on PCs and the associated functionality.

Loading and managing applets using applet management systems can only be expected to become established in the longer term, due to its high complexity. Consequently, in the foreseeable future applets with their associated individual data will be loaded in Java cards in the usual finishing operations of card production and will remain unchanged for the duration of the card's life cycle.

As far as can be seen at present, Java Card is the only global technology for implementing program-based applications in smart cards. There are now a large number of specifications, APIs and extensions to Java Card in a very wide spectrum of application areas, which give it a very broad basis. In addition, the degree of compatibility of Java cards made by different manufacturers that has been achieved in recent years makes them economically attractive to card issuers because it eliminates their dependence on individual manufacturers. Java Card has also made it possible to achieve short development cycles for smart cards, which is an important consideration in the IT world, while at the same time allowing all known and proven development methods to be used. In particular, reduced development and delivery times can open up new markets and applications for smart cards.

### 13.16.3 Multos

Multos is a multiapplication smart card operating system originating from the development of the Mondex system for electronic purses. Based on this system, an operating system primarily optimized to meet the requirements of electronic payment systems has been developed in several steps. Its basic architecture is illustrated in Figure 13.41. The publisher of the specifications, license issuer and operator of the necessary certification services for Multos is the Maosco Consortium [Maosco]. Most of the Multos specifications are confidential, so here we can only present a summary of the features of this operating system. One interesting detail is that certain kernel operating system components of Multos are certified in accordance with ITSEC E6, which is the highest possible evaluation level.



**Figure 13.41** Schematic representation of the basic architecture of the Multos smart card operating system. This example includes three applications based on Multos that have been generated in MEL

Multos is an example of a typical ISO/IEC 7816-4 compliant operating system and can interpret downloadable program code. The program code is typically developed in C and translated into the Multos executable language (MEL) using a special compiler. MEL is hardware-independent program code that is executed by a stack-oriented virtual machine called the application abstraction machine (AAM). From within MEL, an application can access the operating system services of the multiapplication operating system (MAOS) via various interfaces.

Before an application can be loaded into a Multos smart card, it must be digitally signed by a licensed Multos certification service using relatively elaborate mechanisms. As is usual with payment cards, large portions of the completion process are also specified in detail.

### 13.16.4 BasicCard

A smart card operating system with an interpreter for the Basic programming language has been available from the German company Zeitcontrol [Zeitcontrol] since 1996. This operating system is called BasicCard, and it is available in several versions with different features and for hardware platforms with various memory sizes. Along with Java Card and Multos, it is one of the few multiapplication operating systems that allows executable program code to be downloaded by third parties.

The method used to generate downloadable programs for BasicCard is similar to conventional Basic interpreters. A compiler translates the source code into P-code, which is transferred to a memory region of the smart card microcontroller reserved for this purpose using a special loader program. The program code stored in this region can then be processed by the interpreter as necessary.

With regard to data types, control structures and functions, the version of Basic supported by Basic Card corresponds to the presently common simpler dialects of this programming language, which has existed for several decades. It has also been extended to include some

functions specific to smart cards, such as an interface to the smart card file system. In addition, it supports the T = 0 and T = 1 data transmission protocols for contact cards and contactless data transmission in accordance with ISO/IEC 14443 B. For typical security applications, a variety of cryptographic algorithms can be called via an interface, including DES, triple DES, IDEA, AES, RSA with a key length of 1 024 bits, elliptic curves with a key length of 167 bits, and the SHA-1 hash algorithm.

The program code is very compact compared with other smart card operating systems with interpreters, and the execution speed is relatively high. These two aspects are primarily due to the fact that in procedural terms, Basic can be easily and quickly interpreted, and that no sophisticated security mechanisms are used for preparing applications. For certain applications that require simple and quick smart cards program development, BasicCard is certainly a good alternative to other smart card operating systems.

The most unusual aspect of BasicCard is that it is the product of a relatively small company that has continued to refine this product over many years, rather than one of the giants of the IT industry as with some other smart card operating systems with interpreters.

### 13.16.5 Linux

Since the late 1990s, the open-source operating system Linux has altered large segments of the software industry. Up to now, the focus of Linux has been high-performance computers in the PC domain, although efforts to establish Linux in the environment of typical microcontroller applications have been underway for some time. However, currently available versions of Linux require a level of performance that typically can only be provided by 32-bit processors, along with memory requirements of the order of several hundred kilobytes of ROM and a few dozen kilobytes of RAM. Current smart card microcontrollers cannot yet meet these requirements, although it is certainly conceivable that the hardware requirements of Linux could be further reduced. At the same time, the performance of smart card microcontrollers increases with each new generation, so Linux for smart cards might be available in the not too distant future.

Beside Linux, it would naturally be possible for another open-source operating system for smart cards to appear. The primary consideration is that it must be license-free, since the large quantities in which smart cards are produced and used, with correspondingly high license costs, is one of the most significant barriers to the use of standard operating systems in smart cards.

## 13.17 THE SMALL-OS SMART CARD OPERATING SYSTEM

Up to now we have briefly described the properties and principles of smart card operating systems. For readers who wish to immerse themselves more deeply in this subject, this section presents a complete operating system. It describes in detail the internal relationships of a conventional smart card operating system compliant with ISO/IEC 7816-4 or TS 51.011.

The name of the operating system described here is ‘Small-OS’, which reflects its very small memory requirements and the fact that it can run on relatively modest hardware platforms. It is written in a pseudocode resembling Basic. The pseudocode presented here cannot be compiled as is, since some assignments have not been coded down to the last bit and are formulated

**Table 13.30** Summary of the features of Small-OS

OS component name:	Small-OS
Typical application areas	Multiapplication without user-generated program code
Hardware requirements:	8-bit CPU; ROM: $\approx$ 8 KB, EEPROM: $\approx$ 1 KB, RAM: $\approx$ 128 bytes
Command set:	Subset of ISO/IEC 7816-4 with extensions Commands: SELECT, READ BINARY, UPDATE BINARY, READ RECORD, UPDATE RECORD, VERIFY, INTERNAL AUTHENTICATE
Data transmission:	T = 1 data transmission protocol Divisor (CRCF) set to a fixed value of 372 (compliant with ISO/IEC 7816-3) PPS not supported Secure messaging not supported
File system:	One DF level Structures for working EFs: transparent, linear fixed Structure for internal EFs: linear variable (for PINs and keys) One EF Key allowed per directory (i.e. per MF or DF) No dynamic file system (no file deletion or creation, no free memory management); maximum transparent EF size: 255 bytes
State machine:	Independent secure states for MF and DFs Number of secure states: 256 (0 . . . 255) Initial state = 0 Only one allowed input state for using a PIN or a key Only one allowed input state for file access (i.e. ' $<$ ' and '=' comparisons are not possible) No cross-level key access (the EF key is always selected via the currently selected directory (MF or DF)) PIN addressing: at most two PINs (ref. no. 1 and no. 2) Key addressing: up to 31 keys (ref. no. 1 . . . 31) PINs and key retry counters allow up to 15 unsuccessful attempts
Cryptographic algorithm:	AES
Program code:	Generation or loading by external parties not possible

in descriptive terms. We do not wish to present page after page of true program code in a language such as C or C++, which can be incomprehensible and boring to read, but instead an illustrative and enlightening example. The main features of Small-OS are listed in Table 13.30.

The ease of understanding of pseudocode vastly outweighs the benefits of suitability for direct compilation and execution provided by a real programming language. Pseudocode lets you concentrate on the basic process instead of getting lost in the details of the implementation. The operating system described here is a platform-independent implementation that is not tailored to any specific hardware, but there is also a working implementation of Small-OS. It is not programmed in assembly language to run on a smart card microcontroller, but instead

as a simulation called the Smart Card Simulator. This program is available at no charge under GPL via the Internet [Rankl].<sup>20</sup>

### 13.17.1 Programming in pseudocode

A few remarks about programming style and the programming of Small-OS are in order here. The pseudocode, which is based on Basic, essentially amounts to a semi-formal description of the Small-OS smart card operating system. Similar operating system characterizations are often generated for software evaluations according to the Common Criteria, where they provide the basis for evaluation and source code review. The pseudocode in this section, which is presented in tables with extensive comments, thus represents a good example of how formal processes in smart card operating systems are portrayed. Similar forms of explanatory description can be found in the EN 1546 family of standards, for example, in which the internal processes of smart cards for electronic purses are semi-formally described.

The notation used in the pseudocode is described at the front of this book.<sup>21</sup> The program code is based on the usual dialects of Basic with object-oriented extensions. Only generally understandable constructs are used. All labels, constants and references are in English. Numerical values are normally shown in hexadecimal form using ISO notation (e.g. ‘42’). However, decimal or binary form is used where necessary to aid understanding, using a notation based on the ISO notation. For example, all countable values such as lengths are shown in decimal form.

Nobody would program a smart card operating system in this form in assembly language or C because it would be far too cumbersome. One of the design objectives with Small-OS was to create a simple yet powerful smart card operating system in a manner that is easily understood and well commented. Intentionally, no attempt was made to minimize program execution time, code size, RAM usage or stack depth, since doing so would seriously impair the readability of the code. In real smart card operating system programming, for example, JUMP instructions are sometimes used in place of CALL instructions for calls to rarely used subroutines, since this saves two bytes of valuable stack space. A flag set before the subroutine is called with the JUMP instruction is used to determine the return address when the subroutine process is completed. This sort of optimization is not found in Small-OS, for the reasons just given. The pseudocode has been optimized only with regard to readability and comprehensibility. The resulting deviations from real smart card operating systems are identified wherever they occur, either in the text or in the commented pseudocode.

Most of a smart card operating system is located in ROM, due to the notorious scarcity of memory in smart card microcontrollers. It thus cannot be modified after the chip has been manufactured. Except for trivial miniprograms, software cannot be produced with zero errors, but only with as few errors as possible. An error in ROM program code has serious consequences. To make it possible to use bug fixes to patch out such errors, jumps to EEPROM are provided at critical locations in the ROM code. This technique has a well-established tradition in software engineering and is not specific to smart cards. Nearly identical mechanisms

<sup>20</sup> Due to time constraints and lack of demand, it is unfortunately not possible to update the Smart Card Simulator at present. Consequently, in its current form it does not correspond to the current versions of the standards in certain respects

<sup>21</sup> See Program code on page xxvii

are used in the Mac OS and OS/9, for example. Locations in EEPROM that are called by the program code, called ‘handles’, are used for this purpose. A handle normally contains only a RETURN instruction, which causes an immediate return to the calling code. If a bug fix for the ROM code is necessary, corrective code is loaded in the EEPROM at a handle location. In this case, the call to the handle does not produce an immediate return. For reasons of simplicity and ease of understanding, this mechanism is not included in Small-OS. Naturally, it is not necessary with microcontrollers that have flash memory instead of ROM.

The relatively detailed description of this smart card operating system provides an interesting opportunity to examine several typical forms of attack directly in the pseudocode. At suitable locations, possible attacks and defense measures at the operating system level are described in detail. For example, it is possible to examine the process of an attack on the PIN by comparing processing times, which has now become a classic form of attack, in full detail in the pseudocode. A comprehensive description of typical attacks and their defenses is given in Chapter 16 on page 667.

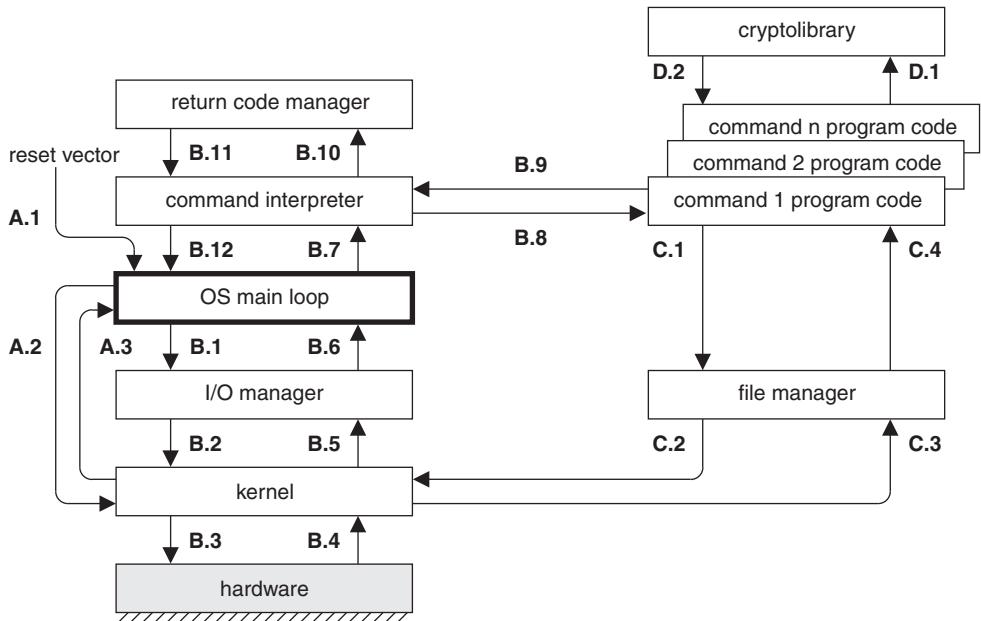
### 13.17.2 Design aspects

The above considerations led to the following design criteria that guided the design and programming of Small-OS, which are summarized in Table 13.31. Small-OS should be a simple smart card operating system that does not need a lot of program code and has a low level of complexity, just like real smart card operating systems. This makes its structure comprehensible and easy to grasp. It has a strictly modular structure, which means that it can be directly extended with additional commands at reasonable cost. The file system and the supported commands are without exception compliant with the international ISO/IEC 7816-4 standard with several extensions commonly used in the smart card world. The layer structure of Small-OS is shown in Figure 13.42.

Small-OS is intended to be used in systems in which it is not necessary to download applications after the cards have been issued. However, several different applications could run in the smart card independently of each other, depending on the amount of available memory. This means that Small-OS is a multiapplication operating system, although it is not possible to download executable program code to the card and run it in the card. In summary,

**Table 13.31** Small-OS design criteria in order of priority

Criterion	Reason
1 Compatibility with ISO/IEC 7816-4	The international standard for smart card operating systems
2 Robustness	High reliability and high fault tolerance
3 Low level of complexity	High reliability and readily comprehensible operation
4 Modular structure	High reliability, high fault tolerance, and easily extended
5 Small memory requirement	An absolute must for smart card operating systems
6 Multiapplication operating system	This has become a standard capability
7 No support for downloadable code	High reliability and low complexity
8 Similar to real smart card operating systems	Small-OS is a realistic and instructive example



**Figure 13.42** The layer structure of Small-OS and the resulting calling scheme. Calls labeled ‘A’ are used when the operating system starts up, calls labeled ‘B’ are used for command execution, and calls labeled ‘C’ are used as necessary inside commands to access the file system. Calls labeled ‘D’ are used with cryptographic algorithms. The numbers indicate the sequence of the program calls

Small-OS is comparable to the first general-purpose smart card operating systems, such as are still used in various forms for GSM applications.

Among other things, the ISO/IEC 7816-4 standard describes a basic file system and several basic commands for smart cards. In this way, it primarily characterizes the interface to the smart card instead of the internal structure of the operating system. This standard also defines numerous options, and it contains some passages that are unfortunately subject to interpretation. The resulting wealth of possible variants must be curtailed in practice by specifications, such as TS 51.011, to ensure compatibility between different implementations.

With actual smart card operating systems, therefore, the designation ‘ISO/IEC 7816-4 compatible’ by no means indicates that they all have the same features and characteristics. This would only be possible with a detailed specification, which in certain respects would be a specific interpretation of the ISO/IEC 7816-4 standard. Consequently, in practice the behavior of different operating systems is the same only when commands are executed successfully, with various differences in case of errors. With Small-OS, interpretations of the ISO/IEC 7816-4 standard are usually identified as such in the pseudocode. With allowance for interpretation of the standard, Small-OS is truly compliant with ISO/IEC 7816-4, and as much as possible it corresponds to the usual interpretations of the standard in the smart card industry.

Major differences between operating systems are frequently found in the return codes. As the use and priority of individual return codes are not described in detail in ISO/IEC 7816-4, assumptions must be made. In contrast to all other operating systems, Small-OS at least has the advantage that the code is public, so it is always possible to determine where each return code is generated.

If Small-OS were implemented for a smart card microcontroller, it would need around 5–6 KB of ROM, 128 bytes of RAM, and at least 1 KB of EEPROM, depending on the number of files needed for the applications. This assumes as a general condition that only the T = 1 data transmission protocol is used and AES is used as the cryptographic algorithm. If certain applications need more memory, a microcontroller with more EEPROM could be used without any problems. This would not affect the operating system or require any modifications.

### 13.17.3 File access

The file system described in the ISO/IEC 7816-4 standard provides an enormous number of options for access conditions and key management. A solution that is often used in practice with multiapplication operating systems was chosen for Small-OS. The file headers of working EFs include default states for the various access commands, such as the condition `....AccessCondition.Read` for the READ command. Each state is represented by a positive integer. This means that the header of each EF has separate state values for read and write accesses, which must be attained in the current directory before the command can be executed. State 0 is the base state (idle), so this condition means that all accesses are allowed.

Separate state variables are assigned to the MF (`SecurityState.MF`) and the currently selected DF (`SecurityState.DF`). They can be altered by security-related commands (VERIFY and INTERNAL AUTHENTICATE). In case of an access to a specific EF, the current state in the directory is compared with the required state in the EF file header. If the actual and required states are the same, the file may be accessed for writing or reading.

Real smart card operating systems often support a large variety of less-than, greater-than, greater-than-or-equal, and not-equal comparisons. It is also frequently possible to define several possible access states independently in the file header. This further complicates a process that is already not exactly simple, and for this reason it is not supported in Small-OS. In principle, however, Small-OS could be extended to provide this support without any structural changes.

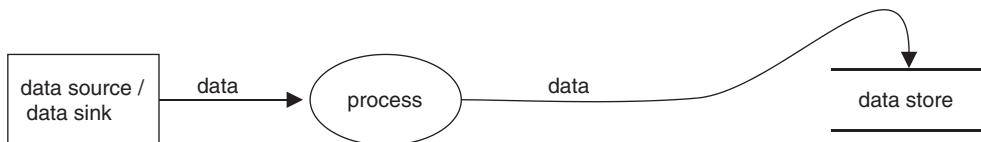
### 13.17.4 Access to internal secrets (PINs and keys)

All PINs and keys are held in specific internal EFs. These EFs are called ‘EF Key’ here. Files of this type can be read and written only by the operating system. External selection or access is not possible. There are no mechanisms or even rudimentary mechanisms in the design that would allow external access to these EFs. This is a part of the security philosophy of the Small-OS smart card operating system.

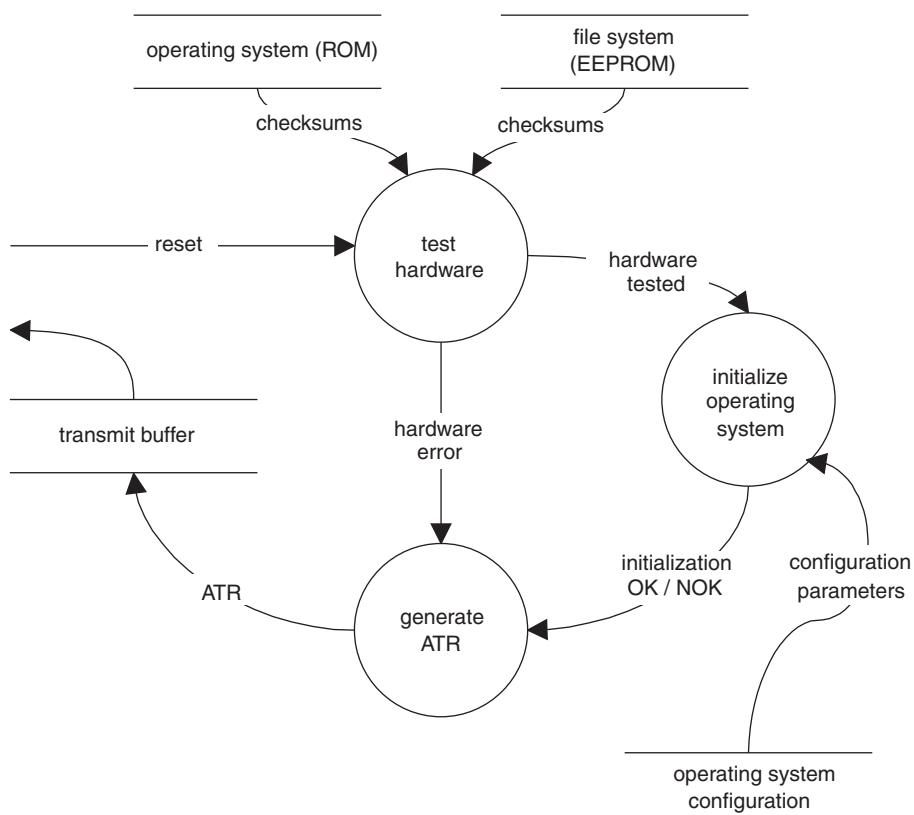
Only one EF Key can be created for each directory. It automatically has a linear variable structure to enable it to store PINs and keys of various lengths in the least possible amount of memory. Each record in an EF Key contains either a PIN or a key, with an address number that is unique within the file (`....KeyNo`). A state value is stored for each secret object (PIN or key). It defines the state (`....EntryState`) necessary for using the object with a command (VERIFY or INTERNAL AUTHENTICATE). The result state (`....ResultState.OK` or `....ResultState.NOK`) is set in the directory of the EF Key file according to the result of command execution (such as PIN comparison successful or not successful). In addition, a retry counter (`....RCntr`) is assigned to each secret object. This counter is incremented on each unsuccessful result until it reaches its maximum value. If the retry counter reaches its maximum value (`....RCntrMaxValue`), the associated secret object can no longer be used.

Some smart card operating systems allow cross-level access to keys. This allows keys stored in the next higher directory of a DF to be accessed from within a particular DF. The mechanisms needed for this are not included in Small-OS because the benefits of this feature would not justify the amount of code needed to implement it. Key access using aliases is also not implemented, for the same reason.

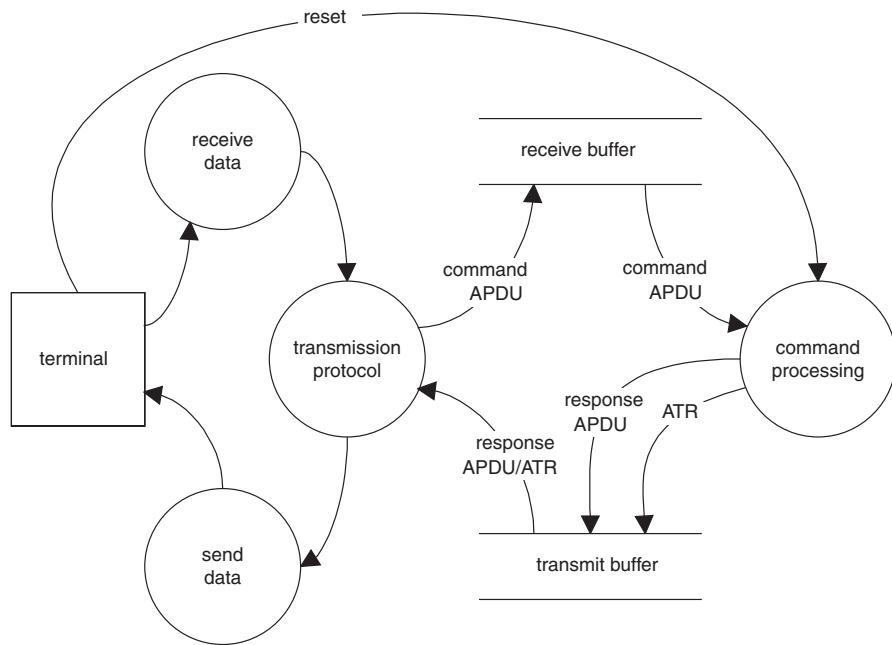
The basic data flows of Small-OS are shown in Figures 13.43 to 13.46.



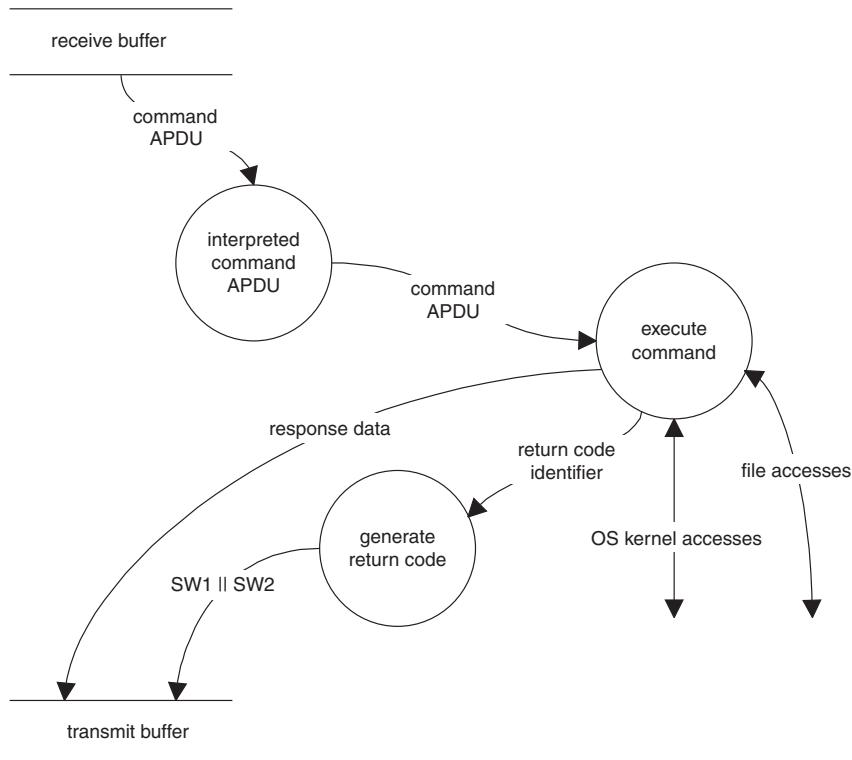
**Figure 13.43** The symbols used in the data flow diagrams for Small-OS. Data sources and data sinks, which are also called terminators, are objects outside the system concerned that exchange data with the system. A process is located inside the system concerned. It processes input data flows and generates output data flows. A data store is a storage place for data, which can be read and written. Robertson [Robertson 96] provides a detailed introduction to data flow diagrams



**Figure 13.44** Small-OS: data flow diagram of the operating system start-up and test processes



**Figure 13.45** Small-OS: data flow diagram for reset and data transmission



**Figure 13.46** Small-OS: data flow diagram of the command processing process

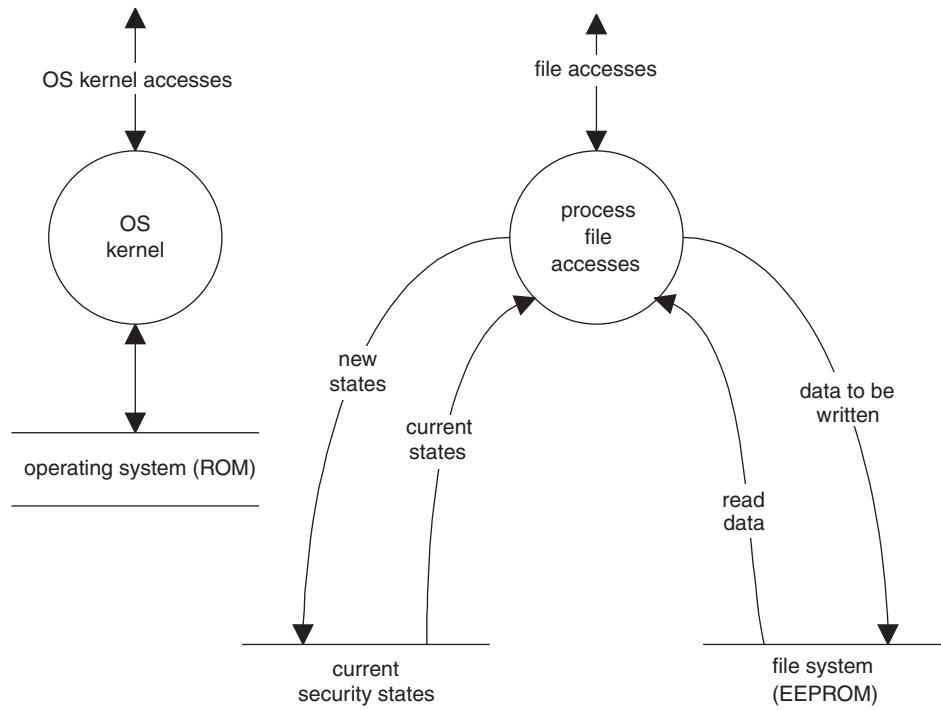


Figure 13.47 Small-OS: data flow diagram of the file access and OS kernel access process

### 13.17.5 Small-OS constants

As a rule, constants are used for numerical and nonnumerical values in the pseudocode where reasonable. This is simply good programming style, and it considerably increases the readability of the code. All constants are identified by the prefix 'C\_'. The values of these constants usually depend on the target hardware or the implementation, so they are not further defined here. All constants for the two-byte return codes have the prefix 'C\_RC\_'. The constants used in Small-OS are listed in Table 13.32. The associated return codes are listed in Table 13.33 on the next page. In practice, the constants of an operating system are usually stored in unalterable form in the ROM.

### 13.17.6 Small-OS variables

The variables of Small-OS can basically be divided into RAM and EEPROM variables. The RAM variables are re-initialized each time the smart card is reset and retain their values only for the duration of one session. However, data can be stored in RAM variables without any time delay, and the number of write cycles is unlimited. EEPROM variables, by contrast, are typically used primarily to implement file management, which requires data and access conditions that persist across sessions. The variables used in Small-OS are listed in Tables 13.34 and 13.35.

**Table 13.32** Constants used in Small-OS (excluding return code constants)

Constant	Meaning
C_Error	Constant for general errors (e.g. when calling a subroutine)
C_InvalidPointer	Constant for an invalid pointer value
C_Equal	Constant for comparisons OR compared objects are equal
C_NotEqual	Constant for comparisons OR compared objects are not equal
C_Found	Constant for search functions OR sought object found
C_NotFound	Constant for search functions OR sought object not found
C_AccessDenied	Constant for access conditions, access denied
C_AccessAllowed	Constant for access conditions, access allowed
C_WriteError	Constant for a data write error (e.g. writing to EEPROM)
C_EFTypeWorking	Constant for a working EF
C_EFTypeInternal	Constant for an internal EF
C_EFStrucLinFix	Constant for a linear fixed EF
C_EFStrucTransp	Constant for a transparent EF
C_CmdVERIFY	Constant for the VERIFY command
C_CmdINTAUTH	Constant for the INTERNAL AUTHENTICATE command

**Table 13.33** Return code constants used in Small-OS

Constant	Value	Meaning
<i>Process completed – warning processing</i>		
C_RC.CounterX	'63Cx'	A counter (usually the PIN retry counter) was incremented and now has the value 'x'
<i>Process aborted – execution error</i>		
C_RC.MemoryFailure	'6581'	EEPROM write error
<i>Process aborted – checking error</i>		
C_RC.WrongLength	'6700'	Invalid length
C_RC.CmdIncompFStruc	'6981'	Command incompatible with file structure
C_RC.SecStateNotSatisfied	'6982'	Security state not satisfied.
C_RC.AuthMethodBlocked	'6983'	Authentication method blocked
C_RC.CondOfUseNotSatisfied	'6985'	Conditions for using data element (PIN or key) not satisfied
C_RC.CmdNotAllowed	'6986'	Command not allowed
C_RC.FktNotSupported	'6A81'	Function not supported
C_RC.FileNotFound	'6A82'	File not found
C_RC.RecordNotFound	'6A83'	Record not found
C_RC.LcInconsistentP1P2	'6A87'	L <sub>c</sub> inconsistent with P1 or P2
C_RC.RefDataNotFound	'6A88'	Data referenced in command not found
C_RC.WrongP1P2	'6B00'	P1 or P2 invalid
C_RC.INSNotSupported	'6D00'	Command not supported
C_RC.CLANotSupported	'6E00'	Class not supported
C_RC.FatalError	'6F00'	Return code constant; internal error (no further description)
<i>Process completed – normal processing</i>		
C_RC.OK	'9000'	Command successfully executed

**Table 13.34** Variables used in Small-OS for data transmission to and from the smart card. These variables are typically held in RAM

Variable	Meaning
.1 . . . 8	Bitwise reading or writing of a variable (for example, APDU.Cmd.CLA.1 corresponds to bit 1 of the class byte in the command APDU))
APDU.Cmd	Command APDU received from the smart card
APDU.Cmd.CLA	Class byte of the command APDU
APDU.Cmd.INS	Command byte of the command APDU
APDU.Cmd.P1	Parameter 1 of the command APDU
APDU.Cmd.P2	Parameter 2 of the command APDU
APDU.Cmd.Lc	Length of the command byte of the command APDU (optional)
APDU.Cmd.Data [ . . . ]	Data part of the command APDU with length 1 . . . n (optional)
APDU.Cmd.Le	Expected length of the response APDU (optional)
APDU.Rsp	Response APDU to be sent by the smart card
APDU.Rsp.Data [ . . . ]	Data part of the response APDU with length 1 . . . n (optional)
APDU.Rsp.SW1	Status word 1 of the response APDU (byte 1 of the return code)
APDU.Rsp.SW2	Status word 2 of the response APDU (byte 2 of the return code)
Return code	Return code := APDU.Rsp.SW1    APDU.Rsp.SW2

**Table 13.35** Variables used in Small-OS for file management and file access. These variables are typically held in RAM

Variable	Meaning
.1 . . . 8	Bitwise reading or writing of a variable (for example, APDU.Cmd.CLA.1 corresponds to bit 1 of the class byte in the command APDU))
Ptr.MF	Pointer in the smart card operating system; always points to the MF
Ptr.CurrentDF	Pointer in the smart card operating system; points to the currently selected DF or to the MF
Ptr.CurrentIEF.Key	Pointer in the smart card operating system; points to the current internal EF with keys (EF Key) in the currently selected DF
Ptr.CurrentWEF	Pointer in the smart card operating system; points to the currently selected working EF in the currently selected DF or the MF.
Ptr.CurrentRecord	Pointer in the smart card operating system; points to the current record in a record-oriented EF. This pointer is invalid if a transparent file is selected.
SecurityState.MF	Attained security state of the MF
SecurityState.DF	Attained security state of the currently selected DF

### 13.17.6.1 Small-OS RAM variables

Most of the RAM variables occupy the region used for the transmit and receive buffers. All data elements of an APDU can be addressed by their specific variables. In order to manage with the small amount of available RAM, partially overlapping transmit and receive buffers are

sometimes used in practice. In this case, some commands must learn from operating system exception handling that data can be written to the transmit buffer only after all data in the receive buffer has been processed. To foster ease of understanding, memory minimization is not used here, which means that the transmit and receive buffers are fully separate.

The second large group of RAM variables are used for managing the file structure. This includes several pointers that reference the current directory (.... .Ptr.CurrentDF), the current file (.... .Ptr.CurrentWEF), and the currently valid key file (.... .Ptr.CurrentIEF.Key). With record-oriented EFs (linear fixed EFs), the currently selected record is also assigned a pointer (.... .Ptr.CurrentRecord). All pointers are identified by the prefix ‘Ptr’, and they are explicitly set to ‘C\_InvalidPointer’ when they cannot be used.

The two variables SecurityState.MF and SecurityState.DF identify the security state of the MF and the currently selected DF, respectively. The security states attained by the individual files are stored in these variables as positive integers. Here the value ‘0’ indicates that no security state has been attained. This value is set automatically when Small-OS is initialized after a reset.

Additional RAM memory is needed for the program stack, the DES cryptographic algorithm, and the working registers. Here this is assumed to be implicit, so no specific variables are assigned for this purpose.

### 13.17.6.2 Small-OS EEPROM variables

The data structures used for file management in Small-OS are described in Tables 13.36, 13.37 and 13.38. For the sake of simplicity, the file structure in the smart card is implemented as a multidimensional field. This would be much too memory-intensive for implementation in a real smart card operating system. One-way linked lists are often used as basic data structures

**Table 13.36** General data structures used for file management in Small-OS. These variables are typically held in EEPROM

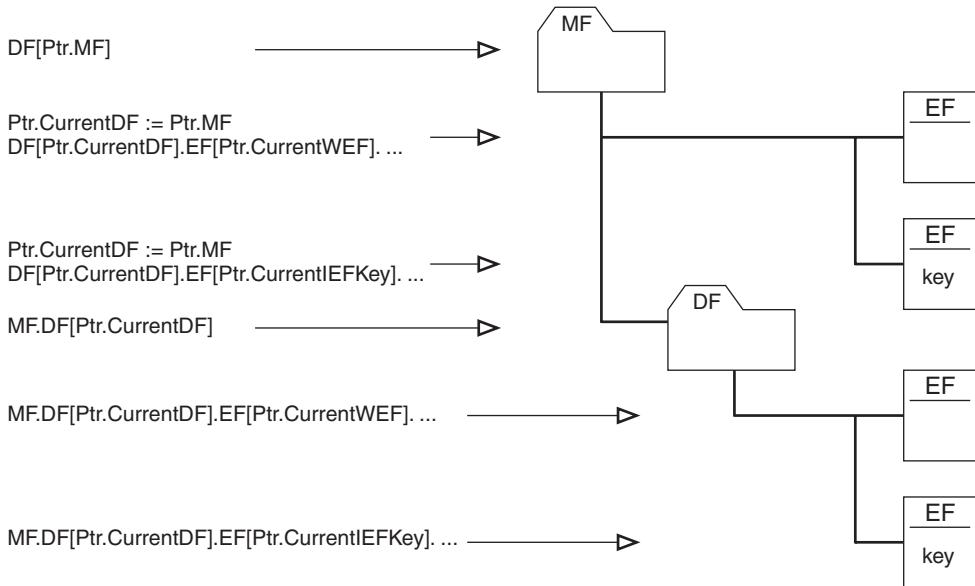
Variable	Description, contents and size
<i>Data structures for the MF and the DFs</i>	
DF[...].FID	File identifier length: 2 bytes; content: 0 ... 255 each element
DF[...].DFName	DF name (includes the application identifier) length: 1 ... 16 bytes; content: 0 ... 255 each element
DF[...].LenDFName	DF name length (length: 1 byte; content: 1 ... 16)
<i>Data structures for EFs</i>	
DF[...].EF[...]....	File tree in the form of a two-dimensional matrix
DF[...].EF[...].FID	File identifier; length 2 bytes; content 0 ... 255 each element
DF[...].EF[...].Structure	EF structure; content: attribute (transparent / linear fixed)
DF[...].EF[...].Type	EF type; content: attribute (working / internal)
DF[...].EF[...].AccessCondition.Read	Condition for READ access to the EF; content 0 ... 255
DF[...].EF[...].AccessCondition.Update	Condition for UPDATE access to the EF; content 0 ... 255

**Table 13.37** Data structures used in Small-OS for managing internal EFs with linear variable structure used to store PINs and keys. These variables are typically held in EEPROM

Variable	Description, contents and size
<i>EF data structures for storing keys</i>	
DF[...].EF[...].Record[x].KeyData	PIN or key in a linear variable file length: $n$ ( $= \dots.KeySize$ ); content: 1 ... 255 each element
DF[...].EF[...].Record[x].KeySize	PIN or key length in bytes
DF[...].EF[...].Record[x].KeyNo	PIN number or key number content with a PIN: 1; 2 content with a key: 1 ... 31 each element
DF[...].EF[...].Record[x].RCntr	Retry counter for a PIN or a key content: 0 ... 15
DF[...].EF[...].Record[x].RCntrMaxValue	Maximum retry counter value for a PIN or key content: 1 ... 15
DF[...].EF[...].Record[x].KeyPurpose	Intended use of EF Key record content (VERIFY or INTERNAL AUTHENTICATE)
DF[...].EF[...].Record[x].EntryState	Necessary state for using a PIN or key content: 1 ... 255
DF[...].EF[...].Record[x].ResultState.OK	State after successful use of a PIN or key content: 1 ... 255
DF[...].EF[...].Record[x].ResultState.NOK	State after unsuccessful use of a PIN or key content: 1 ... 255

**Table 13.38** Data structures used in Small-OS for managing working EFs with transparent and linear fixed structures. These variables are typically held in EEPROM

Variable	Description, contents and size
<i>Data structures for EFs with transparent file structure</i>	
DF[...].EF[...].TransparentData[1 ... n]	Data content of a transparent file length: $n$ ( $= \dots.TransparentContentSize$ ); content: 1 ... 255 each element
DF[...].EF[...].TransparentContentSize	Data size of a transparent file; content: 1 ... 255
<i>Data structures for EFs with linear fixed file structure</i>	
DF[...].EF[...].Record[...].Data[1 ... n]	Data content of an individual record of a linear fixed file; length: 1 ... $n$ ( $n = \dots.Size$ ); content: 0 ... 255 each element
DF[...].EF[...].Record[...].Size	Length of a record in a linear fixed file (same for all records of a linear fixed file); content: 1 ... 255
DF[...].EF[...].NoOfRecords	Number of records in a linear fixed file content: 1 ... 255



**Figure 13.48** Example addresses of directories (MF and DF) and files (EFs) in the data structures available in Small-OS

for file management in real systems. The lengths of the list elements can be made variable by using TLV encoding. This minimizes the amount of memory used for file management, since only the necessary data elements have to be held in memory. Typical file headers of DFs and EFs, with all data necessary for these files, have a size of 20 to 40 bytes. However, the file structure with multidimensional fields used in Small-OS can without question provide a simpler and more efficient solution when the operating system is implemented in a high-level language and runs in a hardware environment with relatively few memory restrictions. A nearly identical structure is used in the Smart Card Simulator program, for example.

The file structure of Small-OS is shown in Figure 13.48. The starting point for file management is always DF[...]. This means that the MF is simply a special form of DF. There is no provision for nested DFs. As a logical consequence, EFs (including their data contents and properties) are addressed via DF[...].EF[...]. There are separate variables for each EF type (working or internal) and structure (transparent, linear fixed, or linear variable). These variables hold the characteristic properties of the various EF types and structures.

### 13.17.7 Main loop and initialization

Tables 13.39 and 13.40 show the pseudocode for the main loop and system initialization. After the smart card is reset, the program counter is loaded with the address of the reset vector and the processor starts to execute the first instruction of the program code. As with PCs, the hardware is checked immediately to verify that it is in good working order. This consists of an initial RAM test followed by the calculation of several checksums for the contents of the ROM and EEPROM. If an error occurs here, the smart card attempts to send a special error

**Table 13.39** Small-OS program code: Operating system start-up and main loop

---

Reset_vector:	Entry point following a CPU reset.
CALL Initialize_Hardware	
CALL Initialize_Operating_System	
CALL IO_Manager_Send_ATR	
Main_Loop:	Main loop of the smart card operating system
CALL IO_Manager_Receive_APDU	
CALL Command_Interpreter	
CALL IO_Manager_Send_APDU	
Goto Main_Loop	

---

**Table 13.40** Small-OS program code: Microcontroller hardware and operating system initialization

---

Initialize_Hardware:	Subroutine for initializing the smart card hardware
CALL Kernel_CheckRAM	Test the RAM for proper operation
IF STATUS(Kernel_CheckRAM = _	
_C_Error) THEN ( _	
GOTO IO_Manager_Send_Error_ATR)	
CALL Kernel_DeleteRAM	Set the entire RAM to '00' to attain a defined initial state after each reset. An intentional side effect is that all variables in RAM are initialized
IF STATUS(Kernel_DeleteRAM = _	
_C_Error) THEN ( _	
GOTO IO_Manager_Send_Error_ATR)	
CALL Kernel_Check_EDC_ROM	Check error detection codes (EDCs) at various places in ROM
IF STATUS(Kernel_Check_EDC_ROM = _	
_C_Error) THEN ( _	
GOTO IO_Manager_Send_Error_ATR)	
CALL Kernel_Check_EDC_EEPROM	Check error detection codes (EDCs) at various places in EEPROM
IF STATUS(Kernel_Check_EDC_EEPROM = _	
_C_Error) THEN ( _	
GOTO IO_Manager_Send_Error_ATR)	
RETURN	
Initialize_Operating_System:	Subroutine for initializing the smart card operating system
CALL IO_Manager_Set_Transmission_Parameter	
CALL SELECT_FILE_MF	
IF STATUS(SELECT_FILE_MF = _	
_C_Error) THEN ( _	
GOTO IO_Manager_Send_Error_ATR)	
SecurityState.MF := 0	
SecurityState.DF := 0	
RETURN	

---

ATR, after which it enters an endless loop where it only waits for the next reset. The success of the attempt to send this special error ATR depends primarily on the nature of the error that occurred. If the error is serious and involves components such as the RAM or code such as the program code of the transmission routine, only a garbled ATR or no ATR at all can be sent.

After the chip hardware has been checked and initialized, the initialization of the operating system starts. The essential elements of this are setting the data transmission parameters for the T = 1 protocol, automatically selecting the MF, and establishing the security conditions for file access. Failure to find the MF is a serious error, and if this happens Small-OS terminates all further actions after sending an error ATR.

### 13.17.8 I/O manager

All serial data transfers from and to the terminal are handled by the I/O manager, whose pseudocode is shown in Table 13.41. Two hardware-dependent program routines in the kernel of the operating system called Kernel.IO\_SendByte and Kernel.IO\_ReceiveByte form the basis for transmitting and receiving messages. All other parts of the I/O program code of Small-OS are independent of the target hardware. ATR generation and processing of the T = 1 transmission protocol are performed entirely in the I/O manager for both successful and unsuccessful command execution. Particularly due to the complicated processes of the T = 1 protocol, this requires the implementation of a dedicated state machine. The ISO/IEC 7816-3 Amd. 1 standard describes the responses of the I/O manager in case of success or failure by means of many examples. When programmed in manually optimized assembly language, a good I/O manager needs at least one kilobyte of memory, usually in ROM.

Real I/O managers preferably use only RAM for the I/O buffer because it can be accessed for write operations at the full speed of the CPU. However, there are now many smart card operating systems whose transmit and receive buffers are larger than the available RAM size. In this case, a certain part of the EEPROM is used as an extension to the data transmission buffer when the data volume exceeds a certain threshold. This avoids imposing restrictions on the size of the I/O buffer, but the price for this is a considerably reduced data transmission rate due to the EEPROM write time. In addition, with this sort of extended I/O buffer there is a risk that the buffer may reach the end of its useful life relatively quickly due to frequent write accesses to the EEPROM. Despite these limitations, this is the only technically feasible way to make the data transmission buffer larger than the amount of available RAM.

### 13.17.9 File manager

In real smart card operating systems, for security reasons all file accesses are handled by a central file manager. In many cases, it also computes the checksum of the file header when a file is accessed. In a concession to simplicity, Small-OS accesses the multidimensional variables of the file structure directly during command processing, without using separate layers or prior checksum calculation. The only test performed by the file manager is to check whether access to the file is allowed; this is done for all commands. Read and write access to files would be handled in a similar manner with a complete file manager. Table 13.42 shows the pseudocode of the file manager and return code manager.

### 13.17.10 Return code manager

The task of the return code manager is to use a return code parameter value to look up the associated return code in a table and then append it to any data block present in the transmit

**Table 13.41** Small-OS program code: Routines for data transmission to and from the smart card

<code>IO_Manager</code>	Hardware-level subroutines for exchanging data with the terminal via the serial I/O line
<code>IO_Manager.Send_Error_ATR:</code>	Subroutine for sending a special ATR indicating a serious error in the smart card hardware or the operating system.
<code>...</code>	
<code>Error_ATR:</code>	Following the ATR, the operating system is suspended in an endless loop
<code>GOTO Error_ATR</code>	
<code>IO_Manager.Send_ATR:</code>	Subroutine that codes all the parameters of the transmission protocol in an ATR string and transmits the string. Uses the operating system kernel routine
<code>...</code>	
<code>RETURN</code>	<code>Kernel_IO_SendByte</code>
<code>IO_Manager.Send_APDU:</code>	Subroutine that converts an APDU already present in the I/O buffer into a TPDU according to the selected transmission parameters and then sends it to the terminal via the serial I/O line. Mechanisms for correcting transmission errors are used as necessary during data transmission. Uses the operating system kernel routine
<code>...</code>	
<code>RETURN</code>	<code>Kernel_IO_SendByte</code>
<code>IO_Manager.Receive_APDU:</code>	Subroutine that receives a TPDU from the terminal via the serial I/O line using the selected transmission parameters, converts it into an APDU, and stores the APDU in the I/O buffer. Mechanisms for correcting transmission errors are used as necessary during data transmission. Uses the operating system core routine
<code>...</code>	
<code>APDU.Cmd := APDU received from .</code>	
<code>the terminal and converted</code>	
<code>...</code>	
<code>RETURN</code>	<code>Kernel_IO_Receive Byte</code>
<code>IO_Manager_Set_Transmission_Parameter:</code>	Subroutine that sets the parameters for serial data transmission
<code>...</code>	
<code>set clock rate conversion factor to 372</code>	
<code>set convention</code>	
<code>set transmit buffer size</code>	
<code>set receive buffer size</code>	
<code>...</code>	
<code>RETURN</code>	

buffer. This is normally performed by a dedicated manager for two main reasons. First, it is good programming style to handle all return code generation in a central location. For example, if a return code must be changed due to a programming error, this can be done easily in a single central location. This is by no means an infrequent occurrence. Second, a return code manager saves precious memory. Clearly, less memory is required to store all the return codes together in a single table than individually in every routine or location where they are needed.

**Table 13.42** Small-OS program code: Return code manager and file manager

Return_Code_Manager:	Manager for setting return code Set the return code based on the supplied value and a return code table.
...	
RETURN	
File_Manager_CheckACRead:	File manager: test the ‘read’ file access condition Set the access status variable to its initial value Check whether the security state required for reading the selected EF with READ BINARY or READ RECORD has been attained. The required security state depends on the currently selected directory (MF or DF).
Status := C_AccessDenied IF Ptr.CurrentDF = Ptr.MF THEN ( // MF is selected IF .EF[Ptr.CurrentWEF].AccessCondition. Read = _ SecurityState.MF THEN ( _ Status := C_AccessAllowed) ELSE ( // DF is selected IF .EF[Ptr.CurrentWEF].AccessCondition. Read = _ SecurityState.DF THEN ( _ Status := C_AccessAllowed) RETURN	
File_Manager_CheckACUpdate:	File manager for checking the ‘update’ file access condition Set the access status variable to its initial value Check whether the security state required for writing the selected EF with UPDATE BINARY or UPDATE RECORD has been attained. The required security state depends on the currently selected directory (MF or DF)
Status := C_AccessDenied IF Ptr.CurrentDF = Ptr.MF THEN ( // MF is selected IF .EF[Ptr.CurrentWEF].AccessCondition. Update = _ SecurityState.MF THEN ( _ Status := C_AccessAllowed) ELSE ( // DF is selected IF .EF[Ptr.CurrentWEF].AccessCondition. Update = _ SecurityState.DF THEN ( _ Status := C_AccessAllowed) RETURN	

### 13.17.11 Operating system kernel

All routines that belong to the kernel of the operating system are combined in the ‘OS kernel’ section (see Table 13.43). Most of these routines are either hardware-dependent or rather time-critical, so they must be adapted when the program is ported to a different hardware platform. The functions provided by these routines are described by their individual names. Some of these subroutines affect security interests. For example, the execution time of a subroutine such as Kernel\_CompareByteString should be constant regardless of the result of the comparison,

**Table 13.43** Small-OS program code: Hardware-dependent routines of the operating system kernel

Kernel_Check_EDC_x:	Hardware-level subroutine of the operating system kernel
...	
RETURN	
Kernel_x:	Hardware-level subroutine of the operating system kernel
...	
RETURN	
Kernel_DES_x:	OS kernel subroutine for DES encryption and decryption encrypt/decrypt $x$ times
...	
RETURN	
Kernel_IO_x:	Subroutine for sending and receiving a single byte via the serial interface of the microcontroller SendByte/ReceiveByte $x$ times
...	
RETURN	

since any variation could be used as the basis for determining the internal results of PIN computations by measuring the processing times. Although this type of attack has now been eliminated by incrementing the retry counter before each PIN comparison as a precautionary measure, until recently it was a very promising form of attack on PINs.

### 13.17.12 Command interpreter

The Small-OS command interpreter has a relatively simple structure, as can be seen from the pseudocode in Table 13.44. It acts as a dispatcher that uses the class and instruction bytes of recognized commands to call specific routines that process these commands. This implementation uses little memory and has the additional important advantage that it is relatively easy to integrate new commands in the operating system. This can be done by simply adding a few lines of code to the command interpreter to identify the new command and the associated subroutine call. With this code in place, the new command can be recognized and executed as necessary.

The structures of command interpreters commonly used in practice are far more complicated. This is partly because they are located in ROM, and partly because it must be possible to download additional program code into the EEPROM when the operating system is completed. It must be possible to recognize and execute this downloaded code at runtime. The principle that is used for this is a jump table located in the EEPROM, which is extended as necessary when the card is completed.

Hard-coded searching for a specific class byte as implemented in the Small-OS command interpreter is only reasonable in operating systems that do not support secure messaging or logical channels and that support only one ISO/IEC 7816-4 command class. In all other cases, the class byte is used to identify previously defined options and is not necessarily the same for all commands.

**Table 13.44** Small-OS program code: Command interpreter

Command.Interpreter:	Command interpreter and dispatcher to program code for the individual commands
IF APDU.Cmd.CLA <> '00' THEN ( Return code := C_RC.CLANotSupported GOTO Command.Interpreter.Exit)	If the class byte does not match class for ISO/IEC 7816-4 commands, abort command processing and set the corresponding return code
IF APDU.Cmd.INS = 'A4' THEN ( CALL SELECT.FILE GOTO Command.Interpreter.Exit)	If the SELECT FILE command was sent to the smart card, call the corresponding subroutine
IF APDU.Cmd.INS = 'B0' THEN ( CALL READ_BINARY GOTO Command.Interpreter.Exit)	If the READ BINARY command was sent to the smart card, call the corresponding subroutine
IF APDU.Cmd.INS = 'D6' THEN ( CALL UPDATE_BINARY GOTO Command.Interpreter.Exit)	If the UPDATE BINARY command was sent to the smart card, call the corresponding subroutine
IF APDU.Cmd.INS = 'B2' THEN ( CALL READ_RECORD GOTO Command.Interpreter.Exit)	If the READ RECORD command was sent to the smart card, call the corresponding subroutine
IF APDU.Cmd.INS = 'DC' THEN ( CALL UPDATE_RECORD GOTO Command.Interpreter.Exit)	If the UPDATE RECORD command was sent to the smart card, call the corresponding subroutine
IF APDU.Cmd.INS = '20' THEN ( CALL VERIFY GOTO Command.Interpreter.Exit)	If the VERIFY command was sent to the smart card, call the corresponding subroutine
IF APDU.Cmd.INS = '88' THEN ( CALL INTERNAL.AUTHENTICATE GOTO Command.Interpreter.Exit)	If the INTERNAL AUTHENTICATE command was sent to the smart card, call the corresponding subroutine
Return code := C_RC.INSNotSupported	The command sent to the smart card is not supported
Command.Interpreter.Exit: CALL Return.Code.Manager RETURN	Place the return code determined by the program code for the command in the buffer and exit the command interpreter and dispatcher

### 13.17.13 Structure of program code for commands

The basic functions of smart card commands are described in Chapter 11 on page 353. The program code for command execution shown here is always divided into three parts separated by thin lines. Subroutines belonging to the command are separated from the actual program code by thick lines. In the first functional block, the command header (CLA, INS, P1 and P2) is analyzed as much as possible at this point in the process. This follows the well-known software design principle that data consistency and value ranges should be checked as early as possible.

Following this, the basic prerequisites for executing the command are checked. For example, monitoring of the file structure or the access conditions occurs in this section. If all these checks are completed without any rejections, the actual execution of the command takes place. This

usually involves only a very small amount of code. Finally, the return code for successful execution of the command is set and a jump to the I/O manager is made via the command interpreter and the return code manager. The I/O manager sends the result of the command processing and then waits for a new command.

If an error occurs in any of the checks in the pseudocode, the relevant return code is immediately set and program execution branches via the main exit point of the subroutine concerned.

### 13.17.14 Command set

The structure and coding of all seven commands listed below comply with the ISO/IEC 7816-4 standard. Due to the freedom of implementation that a standard naturally allows, it is nevertheless necessary to specify the details of each command in addition to referring to the standard. The specifications of all the commands supported by Small-OS are thus described in the following subsections. The specifications have been kept relatively short and formal, so they represent only the essential elements.

#### 13.17.14.1 *SELECT command*

The SELECT command is used to select a file (MF, DF or EF). This is usually done with the two-byte file identifier (FID). The ISO/IEC 7816-4 standard specifies that DFs can also be selected using a DF name, which can contain an application identifier (AID). In Small-OS, the AID can be supplied only in its complete form. In the special case of the MF, no FID is necessary for selection because a suitable command option can be used instead. The MF is also automatically selected after the smart card is reset, and it can be selected from every other directory during a session.

SELECT falls under Case 1 when the MF is selected directly, which means that neither the command APDU nor the response APDU has a data portion. When selection is made using an FID or a DF name, SELECT falls under Case 3, which means that a data portion is present in the command APDU, but not in the response APDU.

The security state of the MF is reset to the initial state (0) after a reset. Selecting a DF does not affect the security state of the MF, but the security state of the DF is automatically set to the initial state (0) when it is selected. When a linear fixed EF is selected, the record pointer is set to ‘invalid’.

The search order, which means whether the operating system first searches for a DF or an EF when selection is made using a FID, is not specified in the standard. However, this is of considerable significance in certain cases. For example, if the MF is currently selected and there is a DF as well as an EF with the same FID, it may not be possible to select the EF, depending on the search routine. Consequently, the search routine of Small-OS is implemented such that with FID selection the list of EFs is always searched first, followed by the list of DFs. In case of a conflict, the DF can always be selected using its DF name, so using the FID is not absolutely necessary in such cases.

The coding of the SELECT command is described in Tables 13.45 to 13.47, the coding of the response is described in Table 13.48, and the pseudocode is listed in Table 13.49.

**Table 13.45** Small-OS: Coding of the Case 1 command SELECT with the ‘direct MF selection’ option

Data element	Coding	Remarks
CLA	‘00’	—
INS	‘A4’	—
P1	‘00’	—
P2	‘00’	—

**Table 13.46** Small-OS: Coding of the Case 1 command SELECT with the ‘file selection using an FID’ option

Data element	Coding	Remarks
CLA	‘00’	—
INS	‘A4’	—
P1	‘00’	—
P2	‘00’	—
L <sub>c</sub>	2	—
DATA	FID	The two-byte FID of an MF, DF or EF

**Table 13.47** Small-OS: Coding of the Case 1 command SELECT with the ‘DF selection using a DF name’ option

Data element	Coding	Remarks
CLA	‘00’	—
INS	‘A4’	—
P1	‘04’	—
P2	‘00’	—
L <sub>c</sub>	1 ... 16	—
DATA	DF name	A DF name (1 to 16 bytes), which may contain the AID of the file to be selected

**Table 13.48** Small-OS: Coding of the response to the SELECT FILE command

Data element	Coding	Remarks
SW1    SW2	‘9000’	Return code in case of correct command execution

**Table 13.49** Small-OS program code: SELECT command in compliance with ISO 7816-4

SELECT_FILE:	Command according to ISO/IEC 7816-4 with extensions
IF APDU.Cmd.P1 = '00' THEN (	If P1 = '00', either the command option for the direct MF selection is set or a file is being selected using its two-byte FID.
IF ((APDU.Cmd.P2 = '00') AND (Length(APDU.Cmd) = 4)) THEN (	If P2 = '00' and only the four-byte command header was sent, the MF should be selected directly. Check whether the command was sent as a Case 1 command. If it was not, set the appropriate return code and abort command processing
IF LENGTH (APDU.Cmd) < 4 THEN (	
Return code := C_RC_WrongLength	
RETURN)	
CALL SELECT_FILE_MF	
RETURN)	
IF APDU.Cmd.Lc = '02' THEN (	If two data bytes were sent, a file should be selected using the two-byte FID. Check whether the command was sent as a Case 3 command.
IF LENGTH (APDU.Cmd) < 6 THEN (	
Return code := C_RC_WrongLength	
RETURN)	
IF APDU.Cmd.Data[1 ... 2] = '3F00' THEN (	If it was not, set the appropriate return code and abort command processing. In case of MF selection (with FID '3F00'), perform the selection immediately. With any other file (DF or EF), perform the selection using a search routine with the given FID as the search string
CALL SELECT_FILE_MF	
RETURN)	
ELSE (CALL SELECT_FILE.FID	
RETURN) ) )	
IF APDU.Cmd.P1 = 0000 0100 THEN (	Check whether a DF should be selected using the DF name. Check whether the command was sent as a Case 3 command. If it was not, set the appropriate return code and abort command processing. Check whether the length of the associated data portion is between 1 and 16 bytes.
IF LENGTH (APDU.Cmd) < 5 THEN (	
Return code := C_RC_WrongLength	
RETURN)	
IF ((APDU.Cmd.Lc < 1) OR (APDU.Cmd.Lc > 16)) THEN (	
Return code := C_RC_LcInconsistentP1P2	
RETURN)	
CALL SELECT_FILE_DFName)	
RETURN	
SELECT_FILE_MF:	Subroutine for MF selection.
SEARCH (MF in file tree)	If the MF cannot be found in the file tree, abort with an internal operating system error
IF STATUS(SEARCH = C_NotFound) THEN (	
// MF not found in file tree	
Return code := C_RC_FatalError	
RETURN)	
// MF found in file tree	Set the current DF pointer to the address of the found MF
Ptr.CurrentDF := located MF address	
SEARCH (EF Key in MF)	Search for an EF Key file and set the appropriate pointer if it is present
IF STATUS(SEARCH) = C_Found THEN (	
Ptr.CurrentIEF.Key := - located EF Key address)	

(Continued)

**Table 13.49** (continuation): Small-OS program code: SELECT command

ELSE (Ptr.CurrentIEF.Key := C_InvalidPointer) Ptr.CurrentWEF := C_InvalidPointer Ptr.CurrentRecord := C_InvalidPointer SecurityState.MF := 0 SecurityState.DF := 0 Return code := C_RC.OK RETURN	Mark the current EF pointer and the current record pointer as invalid. Set the security states of the MF and DF to the initial value
SELECT_FILE_FID: SEARCH (EF with FID in the currently selected DF) IF STATUS(SEARCH) = C_Found THEN ( Ptr.CurrentWEF := located EF address Ptr.CurrentRecord := C_InvalidPointer Return code := C_RC.OK RETURN) // EF with the given FID not found SEARCH ((DF with FID)) IF STATUS(SEARCH) = C_Found THEN ( Ptr.CurrentDF := located DF address SEARCH (EF Key in current DF) IF STATUS(SEARCH) = C_Found THEN ( Ptr.CurrentIEF.Key := located EF Key address) ELSE (Ptr.CurrentIEF.Key := C_InvalidPointer) Ptr.CurrentWEF := C_InvalidPointer Ptr.CurrentRecord := C_InvalidPointer SecurityState.DF := 0 Return code := C_RC.OK RETURN)) // unable to find an EF or DF with the given FID Return code := C_RC.FileNotFound RETURN	Subroutine for selecting a DF or EF using an FID Search the current DF for an EF with the given FID. If an EF can be found, set the current EF pointer.  If the EF cannot be found, search for a DF with the given FID. If a DF can be found, set the current DF pointer to point to it. Then search for an EF Key file and set the appropriate pointer if a file can be found
SELECT_FILE_DFNName: SEARCH (DF with DF name) IF STATUS(SEARCH) = C_NotFound THEN ( Return code := C_RC_FileNotFound RETURN)	Mark the current EF pointer and the current record pointer as invalid. Set the security state of the DF to the initial value  If no matching file was found, set the appropriate return code
	Subroutine for selecting a DF using a DF name Set the current DF pointer to the address of the DF found in the file tree. Then search for the relevant EF Key in the current DF, and set the corresponding pointer to its address if it is found

**Table 13.49** (*continuation*): Small-OS program code: SELECT command

---

```

ELSE (Ptr.CurrentDF := located -
      DF address)
SEARCH (EF Key in current DF)
IF STATUS(SEARCH) = C_Found THEN (
  Ptr.CurrentIEF.Key := located -
  EF Key address)
ELSE (Ptr.CurrentIEF.Key := -
      C_InvalidPointer)
Ptr.CurrentWEF := C_InvalidPointer
Ptr.CurrentRecord :=
C_InvalidPointer
SecurityState.DF := 0
Return code := C_RC_OK
RETURN

```

Mark the current EF pointer and the current record pointer as invalid. Set the security state of the DF to the initial value

---

### 13.17.14.2 READ BINARY command

The READ BINARY command can be used to read data from a transparent EF, starting at a location specified by a 15-bit offset parameter supplied with the command. READ BINARY falls under Case 2, which means that there is no data portion in the command APDU, but there is a data portion in the response APDU.

All length values must be an integer number of bytes. The maximum amount data to be read is limited to the maximum data volume of a transparent EF, which is 255 bytes in Small-OS. If a value of zero is given for the length, all data from the given offset location to the end of the file is read. Implicit selection of EFs using a short FID is not implemented here for the sake of simplicity.

Before data can be read from an EF using this command, the associated access conditions must be satisfied. Otherwise, the command will be rejected with a suitable error message.

The coding of the READ BINARY command is described in Table 13.50, the coding of the response is described in Table 13.51, and the pseudocode is listed in Table 13.52.

**Table 13.50** Small-OS: Coding of the Case 2 command READ BINARY

---

Data element	Coding	Remarks
CLA	'00'	—
INS	'B0'	—
P1	0XXX XXXX	The seven most significant bits of the offset to the data to be read (offset := XXXXXX    Y)
P2	Y	The eight least significant bits of the offset to the data to be read
L_e	Z	Z = 0: read all bytes to the end of the file Z > 0: Z is the number of bytes to be read

---

**Table 13.51** Small-OS: Coding of the response to the READ BINARY command

Data element	Coding	Remarks
DATA	...	If the command was executed correctly, the data requested by the command is located in this data element
SW1    SW2	'9000'	Return code in case of correct command execution

**Table 13.52** Small-OS program code: READ BINARY command in compliance with ISO 7816-4

```

READ_BINARY:
  IF LENGTH (APDU.Cmd) < 5 THEN (
    Return code := C_RC_WrongLength
    RETURN)

  IF APDU.Cmd.P1.b8 = 1 THEN (
    Return code := C_RC_FktNotSupported
    GOTO READ_BINARY_Exit)
  IF Ptr.CurrentWEF = C_InvalidPointer -
  THEN (Return code := C_RC_CmdNotAllowed
    GOTO READ_BINARY_Exit)
  WITH DF[Ptr.CurrentDF]..

  IF .EF[Ptr.CurrentWEF].Structure =
  C_EFStrucTransparent THEN (
    Return code := C_RC_CmdIncompFStruc
    GOTO READ_BINARY_Exit)

  FileMode := (APDU.Cmd.P1 * 256) -
  + APDU.Cmd.P2
  DataLenToRead := 0
  IF Status(File_Manager.CheckACRead) = -
  C_AccessDenied) THEN (
    Return code :=
    C_RC_SecStateNotSatisfied
    GOTO READ_BINARY_Exit)
  IF APDU.Cmd.Le = '00' THEN (
    DataLenToRead :=
    .EF[Ptr.CurrentWEF].-
    TransparentDataSize - FileMode)
  ELSE (
    DataLenToRead := APDU.Cmd.Le)
  IF .EF[Ptr.CurrentWEF].-
  TransparentDataSize -
  < FileMode
  THEN (
    Return code := C_RC_WrongP1P2
    GOTO READ_BINARY_Exit)

```

Command according to ISO/IEC 7816-4  
Check whether the command was sent as a Case 2 command. If it was not, set the appropriate return code and abort command processing  
Check whether an EF should be selected using a short FID  
Check whether an EF is currently selected. Abort the command if no EF is selected  
Set a portion of the file tree as a reference for this command  
Check whether the selected EF has a transparent file structure  
Calculate the offset to the desired data in the file and initialize the variable for the amount of data to be read  
Check whether the required security state for reading data from the selected EF with READ BINARY has been attained  
Check whether all available data should be read (i.e. L.e = '00') or only a certain amount of data  
Check whether the requested offset fits with the file size

**Table 13.52** (*continuation*): Small-OS program code: READ BINARY command

---

IF .EF[Ptr.CurrentWEF].TransparentDataSize < (FileOffset + DataLenToRead) THEN (	Check whether the selected offset and requested data length (L_e) fit with the file size
Return code := C_RC_WrongP1P2	
GOTO READ_BINARY_Exit)	
CALL Kernel_CopyByteString //From .EF[Ptr.CurrentWEF].TransparentData[x ... y]	Copy the requested data from the file to the I/O transmit buffer
//x = FileOffset; y = (FileOffset+DataLenToRead)	
//to APDU.Rsp.Data[1 ... DataLenToRead]	
Return code := C_RC_OK	The command has been processed with no errors, as otherwise an error exit would have occurred
READ_BINARY_Exit:	
END WITH	
RETURN	

---

### 13.17.14.3 UPDATE BINARY command

The UPDATE BINARY command can be used to write data to a transparent EF, starting at a location specified by a 15-bit offset parameter supplied with the command. UPDATE BINARY falls under Case 3, which means that there is a data portion in the command APDU, but not in the response APDU.

All length values must be an integer number of bytes. The maximum length of the data to be read is limited to the maximum data volume of a transparent EF, which is 255 bytes in Small-OS. Implicit selection of EFs using a short FID is not implemented here for the sake of simplicity.

The coding of the UPDATE BINARY command is described in Table 13.53, the coding of the response is described in Table 13.54, and the pseudocode is listed in Table 13.55.

**Table 13.53** Small-OS: Coding of the Case 3 command UPDATE BINARY

---

Data element	Coding	Remarks
CLA	'00'	—
INS	'D6'	—
P1	0XXX XXXX	The seven most significant bits of the offset to the data to be read (offset := XXX XXXX    Y)
P2	Y	The eight least significant bits of the offset to the data to be read
L <sub>c</sub>	...	Number of bytes to be written
DATA	...	The data bytes to be written

---

**Table 13.54** Small-OS: Coding of the response to the UPDATE BINARY command

Data element	Coding	Remarks
SW1    SW2	'9000'	Return code in case of correct command execution

**Table 13.55** Small-OS program code: UPDATE BINARY command in compliance with ISO 7816-4

UPDATE.BINARY:	Command according to ISO/IEC 7816-4
IF LENGTH (APDU.Cmd) < 6 THEN (	Check whether the command was sent as a
Return code := C_RC_WrongLength	Case 3 command. If it was not, set the
RETURN)	appropriate return code and abort
IF APDU.Cmd.P1.b8 = 1 THEN (	command processing
Return code := C_RC_FktNotSupported	Check whether an EF should be selected
GOTO UPDATE.BINARY.Exit)	using a short FID (i.e. the MSB of P1 is set)
IF Ptr.CurrentWEF = C_InvalidPointer ..	Check whether an EF is currently selected. If
THEN (Return code :=	not, abort the command
C_RC_CmdNotAllowed	
GOTO UPDATE.BINARY.Exit)	
WITH DF[Ptr.CurrentDF].	Set a portion of the file tree as a reference for
	this command
IF .EF[Ptr.CurrentWEF].Structure <> ..	Check whether the selected EF has a
C_EFStrucTransparent THEN (	transparent file structure
Return code := C_RC_CmdIncompFStruc	
GOTO UPDATE.BINARY.Exit)	
FileOffset:=(APDU.Cmd.P1*256)	Calculate the offset to the data in the file
+APDU.Cmd.P2	
IF Status(File_Manager.CheckACUpdate) = ..	Check whether the security state required for
C_AccessDenied) THEN (	writing the selected EF with UPDATE
Return code :=	BINARY has been attained
C_RC_SecStateNotSatisfied	
GOTO UPDATE.BINARY.Exit)	
IF .EF[Ptr.CurrentWEF].	Check whether the selected offset and
TransparentDataSize)< ..	requested data length ( $L_c$ ) fit with the file
(FileOffset + APDU.Cmd.Lc) THEN (	size.
Return code := C_RC_WrongP1P2	
GOTO UPDATE.BINARY.Exit)	
CALL Kernel_CopyByteString	Copy the transferred data from the I/O
// From APDU.Cmd.Data[1 ...	receive buffer to the file. If an error occurs,
APDU.Cmd.Lc]	abort and report the error to the terminal
// To .EF[Ptr.CurrentWEF]. ..	
TransparentData[x ... y]	
// x=FileOffset;	
y=(FileOffset+APDU.Cmd.Lc)	
IF Status(Kernel.CopyByteString)= ..	
C_WriteError THEN (	
Return code := C_RC_MemoryFailure	
GOTO UPDATE.BINARY.Exit)	

**Table 13.55** (*continuation*): Small-OS program code: UPDATE BINARY command

---

<pre>Return code := C_RC_OK</pre> <pre>UPDATE_BINARY_Exit:</pre> <pre>END WITH</pre> <pre>RETURN</pre>	<p>The command has been processed with no errors, as otherwise an error exit would have occurred</p>
--	--

---

Before data can be written to an EF using this command, the associated access conditions must be satisfied. Otherwise, the command will be rejected with a suitable error indication.

#### **13.17.14.4 READ RECORD command**

The READ RECORD command can be used to read a record from a linear fixed EF. The maximum amount of data to be read is limited to the maximum record length of 255 bytes. The length specification must either match the length of the addressed record or be set to zero. With a length of zero, the entire record is automatically read. All length values must be an integer number of bytes. Implicit selection of EFs using a short FID is not implemented here for the sake of simplicity. READ RECORD falls under Case 2, which means that there is no data portion in the command APDU, but there is a data portion in the response APDU.

A record in a linear fixed EF can be addressed in three different ways. The number of the desired record can be supplied directly with the READ RECORD command. If this record is present in the file, its contents are returned in the response; otherwise the response contains a suitable error indication. This type of access does not affect the record pointer, which can only be modified with the command options ‘first’, ‘last’, ‘next’, and ‘previous’. The record pointer is set to ‘invalid’ immediately after an EF is newly selected. If the ‘next’ or ‘previous’ option is selected with the record pointer marked as invalid, the record pointer is automatically set to the first or last record of the file, respectively. This makes it possible to read the records in a file by first selecting the EF and then sending a series of READ RECORD commands with the ‘next’ option, without having to use any other commands. The third type of access uses the ‘current’ option. In this case the record that is indicated by the current record pointer is read. If the record pointer is invalid, the command is aborted with an appropriate error indication.

Before data can be read from an EF using this command, the associated access conditions must be satisfied. Otherwise, the command will be rejected with a suitable error indication. The record returned in the response when the command is successfully executed is not TLV coded, although this is optionally allowed by the ISO/IEC 7816-4 standard.

The coding of the READ RECORD command is described in Table 13.56, the coding of the response is described in Table 13.57, and the pseudocode is listed in Table 13.58.

**Table 13.56** Small-OS: Coding of the Case 2 command READ RECORD

Data element	Coding	Remarks
CLA	'00'	—
INS	'B2'	—
P1	X	$Y = 0000\ 0100$ , $X = 0$ read the current record ( <code>Ptr.CurrentRecord</code> ). $Y = 0000\ 0100$ , $X <> 0$ read record number $X$
P2	Y	$Y = 0000\ 0100$ read the record using the method indicated in P1 $X = 0$ , $Y = 0000\ 0000$ read the first record in the file $X = 0$ , $Y = 0000\ 0001$ read the last record in the file $X = 0$ , $Y = 0000\ 0010$ read the next record in the file $X = 0$ , $Y = 0000\ 0011$ read the previous record in the file
L_e	Z	$Z = 0$ : read all bytes until the end of the record $Z > 0$ : $Z$ is the record length

**Table 13.57** Small-OS: Coding of the response to the READ RECORD command

Data element	Coding	Remarks
DATA	...	If the command was correctly executed, the record requested by the command is located in this data element
SW1    SW2	'9000'	Return code in case of correct command execution

**Table 13.58** Small-OS program code: READ RECORD command in compliance with ISO 7816-4

```

READ_RECORD:
  IF LENGTH (APDU.Cmd) < 5 THEN (
    Return code := C_RC_WrongLength
    RETURN)
  IF APDU.Cmd.P2.b8 ... b4 <> 00000 THEN
    (
    Return code := C_RC_FktNotSupported
    GOTO READ_RECORD_Exit)
  IF Ptr.CurrentWEF = C_InvalidPointer
  THEN (
    Return code := C_RC_CmdNotAllowed
    GOTO READ_RECORD_Exit)
  WITH DF[Ptr.CurrentDF].
  IF .EF[Ptr.CurrentWEF].Structure <> .
  C_EFStrucLinFix THEN (
    Return code := C_RC_CmdIncompFStruc
    GOTO READ_RECORD_Exit)

  Command according to ISO/IEC 7816-4
  Check whether the command was sent as a
  Case 2 command. If it was not, set the
  appropriate return code and abort command
  processing
  Check whether an EF should be selected using
  a short FID
  Check whether an EF is currently selected
  Set a portion of the file tree as a reference for
  this command
  Check whether the selected EF has a linear
  fixed structure

```

**Table 13.58** (*continuation*): Small-OS program code: READ RECORD command

---

IF Status(File_Manager_CheckACRead) = _C_AccessDenied THEN (	Check whether the security state required for reading the selected EF with READ RECORD has been attained
Return code :=	
C_RC_SecStateNotSatisfied	
GOTO READ_RECORD_Exit)	
RecordNoToRead := 0	Initialize the variables for the number of the record to be read and its length
RecordLenToRead := 0	
IF APDU.Cmd.P2.b3 ... b1 = 000 THEN (	If the ‘read first record’ option was selected, set the current record pointer to the first record in the file
Ptr.CurrentRecord := 1	
RecordNoToRead := Ptr.CurrentRecord)	
IF APDU.Cmd.P2.b3 ... b1 = 001 THEN (	If the ‘write last record’ option was selected, set the current record pointer to the last record in the file
Ptr.CurrentRecord :=	
.EF[Ptr.CurrentWEF].	
NoOfRecords	
RecordNoToRead := Ptr.CurrentRecord)	
IF APDU.Cmd.P2.b3 ... b1 = 010 THEN (	If the ‘read next record’ option was selected, set the current record pointer to the next record in the file if it does not already point to this record
IF Ptr.CurrentRecord =	
C_InvalidPointer	
THEN (Ptr.CurrentRecord = 1	
RecordNoToRead := Ptr.CurrentRecord)	
ELSE (	
IF Ptr.CurrentRecord < .EF -	
[Ptr.CurrentWEF].NoOfRecords THEN (	
Ptr.CurrentRecord:=Ptr.CurrentRecord+1	
RecordNoToRead := Ptr.CurrentRecord)	
ELSE (	
Return code = C_RC_RecordNotFound	
GOTO READ_RECORD_Exit))	
IF APDU.Cmd.P2.b3 ... b1 = 011 THEN (	If the ‘read previous record’ option was selected, set the current record pointer to the previous record in the file if it does not already point to this record
IF Ptr.CurrentRecord =	
C_InvalidPointer	
THEN (Ptr.CurrentRecord = .EF -	
[Ptr.CurrentWEF].NoOfRecords	
RecordNoToRead := Ptr.CurrentRecord)	
ELSE (	
IF Ptr.CurrentRecord > 1 THEN (	
Ptr.CurrentRecord:=Ptr.CurrentRecord-1	
RecordNoToRead := Ptr.CurrentRecord)	
ELSE (	
Return code = C_RC_RecordNotFound	
GOTO READ_RECORD_Exit))	

---

(Continued)

**Table 13.58** (*continuation*): Small-OS program code: READ RECORD command

---

```

IF ((APDU.Cmd.P2.b3 ... b1 = 100) AND _ . .
(APDU.Cmd.P1 = 0)) THEN (
  IF Ptr.CurrentRecord <>
    C_InvalidPointer .
  THEN (Return code = C_RC_WrongP1P2
    GOTO READ_RECORD_Exit)
  ELSE (
    RecordNoToRead := Ptr.CurrentRecord))
  IF (APDU.Cmd.P2.b3 ... b1 = 100) AND _ .
  (P1>0) THEN
    IF .EF[Ptr.CurrentWEF].NoOfRecords < _ .
    APDU.Cmd.P1 THEN (
      Return code = C_RC_WrongP1P2
      GOTO READ_RECORD_Exit)
    ELSE (
      RecordNoToRead := APDU.Cmd.P1)
    IF APDU.Cmd.Le = '00' THEN (
      RecordLenToRead := .EF[Ptr.CurrentWEF]. .
      Record[RecordNoToRead].Size)
    ELSE (
      RecordLenToRead := APDU.Cmd.Le)
    IF RecordLenToRead <> .EF[Ptr.CurrentWEF]. .
    - .
      Record[RecordNoToRead].Size) THEN (
      Return code = C_RC_LcInconsistentP1P2
      GOTO READ_RECORD_Exit)
    CALL Kernel_CopyByteString
    // From .EF[Ptr.CurrentWEF].Record .
    [RecordNoToRead] . .
    Data[1 ... RecordLenToRead]
    // To APDU.Rsp.Data[1 ...
    RecordLenToRead]
  Return code = C_RC_OK

READ_RECORD_Exit:
  END WITH
  RETURN

```

---

#### **13.17.14.5 UPDATE RECORD command**

The UPDATE RECORD command can be used to write a record to a linear fixed EF. The data supplied with the command cannot be TLV coded, although this is allowed as an option by the ISO/IEC 7816-4 standard. The maximum amount of data to be written is limited to

the maximum record length of 255 bytes. The length specification must exactly match the length of the addressed record, and all length specifications must be an integer number of bytes. Implicit selection of EFs using a short FID is not implemented here for the sake of simplicity. UPDATE RECORD falls under Case 3, which means that there is a data portion in the command APDU, but not in the response APDU.

A record in a linear fixed EF can be addressed in three different ways. The number of the desired record can be supplied directly with the UPDATE RECORD command. If this record is present in the file, its contents are returned in the response; otherwise, the response contains a suitable error indication. This type of access does not affect the record pointer, which can only be modified with the command options ‘first’, ‘last’, ‘next’, and ‘previous’. The record pointer is set to ‘invalid’ immediately after an EF is newly selected.

If the ‘next’ or ‘previous’ option is selected with the record pointer marked as invalid, the record pointer is automatically set to the first or last record of the file, respectively. This makes it possible to write all the records in a file by first selecting the EF and then sending a series of UPDATE RECORD commands with the ‘next’ option, without having to use any other commands. The third type of access uses the ‘current’ option. In this case, the record that is indicated by the current record pointer is written. If the record pointer is invalid, the command is aborted with an appropriate error indication.

Before data can be written to an EF using this command, the associated access conditions must be satisfied. Otherwise, the command will be rejected with a suitable error indication.

The coding of the UPDATE RECORD command is described in Table 13.59, the coding of the response is described in Table 13.60, and the pseudocode is listed in Table 13.61.

**Table 13.59** Small-OS: Coding of the Case 3 command UPDATE RECORD

Data element	Coding	Remarks
CLA	‘00’	—
INS	‘DC’	—
P1	X	$Y = 0000\ 0100$ , $X = 0$ write the current record ( $\text{Ptr}.\text{CurrentRecord}$ ). $Y = 0000\ 0100$ , $X \neq 0$ write record number X
P2	Y	$Y = 0000\ 0100$ write the record using the method specified by P1 $X = 0$ , $Y = 0000\ 0000$ write the first record in the file $X = 0$ , $Y = 0000\ 0001$ write the last record in the file $X = 0$ , $Y = 0000\ 0010$ write the next record in the file $X = 0$ , $Y = 0000\ 0011$ write the previous record in the file
$L_c$	...	Number of bytes to be written
DATA	...	The record to be written

**Table 13.60** Small-OS: Coding of the response to the UPDATE RECORD command

Data element	Coding	Remarks
SW1    SW2	‘9000’	Return code in case of correct command execution

**Table 13.61** Small-OS program code: UPDATE RECORD command in compliance with ISO 7816-4

UPDATE_RECORD:	Command according to ISO/IEC 7816-4
IF LENGTH (APDU.Cmd) < 6 THEN (	Check whether the command was sent as a
Return code := C_RC_WrongLength	Case 3 command. If it was not, set the
RETURN)	appropriate return code and abort command
	processing
IF APDU.Cmd.P2.b8 ... b4 <> 00000 THEN (	Check whether an EF should be selected using
	a short FID
Return code := C_RC_FktNotSupported	
GOTO UPDATE_RECORD_Exit)	
IF Ptr.CurrentWEF = C_InvalidPointer	Check whether an EF is currently selected
THEN (	
Return code := C_RC_CmdNotAllowed	
GOTO UPDATE_RECORD_Exit)	
WITH DF[Ptr.CurrentDF].	Set a portion of the file tree as a reference for
	this command
IF .EF[Ptr.CurrentWEF].Structure = ..	Check whether the selected EF has a linear
C_EFStrucLinFix THEN (	fixed structure
Return code := C_RC_CmdIncompFStruc	
GOTO UPDATE_RECORD_Exit)	
IF Status(File_Manager_CheckACUpdate)	
= ..	
C_AccessDenied) THEN (	Check whether the security state required for
Return code :=	writing the selected EF with UPDATE
C_RC_SecStateNotSatisfied	RECORD has been attained
GOTO UPDATE_RECORD_Exit)	
RecordNoToUpdate := 0	Initialize the variable for the number of the
	record to be written
IF APDU.Cmd.P2.b3 ... b1 = 000 THEN (	If the ‘write first record’ option was selected,
Ptr.CurrentRecord := 1	set the current record pointer to the first
RecordNoToUpdate := Ptr.CurrentRecord)	record in the file
IF APDU.Cmd.P2.b3 ... b1 = 001 THEN (	If the ‘write last record’ option was selected,
Ptr.CurrentRecord := .EF ..	set the current record pointer to the last
[Ptr.CurrentWEF].NoOfRecords	record in the file
RecordNoToUpdate := Ptr.CurrentRecord)	
IF APDU.Cmd.P2.b3 ... b1 = 010 THEN (	If the ‘write next record’ option was selected,
IF Ptr.CurrentRecord =	set the current record pointer to the next
C_InvalidPointer ..	record in the file if it does not already point
THEN (Ptr.CurrentRecord := 1	to this record
RecordNoToUpdate := Ptr.CurrentRecord)	
ELSE (	
IF Ptr.CurrentRecord < .EF ..	
[Ptr.CurrentWEF].NoOfRecords THEN (	
Ptr.CurrentRecord:=Ptr.CurrentRecord+1	
RecordNoToUpdate := Ptr.CurrentRecord)	
ELSE (	
Return code = C_RC_RecordNotFound	
GOTO UPDATE_RECORD_Exit)))	
IF APDU.Cmd.P2.b3 ... b1 = 011 THEN (	

**Table 13.61** (*continuation*): Small-OS program code: UPDATE RECORD command

---

<pre> IF Ptr.CurrentRecord = C_InvalidPointer - THEN (Ptr.CurrentRecord = .EF - [Ptr.CurrentWEF].NoOfRecords) RecordNoToUpdate :=  Ptr.CurrentRecord) ELSE ( IF Ptr.CurrentRecord &gt; 1 THEN ( Ptr.CurrentRecord:=Ptr.CurrentRecord- 1) RecordNoToUpdate:=Ptr.CurrentRecord) ELSE ( Return code = C_RC_RecordNotFound GOTO UPDATE_RECORD_Exit))) IF ((APDU.Cmd.P2.b3 ... b1 = 100) AND - (APDU.Cmd.P1 = 0)) THEN ( IF Ptr.CurrentRecord &lt;&gt; C_InvalidPointer - THEN (Return code = C_RC_WrongP1P2 GOTO UPDATE_RECORD_Exit) ELSE ( RecordNoToUpdate :=  Ptr.CurrentRecord) IF (APDU.Cmd.P2.b3 ... b1 = 100) AND - (P1&gt;0) THEN ( IF .EF[Ptr.CurrentWEF].NoOfRecords &lt; - APDU.Cmd.P1 THEN ( Return code = C_RC_WrongP1P2 GOTO UPDATE_RECORD_Exit) </pre>	<p>If the ‘write previous record’ option was selected, set the current record pointer to the previous record in the file if it does not already point to this record</p>
<pre> IF .EF[Ptr.CurrentWEF].Record - [Ptr.CurrentRecord].Size &lt;&gt; - APDU.Cmd.Lc THEN ( Return code := C_RC_LcInconsistentP1P2 GOTO UPDATE_RECORD_Exit) CALL Kernel_CopyByteString // From APDU.Cmd.Data[1 ... APDU.Cmd.Lc] // To .EF[Ptr.CurrentWEF]. - Record[RecordNoToUpdate]. - Data[1 ... APDU.Cmd.Lc] IF Status(Kernel_CopyByteString) = - C_WriteError THEN - (Return code := C_RC_MemoryFailure GOTO UPDATE_RECORD_Exit) Return code := C_RC_OK UPDATE_RECORD_Exit: END WITH RETURN </pre>	<p>If the ‘write current record’ option was selected, check whether the corresponding pointer is valid. If it is, set the internal variable of the command to the current record to be written</p> <p>If the ‘address record directly with P1’ option was selected, check whether the specified record is present in the file</p> <p>Check whether the specified record length (<math>L_c</math>) matches the record size of the file</p> <p>Copy the supplied data from the I/O receive buffer to the file. If an error occurs, abort and report the error to the terminal</p> <p>The command has been processed with no error; an error exit is taken in all other cases</p>

---

### 13.17.14.6 VERIFY command

The VERIFY command is used to compare a confidential item that has been supplied to the smart card, such as a PIN, to a stored reference value. The length of the PIN must be between one and eight bytes. The operating system does not check the coding of the supplied data string. For instance, a four-digit PIN such as 1234 could be coded as two BCD bytes ('12' || '34') or as four ASCII bytes ('1' || '2' || '3' || '4' = '31' || '32' || '33' || '34'). The VERIFY command falls under Class 3, which means that there is a data portion in the command APDU, but not in the response APDU.

At most two PINs (PIN number 1 and PIN number 2) can be addressed. They can be located in either the EF Key of the MF or the EF Key of the currently selected DF. A PIN that is stored in the EF Key of the MF is used as a common PIN for all applications in the smart card. If a PIN is stored in the EF Key of a DF, it can be used only for the application associated with that DF. Such a PIN is thus an application-specific PIN.

Every PIN has a retry counter that is reset to zero when a positive comparison result is obtained and incremented by one when a negative result is obtained. If the state of the retry counter is not zero, the number of PIN attempts still allowed is encoded in SW2. If the retry counter reaches its maximum value, this is indicated by a separate return code.

As Small-OS does not have any command to reset a retry counter, no further PIN comparisons can be made after a retry counter reaches its maximum value. Depending on the application, this could mean that the smart card could no longer be used. The PIN located in the EF Key cannot be altered by the user, although this option is often present in many smart card applications. A specific command would be needed for this (CHANGE REFERENCE DATA as specified in ISO/IEC 7816-8), but this is not defined in the ISO/IEC 7816-4 standard.

The implementation described here has a small peculiarity due to simplification. As you know, the retry counter is located in EEPROM. However, EEPROM write accesses need not always be successful, due to the possibility of a write error. Consequently, it is necessary to check the data after each write operation to verify that it has been written correctly. If the result of the check is negative, a suitable return code is set. The retry counter is altered so frequently in the VERIFY command that this check is not included in the pseudocode, in order to avoid obscuring the basic operations of the code. You should bear this in mind when examining the code.

The VERIFY command is naturally an ideal choice for potential attacks on a PIN. The implementation here is designed to make it impossible to base an attack on an analysis of the timing or the electrical current consumption. The retry counter is always incremented before the received PIN is compared with the reference PIN stored in the EF Key. This ensures that cutting off the power supply to the card immediately following a PIN comparison does not prevent incrementing of the retry counter, which would allow an attacker to perform an unlimited number of PIN comparisons.

The actual EEPROM writing process for incrementing the retry counter is by no means as trivial as you might think. The coding of the retry counter must be structured such that aborting the process during the write operation or during the erase operation that may be necessary before the write operation cannot result in the retry counter being reset to its initial zero value. This means that the code in the operating system must be designed with reference to the minimum-energy state of the EEPROM, which is its secure state. For example, with an EEPROM whose secure state is zero the initial value of the retry counter may not be coded as zero, as otherwise the retry counter could be reset to zero by skilfully interrupting power to the card during the EEPROM write operation. It would then be possible to determine the

PIN relatively quickly by trial and error because the retry counter could not keep track of unsuccessful PIN comparisons. An ideal solution would be to use an atomic operation for writing the retry counter.

The coding of the VERIFY command is described in Table 13.62, the coding of the response is described in Table 13.63, and the pseudocode is listed in Table 13.64.

**Table 13.62** Small-OS: Coding of the Case 3 command VERIFY

Data element	Coding	Remarks
CLA	'00'	—
INS	'20'	—
P1	'00'	—
P2	Y	Y = 100Z ZZZZ Use a reference PIN stored in the EF Key of the currently selected directory (MF or DF) (specific reference data) Z = 0 0001 . . . Z = 0 0010 Number of the referenced PIN (1 or 2)
L <sub>c</sub>	...	Length of the supplied PIN
DATA	...	The supplied PIN

**Table 13.63** Small-OS: Coding of the response to the VERIFY command

Data element	Coding	Remarks
SW1    SW2	'9000'	Return code in case of correct command execution (successful PIN comparison)

**Table 13.64** Small-OS program code: VERIFY command in compliance with ISO 7816-4

VERIFY:	VERIFY command according to ISO 7816-4
IF LENGTH (APDU.Cmd) < 6 THEN (	Check whether the command was sent as a Case 3 command. If it was not, set the appropriate return code and abort command processing
Return code := C_RC_WrongLength	
RETURN)	
IF APDU.Cmd.P1 <> '00' THEN (	Check whether P1 has the allowed value (P1 must be '00')
Return code := C_RC_WrongP1P2	
GOTO VERIFY_Exit)	
IF ((APDU.Cmd.P2 < '01') OR (	Check whether P2 has one of the two allowed values (P2 must be either 1 or 2)
(APDU.Cmd.P2 > '02')) THEN (	
Return code := C_RC_WrongP1P2	
GOTO VERIFY_Exit)	
IF ((APDU.Cmd.Lc <= 1) OR (	Check whether the length of the supplied data (the PIN) lies within the allowed range ( $1 \leq Lc \leq 8$ )
(APDU.Cmd.Lc >= 8)) THEN (	
Return code := C_RC_LcInconsistentP1P2	
GOTO VERIFY_Exit)	

(Continued)

**Table 13.64 (continuation):** Small-OS program code: VERIFY command in compliance with ISO 7816-4

---

IF Ptr.CurrentIEF.Key = C_InvalidPointer THEN (	Check whether an EF Key file is present in the current directory
Return code := C_RC_RefDataNotFound	
GOTO VERIFY_Exit)	
SEARCH (for the PIN with the requested reference number in DF[Ptr.CurrentDF].EF[Ptr.CurrentIEF.Key].)	Search for the requested reference number in EF Key. If a PIN with the specified reference number is found, set the current key pointer to reference it as the current key (PIN)
IF STATUS(SEARCH) = C_Found) THEN (	
set KeyRecord to the record containing the found PIN	
WITH DF[Ptr.CurrentDF].EF[Ptr.CurrentIEF.Key].)	
ELSE (	
Return code := C_RC_RefDataNotFound	
GOTO VERIFY_Exit)	
IF .Record[KeyRecord].KeyPurpose <> _C_CmdVERIFY	Check whether the selected data can be used with the VERIFY command
THEN (Return code := _C_RC_CondOfUseNotSatisfied	
GOTO VERIFY_Exit)	
IF Ptr.CurrentDF = Ptr.MF THEN (	If the MF is selected, check whether its current security state allows the VERIFY command to be used. The security state of the MF must satisfy the security conditions specified in the key record
// MF is selected	
IF ((.Record[KeyRecord].EntryState <> SecurityState.MF) THEN (	
Return code := C_RC_SecStateNotSatisfied	
GOTO VERIFY_Exit))	
ELSE (	
// DF is selected	
IF .Record[KeyRecord].EntryState <> _SecurityState.DF THEN (	If the DF is selected, check whether its current security state allows the VERIFY command to be used. The security state of the currently selected DF must satisfy the security conditions specified in the key record
Return code := C_RC_SecStateNotSatisfied	
GOTO VERIFY_Exit)	
IF .Record[KeyRecord].RCntr >= .Record[KeyRecord].RCntrMax THEN (	Check whether the retry counter has reached its maximum value
Return code := C_RC_AuthMethodBlocked	
GOTO VERIFY_Exit)	
IF APDU.Cmd.Lc <> .Record[KeyRecord].KeySize THEN (	Check whether the supplied PIN has the same length as the reference PIN
Return code := C_RC_LcInconsistentP1P2	
GOTO VERIFY_Exit)	

---

**Table 13.64** (*continuation*): Small-OS program code: VERIFY command in compliance with ISO 7816-4

---

.Record[KeyRecord].RCntr := _	As a precautionary measure, increment the retry counter before making the actual PIN comparison. This defends against a potential attack based on analyzing the processing time or current consumption
.Record[KeyRecord].RCntr + 1	
CALL Kernel.CompareByteString	Compare the supplied PIN with the stored reference PIN
// Data 1: APDU.Cmd.Data[1 ...	
APDU.Cmd.Lc]	
// Data 2: .Record[KeyRecord]..	
KeyData[1 ... APDU.Cmd.Lc]	
IF STATUS(Kernel.CompareByteString) = _	
C_Equal THEN (	
.Record[KeyRecord].RCntr := 0	
IF Ptr.CurrentDF = Ptr.MF THEN (	If the supplied PIN matches the stored reference PIN, set the retry counter to zero unsuccessful attempts and set the security state to the state for successful PIN verification
// MF is selected	
SecurityState.MF :=	
.Record[KeyRecord]._	
ResultState.OK)	
ELSE (	
// DF is selected	
SecurityState.DF :=	
.Record[KeyRecord]._	
ResultState.OK))	
IF STATUS(Kernel.CompareByteString) = _	If the supplied PIN does not match the stored reference PIN, the retry counter will have already been incremented before the PIN comparison. Set the security state that results from an unsuccessful PIN comparison. If the retry counter has not reached its maximum value, set SW2 to the number of unsuccessful attempts still allowed. Otherwise, indicate in the return code that no further PIN verifications are possible
C_NotEqual THEN (	
IF Ptr.CurrentDF = Ptr.MF THEN (	
// MF is selected	
SecurityState.MF := _	
.Record[KeyRecord].ResultState.NOK)	
ELSE (	
// DF is selected	
SecurityState.DF := _	
.Record[KeyRecord].ResultState.NOK))	
IF .Record[KeyRecord].RCntr = _	
.Record[KeyRecord].RCntrMaxValue	
THEN (	
Return code := C_RC_AuthMethodBlocked	
ELSE (	
Return code := C_RC_CounterX (_	
.Record[KeyRecord].RCntrMax - _	
.Record[KeyRecord].RCntr)))	
GOTO VERIFY_Exit)	
Return code := C_RC_OK	The command has been processed with no error; an error exit is taken in all other cases
VERIFY_Exit:	
END WITH	
RETURN	

---

### 13.17.14.7 INTERNAL AUTHENTICATE command

The INTERNAL AUTHENTICATE command is used to authenticate the smart card using a challenge–response procedure. This is done by sending an eight-byte random number to the smart card, which encrypts it using the DES algorithm. The number of the key to be used must be given in parameter P2, which must also indicate whether the key to be used is located in the EF Key file of the MF or the currently selected DF. INTERNAL AUTHENTICATE falls under Case 4, which means that a data portion is present in the command APDU as well as the response APDU.

The ISO/IEC 7816-4 standard specifies only a few parameters for authentication commands. The cryptographic algorithm, for example, is not specified. In Small-OS, the DES algorithm is used as the default cryptographic algorithm. As an extension to the ISO/IEC 7816-4 standard, a five-byte key number is supplied with the command.

In Small-OS, INTERNAL AUTHENTICATE can in principle be used to encrypt eight bytes of plaintext into eight bytes of ciphertext using a selectable key. A smart card with Small-OS would thus fall under strict export control in almost all countries, so it would take several weeks or even months to obtain an export permit for the card. Consequently, INTERNAL AUTHENTICATE is implemented in many real smart cards such that it is not possible to directly encrypt data. This avoids the export restrictions.

The ability to directly encrypt a plaintext block into a ciphertext block is equally risky from a cryptographic perspective, since it could be used to generate plaintext–ciphertext pairs for brute-force attacks. As repeated improper use of INTERNAL AUTHENTICATE cannot be reliably prevented by using a retry counter, this implementation would also be very susceptible to differential fault analysis (DFA).<sup>22</sup> The most easily implemented form of attack would be a timing attack<sup>23</sup> on the AES computation, which for this reason must be noise-free so that the key cannot be determined by measuring the computation time.

For all these reasons, in practice the initial value is usually extended by appending a random number generated inside the smart card and the unique card number. The resulting number is then encrypted, and the ciphertext is sent back to the terminal along with the extended data. As a result, this command can no longer be used to encrypt data, which solves the export problem. In addition, encryption of a different number each time provides the basis for protection against differential fault analysis (DFA)<sup>24</sup> and differential performance analysis (DPA).<sup>25</sup> These measures illustrate relatively dramatically that the specification and implementation of even an apparently simple command such as INTERNAL AUTHENTICATE require considerable expertise and experience to protect the keys of a smart card application against attack.

The coding of the INTERNAL AUTHENTICATE command is described in Table 13.65, the coding of the response is described in Table 13.66, and the pseudocode is listed in Table 13.67.

<sup>22</sup> See also Section 16.5, ‘Attacks and Defense Measures During Card Usage’, on page 682

<sup>23</sup> See also Section 16.5.3, ‘Attacks on applications’, on page 727

<sup>24</sup> See also Section 16.5.1, ‘Attacks on the hardware’, on page 684

<sup>25</sup> See also Section 16.5.1, ‘Attacks on the hardware’, on page 684

**Table 13.65** Small-OS: Coding of the Case 4 command INTERNAL AUTHENTICATE

Data element	Coding	Remarks
CLA	'00'	—
INS	'88'	—
P1	'00'	—
P2	Y	$Y = 100Z ZZZZ$ use a key from the EF Key file in the currently selected directory (MF or DF) (specific reference data) $Z ZZZZ$ number of the referenced key (1 . . . 31)
$L_c$	8	Length of the supplied random number
DATA	...	The supplied random number
$L_e$	8	Length of the returned random number

**Table 13.66** Small-OS: Coding of the response to the INTERNAL AUTHENTICATE command

Data element	Coding	Remarks
DATA	...	If the command was correctly executed, this data element contains the encrypted random number that has been encrypted using the key referenced by the command
SW1    SW2	'9000'	Return code in case of correct command execution (successful PIN comparison)

**Table 13.67** Small-OS program code: INTERNAL AUTHENTICATE command in compliance with ISO 7816-4, extended with global and specific reference data (selectable reference to the MF or DF)

```

INTERNAL.AUTHENTICATE:
IF APDU.Cmd.P1 <> '00' THEN (
    Return code := C_RC_WrongP1P2
    Return code := C_RC_WrongP1P2
IF APDU.Cmd.Lc <> 8 THEN (
    Return code := C_RC_WrongLength
    GOTO INTERNAL.AUTHENTICATE_Exit)

IF Ptr.CurrentIEF.Key = C_InvalidPointer
THEN (
    Return code := C_RC_RefDataNotFound
    GOTO INTERNAL.AUTHENTICATE_Exit)
IF APDU.Cmd.P2.b5 ... b1 = 00000 THEN (
    Determine the number of the key to be used
    from P2
    Return code := C_RC_WrongP1P2
    GOTO INTERNAL.AUTHENTICATE_Exit)
ELSE (
    KeyNumber := APDU.Cmd.P1.b5 ... b1)

```

(Continued)

**Table 13.67 (continuation):** Small-OS program code: INTERNAL AUTHENTICATE command in compliance with ISO 7816-4, extended with global and specific reference data (selectable reference to the MF or DF)

---

<pre> SEARCH (for the key with the key number in - DF[Ptr.CurrentDF].EF[Ptr.CurrentIEF.Key].) IF STATUS(SEARCH) = C_Found THEN (     set KeyRecord to the record     containing - the found key WITH     DF[Ptr.CurrentDF]     .EF[Ptr.CurrentIEF.Key].) ELSE (     Return code := C_RC_RefDataNotFound     GOTO INTERNAL.AUTHENTICATE.Exit) IF .Record[KeyRecord].KeyPurpose &lt;&gt; - C_CmdINTAUTH THEN (     Return code :=     C_RC_CondOfUseNotSatisfied     GOTO INTERNAL.AUTHENTICATE.Exit) IF Ptr.CurrentDF = Ptr.MF THEN (     // MF is selected     IF .Record[KeyRecord].EntryState &lt;&gt; -         SecurityState.MF THEN (             Return code :=             C_RC_SecStateNotSatisfied             GOTO INTERNAL.AUTHENTICATE.Exit))     ELSE (         // DF is selected         IF .Record[KeyRecord].EntryState &lt;&gt; -             SecurityState.DF THEN (                 Return code :=                 C_RC_SecStateNotSatisfied                 GOTO INTERNAL.AUTHENTICATE.Exit)) CALL Kernel.DES_Encrypt     // plaintext: APDU.Cmd.Data[1 ... 8]     // key: .Record[KeyRecord]._     KeyData[1 ... 8]     // ciphertext: stored in APDU.Rsp.-     Data[1 ... 8] Return code := C_RC_OK INTERNAL.AUTHENTICATE.Exit:     END WITH RETURN </pre>	<p>Search EF Key for requested reference number specified by P2. If a key with the specified reference number is found, set the current key pointer to reference it</p> <p>Check whether the selected key can be used with the INTERNAL AUTHENTICATE command</p> <p>The security state corresponding to the condition specified by the key record must have been attained in the currently selected directory (MF or DF)</p> <p>Encrypt the supplied data (the random number) using the referenced key and place the result in the transmit buffer</p> <p>Command processed with no error; in all other cases an error exit is taken</p>
--	--

---

### 13.17.15 A simple application example

The following simple smart card application is intended to illustrate the structure and content of the EEPROM variables in the Small-OS operating system. Its operation can be described briefly as follows: it creates a 50-byte file whose contents can always be read and can be overwritten after successful verification of the PIN ‘1234’. The reference number of the PIN is 1, and a maximum of three unsuccessful attempts are allowed for PIN input. The EF containing the file is located below its own DF. All file names (DF names and FIDs) can be freely chosen.

Table 13.36 on page 532 shows how the required functions can be implemented by setting appropriate values in the structures used for the file tree, as described in Table 13.68. In order to realize the required access conditions, the state machine shown in Figure 13.49 on page 565 is implemented. After a reset, Small-OS automatically sets the security state of the DF to zero. In this state, the EF containing the data can be read, but not written. State 1 is necessary for writing to the file. If a VERIFY command is successfully executed with the correct PIN, the DF is set to security state 1 (... .ResultState.OK) and the file can be written. If PIN verification is not successful, the DF is set to security state 0 (... .ResultState.NOK).

The READ BINARY and UPDATE BINARY commands are used to read and write data from and to the file. The entire file or only part of the file can be read or written. The only requirement is that the security state attained in the DF must correspond to the desired type of access.

This example clearly shows two limitations of Small-OS that arise solely from the desire to keep the extent of the pseudocode within reasonable limits. The file cannot be read after security state 1 is attained, since reading is only allowed in security state 0. This could be remedied by implementing a ‘greater than or equal’ comparison in Small-OS, in addition to the ‘equal’ comparison. Another option would be to allow more than one access condition comparison for each operation (reading or writing) instead of only one comparison. With this approach, reading the file would be possible in state 1 as well as state 0, even if only ‘equal’ comparisons are possible.

Multiple access conditions for an access operation could be included in Small-OS just as easily as checking for greater than or equal. However, the pseudocode and the amount of code for a real implementation would be somewhat larger. In real smart card operating systems, such extensions can easily cause the amount of code to exceed the amount of memory (ROM or EEPROM) available in the microcontroller. These severe memory size limitations often prevent the implementation of certain useful functions in practice.

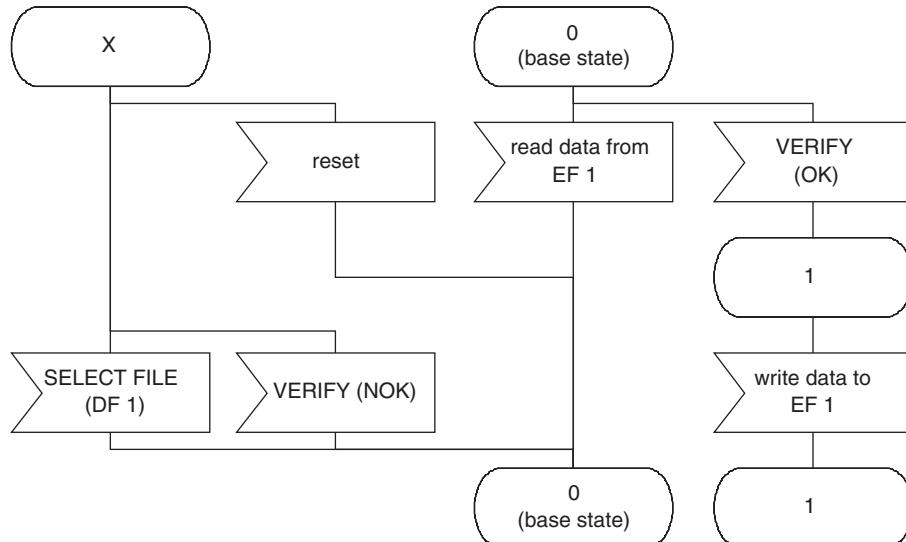
In the application described here, state 1 cannot be exited once it has been attained by successful PIN verification, which means that the file can no longer be read due to the access conditions. State 0 can be reached by unsuccessful PIN verification or by resetting the smart card. In terms of clean application design, this is a distinctly undesirable solution. However, the problem can be remedied by using a simple trick to return to the initial state. As you know, selecting a DF causes the current security state of the DF concerned to be reset. Consequently, the DF can be selected once more if necessary, which automatically resets its security state from 1 to 0.

Until recently, applications such as this were coded manually in assembly language in the form shown here. Now there are programs called application generators available for almost all commercial smart card operating systems. These programs run on PCs and have graphical user

**Table 13.68** Sample values of file management variables for a simple example application. It has a DF directly below the MF, which contains a transparent EF with a size of 50 bytes. The contents of this EF can be read at any time, but they can be modified only after a PIN has been entered

Variable name	Description, contents and size
MF data structures	
DF[1].FID := ‘3F00’	The standard MF FID
DF data structures	
DF[2].FID := ‘DF01’	The DF FID is ‘DF01’ (freely selectable)
DF[2].DFName := ‘D276’    ‘000060’	The registered AID of Wolfgang Rankl is used here as the DF name
DF[2].LenDFName := 5	The length of the DF name is 5 bytes
EF data structures	
DF[2].EF[1].FID := ‘0001’	The EF FID is ‘0001’ (freely selectable)
DF[2].EF[1].Structure := C_EFStrucTransp	The EF has a transparent file structure
DF[2].EF[1].AccessCondition.Read := 0	The required state for using the READ command with the EF. The access condition is set to 0, which means that reading the file is always allowed
DF[2].EF[1].AccessCondition.Update := 1	The required state for using the UPDATE command with the EF. The access condition is set to 1, which means that altering the file is allowed only after successful PIN entry
DF[2].EF[1].TransparentContentSize := 50	The size of the transparent file is 50 bytes
EF Key data structures	
DF[2].EF[2].Record[x].KeyData := ‘1234’	The (hexadecimal) PIN value is ‘1234’
DF[2].EF[2].Record[x].KeySize := 2	The length for the PIN is 2 bytes
DF[2].EF[2].Record[x].KeyNo := 1	The reference number of the PIN is 1
DF[2].EF[2].Record[x].RCntr := 0	The initial value of the retry counter is 0
DF[2].EF[2].Record[x].RCntrMaxValue := 3	The maximum value of the retry counter is 3, which means that the user is allowed at most three incorrect PIN entries.
DF[2].EF[2].Record[x].KeyPurpose := C_CmdVERIFY	The data in the EF Key record can only be used for PIN verification with the VERIFY command
DF[2].EF[2].Record[x].EntryState := 0	PIN verification is only possible in state 0
DF[2].EF[2].Record[x].ResultState.OK := 1	If the PIN comparison is successful, state 1 is set in the current DF
DF[2].EF[2].Record[x].ResultState.NOK := 0	If the PIN comparison is not successful, state 0 is set in the current DF

interfaces for generating files and access conditions. A similar process is used in the Smart Card Simulator when a new application or file is generated. After the files needed for the application have been generated, the data can be loaded into a smart card using the application generator and initial testing of the new application can be performed.



**Figure 13.49** State diagram of the file with FID ‘0001’ (EF 1). The EF data can be read or written by successful execution of READ BINARY or UPDATE BINARY commands sent to the smart card. State ‘X’ stands for any arbitrary state

This sample application can also be used to illustrate some interesting forms of attack. Although they are more theoretical than practical because they require sophisticated equipment, they nevertheless illustrate some noteworthy approaches. A prerequisite for these attacks is the ability to modify specific EEPROM contents, which in technical terms amounts to the ability to manipulate the electrical charges stored in the floating gates of individual EEPROM cells. The necessary techniques are described in more detail in Chapter 16, ‘Smart Card Security’, on page 667. Here we only show the consequences of this.

If it were possible to deliberately alter the content of the file tree pointer, which normally references the EF data, it could be manipulated to reference the data in EF Key. The EF is always readable in state 0, and in such a situation READ BINARY would read the PIN instead of the 50 data bytes in the EF. Of course, this attack requires knowledge of the specific address of the data portion of EF Key, and considerable insider knowledge is necessary to obtain this address. It would be easier if the variable DF[2].EF[1].TransparentDataSize could be altered. For example, if this variable is set to a large value, then READ BINARY can be used to read a block of data extending past the end of the file, corresponding to the new value of the variable. If EF Key is located in memory after EF 1, the EF Key header and file content can be read directly.

Manipulation of the EEPROM could also be used to repeatedly reset the PIN retry counter. The PIN could then be determined within an acceptable length of time by trial and error. An even simpler approach would be to set the PIN to a known value.

These examples clearly show that the security of a smart card would collapse completely if it were possible to manipulate the EEPROM. It would not matter if the contents of the EEPROM could only be overwritten instead of read; the PIN and the keys of the card could still be determined. The only thing that would need to be known is the specific memory

addresses where the manipulations must be performed. Checksums for the header contents, if present, would only make the necessary changes to the EEPROM more complicated, but they ultimately could not prevent these changes. However, it is presently not technically possible to modify individual bits at any desired location in an EEPROM. The approaches described there thus represent theoretically interesting forms of attack rather than truly dangerous forms of attack. However, these examples clearly and unambiguously show the potential dangers that would arise if this sort of EEPROM manipulation became possible in the future.

# 14

## Smart Card Production

This chapter describes the life history of smart cards, starting with the fabrication of the semiconductor chips, continuing with the production of the cards, and ending with the recycling of the card materials. The life cycle of smart card applications is also described in Section 14.2, ‘The Smart Card Life Cycle’, on page 569 because it is especially important for multiapplication cards, which are becoming increasingly numerous. The materials used to make the card body, the various types of modules, and other card components are described in Section 3.3, ‘Card Body’, on page 38.

A smart card basically consists of two completely different components. The first component is the card body, with its printing, security features, a magnetic stripe in some cases, and occasionally other card components.<sup>1</sup> The second component, which is what transforms the card body into a smart card, is the module incorporating the chip. This division of the card into two components applies equally well to memory cards and microcontroller cards.

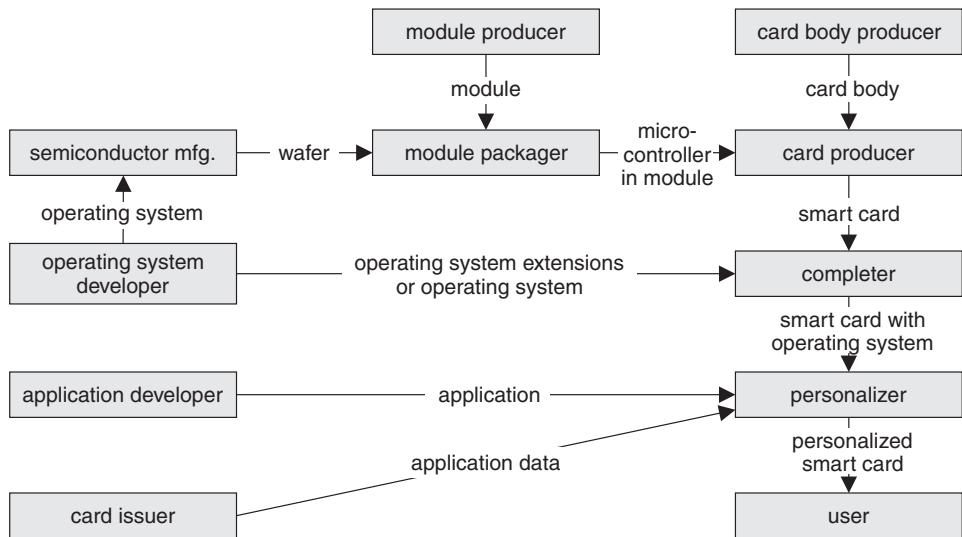
The data transmission mechanism also affects the structure of the card. Smart cards with contacts make electrical connections to terminals by means of six or eight externally visible contacts. By contrast, contactless smart cards have an antenna coil embedded in the card body and connected to the microcontroller. Naturally, this has a significant effect on the production methods used for the latter type of card.

The production process also depends to a considerable degree on other elements of the card, such as the card body material, the printing method, and the security features. However, regardless of all these parameters, one factor has top priority in the production of smart cards: cost reduction. Manufacturing smart cards is a mass production process with typical batch sizes ranging from 10 000 units to as much as 10 million units. Highly optimized production is the most important prerequisite for economical manufacturing of high-quality card products.

### 14.1 TASKS AND ROLES IN THE PRODUCTION PROCESS

Many different organizations, each responsible for a specific part of the overall process, are involved in the production of smart cards with the presently customary division of labor. The

<sup>1</sup> See also Section 3.5, ‘Card Components and Security Features’, on page 42



**Figure 14.1** Typical tasks and roles in the production of smart cards

resulting relationships, as shown in Figure 14.1, are very typical of the smart card industry. Naturally, the most important player in this process is the smart card producer, who coordinates all the activities from the start of the value chain to the handover of the smart cards to the card users.

The chain begins with the development of an operating system. It is usually produced by an organizational unit of the card producer and tailored to a specific smart card microcontroller. In special cases, the operating system may also be a licensed product of an independent software development company. If the operating system is intended to be embedded in the ROM of a microcontroller, it is transferred to the semiconductor manufacturer after completion of its development. The semiconductor manufacturer generates a ROM mask from the operating system software and enters it in the ROM during chip fabrication.

In the next step, a silicon wafer with thousands of individual microcontroller chips is sown into individual dice, and these dice are glued into modules and electrically connected to the contacts on the front of the module. This process is performed either by the semiconductor manufacturer or by a specialized module packaging company. The modules are supplied by a module producer. All of these process steps are monitored by suitable tests designed to check the mechanical and electrical properties of mass-produced items.

With the most common division of labor, the card producer receives suitably prepared card bodies from a supplier and modules with embedded smart card microcontrollers from a module packager. However, it is common practice for card producers to make their own card bodies in house. In the next step of the process, the modules are glued into the card bodies, which yields physically complete smart cards.

In the case of a ROM-based operating system, the next phase is card completion, in which the operating system extensions and corrections are stored in the nonvolatile memory (EEPROM) of the microcontroller chips. This is usually done by the smart card producer, but in certain cases it can just as easily be done by an external party.

If the operating system is designed for a microcontroller with flash memory, it is loaded into the smart cards at this stage with the aid of a boot loader located in the microcontroller.

Any necessary software modifications can also be incorporated at this point. The result of this phase is a smart card with fully complete hardware and a functional operating system. The cards can be sold as ‘white plastic’ at this point, although this is rarely done.

The vast majority of cards proceed from here to the initialization and personalization stage. For this purpose, the card issuer supplies the necessary constants and individual data for the application. If the application is code-based, the application developer also provides the program code (such as an applet) for the application. Depending on the specific scenario, the personalizer may also generate supplementary data. The data and code are then loaded into the cards with the aid of the smart card operating system. In addition to this electrical personalization, visual personalization must be performed – which means applying visual data to the cards, such as text or images. After this has been done, smart card production is complete and the cards can be transported to the end users via various channels.

## 14.2 THE SMART CARD LIFE CYCLE

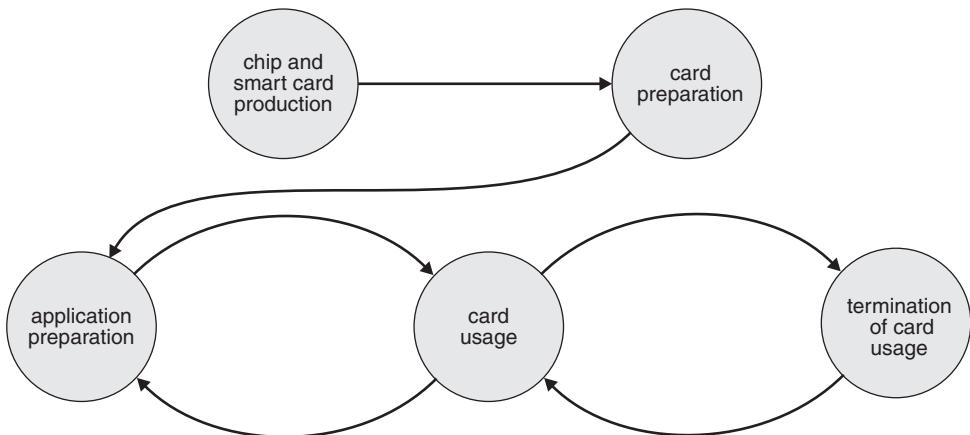
The life cycle of a smart card depends on its application area. A smart card for the GSM mobile telecommunication system, for example, follows a different path than a credit card with an implanted chip. Nevertheless, the various types of cards have much in common.

The ISO 10202-1 standard defines a card life cycle that applies equally well to all production methods and a wide variety of applications. This standard is very strongly oriented toward financial transaction applications and the information technology used in these applications, rather than the actual production of card bodies and chips. Nevertheless, it represents a reasonably successful attempt to provide a structured description of the life history of smart cards from the beginning to the end. It is thus used here as the basis for describing the smart card life cycle.

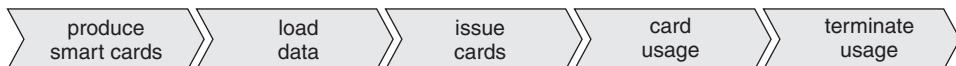
According to ISO 10202-1, the life of a card is divided into five phases, which are linked by precisely specified transitions. This is shown in Table 14.1 and illustrated in Figures 14.2

**Table 14.1** Individual life cycle phases and associated activities according to the ISO 10202-1 standard

Smart card life cycle phase	Typical activities
Phase 1: Production of the chip and the smart card	<ul style="list-style-type: none"> <li>• designing the chip</li> <li>• generating the smart card operating system</li> <li>• producing the chips and modules</li> <li>• producing the card body</li> <li>• implanting the modules in the card bodies</li> </ul>
Phase 2: Card preparation	<ul style="list-style-type: none"> <li>• completing or loading the smart card operating system</li> </ul>
Phase 3: Application preparation	<ul style="list-style-type: none"> <li>• initializing application(s)</li> <li>• personalizing (individualizing) the application(s), both visually and electrically</li> </ul>
Phase 4: Card usage	<ul style="list-style-type: none"> <li>• activating applications</li> <li>• deactivating applications</li> </ul>
Phase 5: Termination of card usage	<ul style="list-style-type: none"> <li>• deactivating applications</li> <li>• deactivating the card</li> </ul>



**Figure 14.2** The life cycle of a smart card according to the phase model of the ISO 10202-1 standard



**Figure 14.3** The essential process steps in the life cycle of smart cards

and 14.3. All stockpiles of cards made necessary by logistics of the production process and all transportation paths between the various companies that perform the various production operations must be physically or cryptographically secured in order to make the manipulation or theft of semi-finished products as difficult as possible.

All production steps must naturally be accompanied by appropriate quality assurance. As smart cards are normally used in areas related to security, it is customary to ensure the traceability of the production process. This means that all production steps must be logged using batch and chip numbers. It must be possible to reconstruct the production steps undergone by each individual smart card at any desired time afterward. This makes it easier to analyze the causes of any manufacturing faults that may show up.

Each microcontroller is unique after semiconductor fabrication as a result of assigning individual chip numbers during fabrication, so traceability can be implemented relatively easily using these chip numbers. Production traceability is implemented either by maintaining a production database or by writing all the information relevant to the fabrication of each chip (see Table 14.2 on the facing page) directly to the chip. ISO 10202-1 recommends storing the production data in the chips, which has certain advantages compared with storing the data in a database. If the data is stored in the chips, the production data for any chip can be obtained without having to access a database, although this data occupies space in the EEPROM or flash memory of the microcontroller. Both storage locations are commonly used in practice, since this is the most flexible solution and high-performance processor cards have sufficient space in nonvolatile memory for the production data.

The production of security-related devices such as smart cards is often verified by audits. The requirement for such audits may be imposed by an industrial organization or by customers. In many cases, successful completion of an audit is required before the smart cards can be delivered to the customer. There are specific audits for almost all large markets for smart

**Table 14.2** Production data typically stored in chips. This data can be written just once, after which it can only be read (WORM attribute of a file or data object)

Life cycle phase	Typical production data
Phase 1: Chip and smart card production	<ul style="list-style-type: none"> <li>• ID of the chip manufacturer</li> <li>• ID of the fabrication line</li> <li>• unique chip number</li> <li>• chip type</li> <li>• ID of module packager</li> <li>• date and time of packaging the chip in the module</li> </ul>
Phase 2: Card preparation	<ul style="list-style-type: none"> <li>• ID of the initializer</li> <li>• ID of the production machine</li> <li>• date and time of initialization</li> </ul>
Phase 3: Application loading	<ul style="list-style-type: none"> <li>• ID of the personalizer</li> <li>• ID of the production machine</li> <li>• date and time of personalization</li> </ul>

cards, such as telecommunication, payment systems, and so on. They have certain similarities in terms of their requirements, but they differ in many details and have different focuses.

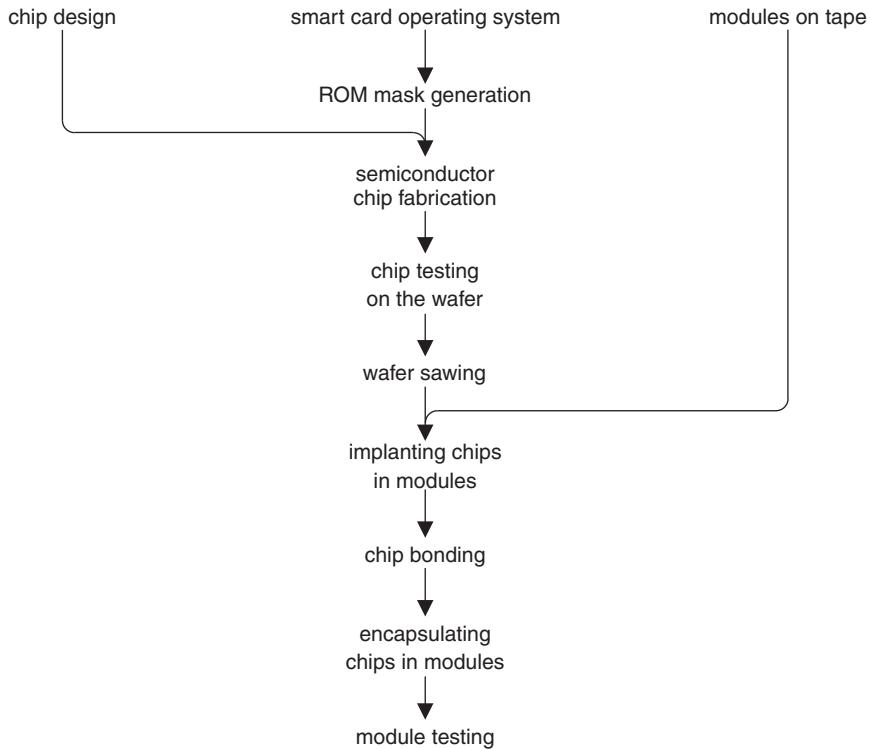
A typical example of a smart card production audit is the SAS audit performed by the GSM Association (GSMA) [GSMA]. SAS stands for Security Accreditation Scheme, and it concentrates primarily on the production of SIM cards and the associated production environment. An SAS audit is usually performed by two independent external auditors on request of the card producer, and it takes around eight working days. The auditors examine the entire smart card production process and the producer's security system based on an analysis of the relevant documents, random-sample checks of production equipment, and interviewing the responsible persons. In addition to the actual production process, this covers related topics such as key generation and key management. If a deficiency is recognized, the producer is obliged to correct it within twenty working days, after which the result is checked by a follow-up audit. Finally, the audit results are recorded in an audit report and an official certificate (for hanging on the wall and presentation as appropriate) is generated. The audit is typically required to be repeated every two years to maintain the validity of the certificate. The basic procedure of an SAS audit is described in the GSDMA document 'Security Accreditation Scheme – Methodology'.<sup>2</sup>

## 14.3 CHIP AND MODULE PRODUCTION

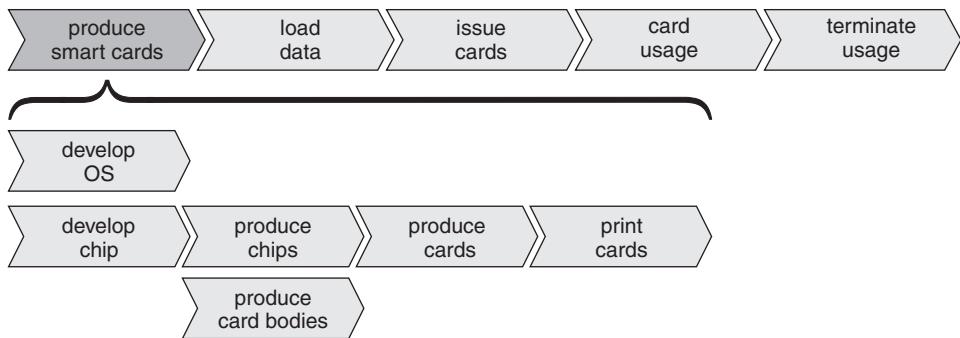
As illustrated in Figures 14.4 and 14.5, the first phase of the life cycle can be divided into two parts. The first part consists of generating the smart card operating system and the semiconductor fabrication process for producing the microcontrollers, while the second part is a parallel process that comprises the entire technology for producing the card bodies.

The semiconductor devices are produced in protected facilities with restricted access. Restricted access is relatively easy to achieve with cleanroom facilities, since they already have physically controlled access via air locks. However, restricted access is also important in terms of security, since it is the only way to ensure that no semiconductor components with Trojan horses or other malware in their software can be smuggled into the system during chip

<sup>2</sup> See [SAS 07]



**Figure 14.4** Chip and module production in the life cycle of a smart card with the operating system in the microcontroller ROM



**Figure 14.5** The process steps for producing smart cards as part of the overall smart card life cycle

fabrication or after the dice have been separated. This would otherwise be a very serious and relatively dangerous form of attack on the security of smart card applications.

### 14.3.1 Chip design

A chip for a memory or microcontroller card is always square or as close as possible to square because this minimizes the risk of chip breakage from the stresses generated by card flexing.

Although full protection of the chip against bending stresses would be technically possible with an extremely stiff module, this is not desirable in practice. A very stiff module would eventually cause the card body to crack due to the alternating bending stresses to which the card is constantly exposed.

The semiconductor components used in the chip design, such as the CPU, numeric co-processor and various types of memory, are normally standard components that have been hardened to provide increased security.<sup>3</sup> Sometimes components intended for the automotive sector are used, since they must be designed to meet similarly severe environmental and reliability requirements. However, they must be fully adapted and revised to meet the security requirements of smart card microcontrollers.

After the functional specification has been generated, the process of chip design begins with the definition of the general architecture in the form of a block diagram and a rough microcontroller layout. After this, the block diagram is refined step by step at increasing levels of detail: logic blocks, logical gates, transistors, and ultimately the geometric structures of the individual exposure masks. In this process, each step is accompanied by simulation of the circuitry and elaborate tests. This complex process consists of many individual steps, and it takes considerable experience to achieve an optimal arrangement of the components on the chip. Finally, samples are produced on a test fabrication line in a semiconductor plant, and these first reference products are measured and tested with the utmost precision. A security assessment is often performed in parallel, although it cannot be completed until the first true chips are available.

Chip design is a process that can take several months to a year before fully functional chips that meet all the requirements for mass production are available. This level of effort explains why the time between successive generations of smart card microcontrollers is two to five years. Due to the cost of making major changes to the chip, the most common forms of modification are chip shrinking to achieve better utilization of the wafer area and minor hardware improvements or extensions.

### 14.3.2 Smart card operating system development

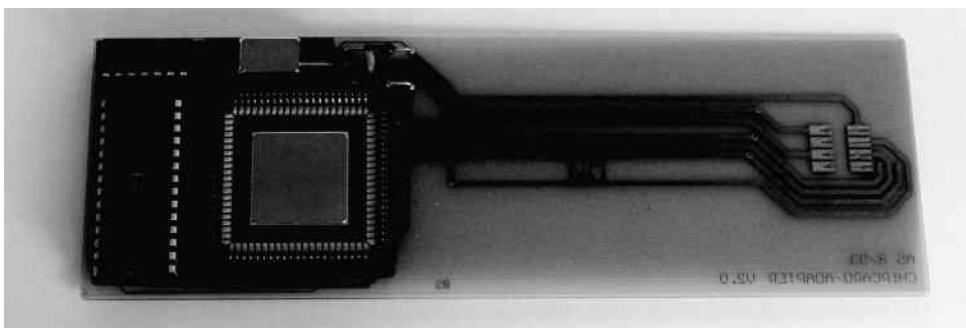
Operating systems and other software for smart card microcontrollers are such a large subject that an entire chapter is devoted to discussing all aspects of this subject in detail.<sup>4</sup> However, it should be noted that a significant part of the technical basis for the security of a smart card during the rest of its life cycle is established during chip fabrication. Even a high-quality operating system with elaborate cryptographic protection is of little use if all the secret data can be read out due to an error in semiconductor fabrication.

Due to the small memory capacities of smart card microcontrollers, software for smart card operating systems and the applications based on these operating systems is written in the C language, which is relatively close to the hardware level. Compared with assembly language, which was commonly used in the early history of smart cards, C helps reduce software development time.

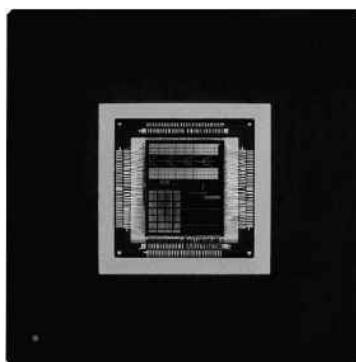
The first smart card microcontrollers could only store program code in ROM, since the structure width of ROM was significantly smaller than that of EEPROM. However, around 2003 semiconductor technology had progressed to the point that the difference in the memory

<sup>3</sup> See also Section 5, ‘Smart Card Microcontrollers’, on page 73

<sup>4</sup> See also Chapter 13, ‘Smart Card Operating Systems’, on page 441



**Figure 14.6** Mini smart card emulator with the mask ROM replaced by a removable EEPROM in a DIP package. The large chip is a smart card microcontroller with all buses freely accessible (bond-out chip)



**Figure 14.7** Example of a bond-out chip for use in smart card microcontroller emulators (Infineon)

densities of ROM and EEPROM (or flash memory, which is now available) became negligible. As a result, more and more operating systems are located in EEPROM or flash memory instead of being ROM-based. However, even modern flash-based microcontrollers usually have a small ROM region that holds a boot loader and a few basic test routines for the hardware.

The tests located in the ROM or downloaded into flash memory are elaborate and comprehensive, since it is practically impossible to correct errors in this software after chip fabrication.<sup>5</sup> This always involves generating a ROM mask, which holds the software that will later be located in the microcontroller ROM where it cannot later be modified. If a software error is detected in one of the subsequent production steps, it can only be corrected by repeating all of the preceding steps.

If a microcontroller with flash memory is used instead, some of the previously described steps become simpler. The most important factor is elimination of the ROM mask, which yields time savings of at least several weeks. Another advantage is that the operating system can be tested using real hardware during development.

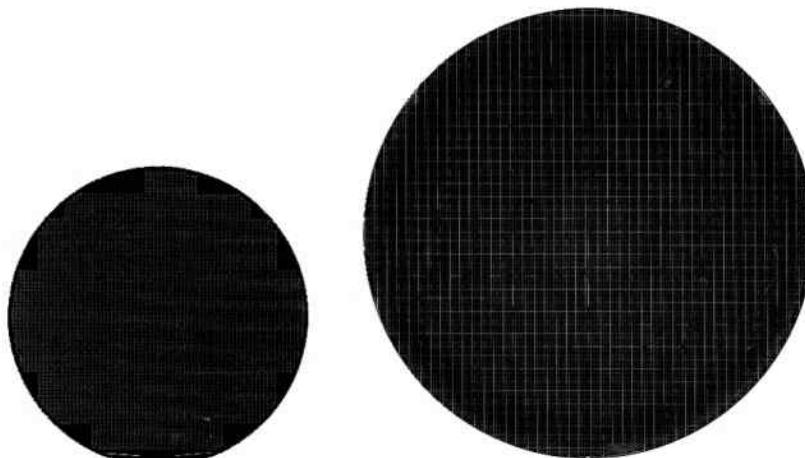
<sup>5</sup> See also Section 15.4, ‘Test Methods for Software’, on page 645

In order to make optimal use of the available memory space in the microcontroller, the program code must be adapted to the specific type of chip that is used. Porting the software to another type of chip thus requires additional time and effort. As a result, the development time for a new smart card operating system is around nine months, although it may be significantly shorter if existing program code can be reused. After the development of the operating system is complete, it can be handed over to the semiconductor manufacturer if it is ROM-based.

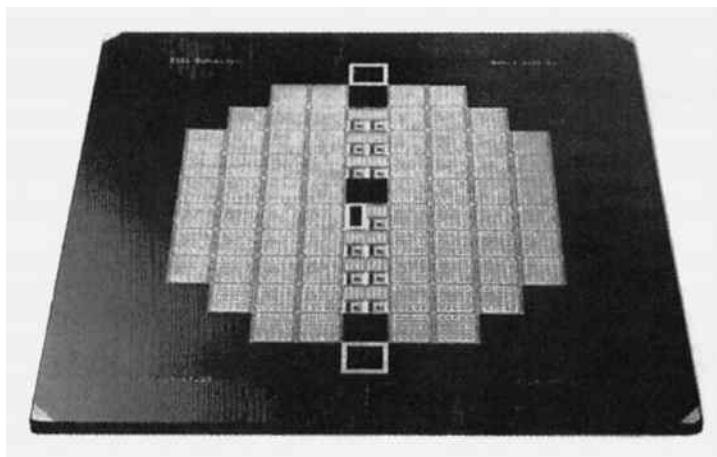
### 14.3.3 Chip fabrication in semiconductor plants

In the case of a ROM-based operating system, the semiconductor manufacturer uses the software supplied on a data storage medium or by data telecommunication to generate an exposure mask for the microcontroller ROM. Operating system designers call this exposure mask, which contains the program code, the ROM mask or simply the mask. If the structures are reduced in size when the ROM mask is imaged onto the wafer, it is called a reticule. The ROM mask is only one of around 25 to 30 masks needed to produce a microcontroller. It is also not the first mask in the series, but instead somewhere in the middle. This has the benefit that the microcontroller can be produced in advance up to this fabrication step, since this is when the operating system is embedded in the hardware using the ROM mask. A typical fabrication line has a throughput of 5000 to 8000 wafers per week.

The semiconductor structures of what will ultimately become the microcontroller chips are produced on suitably prepared high-purity disks of silicon called wafers (see Figure 14.8). Wafers with a diameter of 8 inches (20.5 cm) are normally used at present to produce smart card microcontrollers, although 12-inch (30.7-cm) wafers are used in isolated cases. Assuming die dimensions of  $5 \times 5$  mm and 90 % yield, an 8-inch wafer can hold approximately 5700 microcontrollers and a 12-inch wafer can hold approximately 13 000.



**Figure 14.8** An eight-inch wafer with approximately 5700 microcontrollers and a twelve-inch wafer with approximately 13 000 microcontrollers. The milled flat (primary flat) on the left-hand wafer is used to align the wafer during semiconductor fabrication. With twelve-inch wafers, alignment is performed optically using fiducial marks (Samsung)



**Figure 14.9** A quartz glass photomask for simultaneous exposure of an entire wafer (Reproduced with permission from Philips)

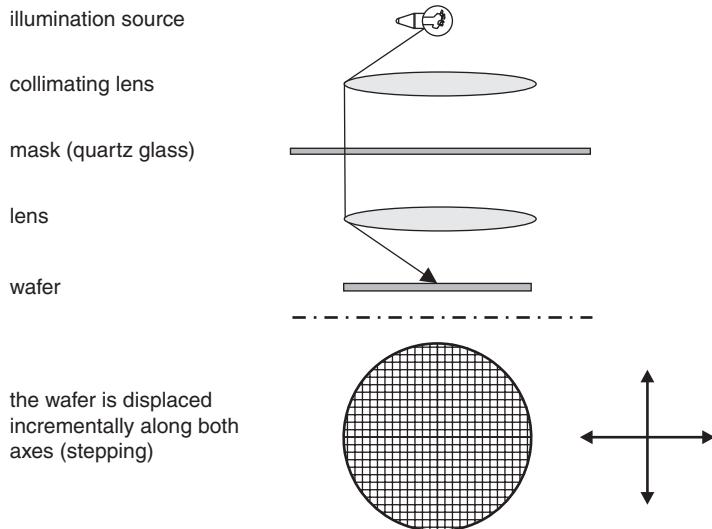
There is a general trend in the semiconductor industry toward fabrication using larger wafers and smaller structure widths. It can be assumed that in the near future, 12-inch wafers and 90-nm technology will become the prevailing standard for the production of smart card microcontrollers. The cost of a fabrication plant at this level of technology is around two billion euros.

As recently as the mid-1990s, semiconductors were fabricated using photomasks that allowed all the devices on a 4-inch wafer (such as smart card microcontrollers) to be exposed at the same time. Contact exposure was normally used with these masks. With increasingly smaller structure widths, this technique was no longer possible because yields were too low.

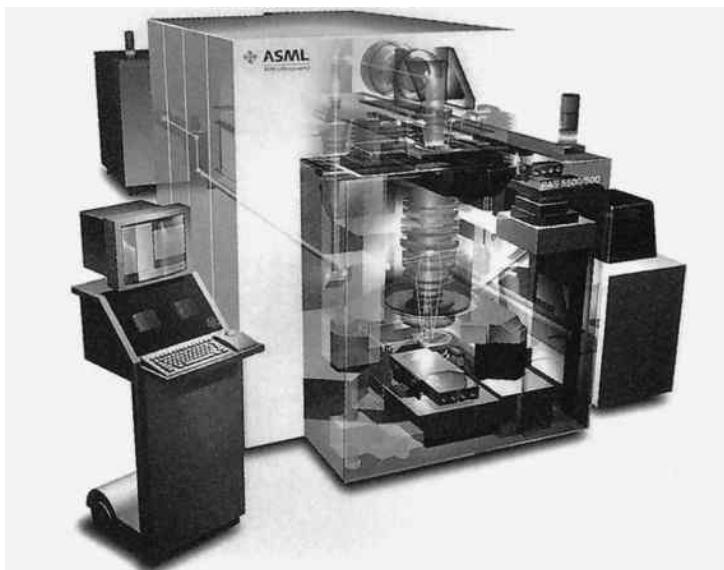
In all new production methods, a set of photomasks represents only a single chip instead of an entire wafer. These very delicate masks are made from sheets of quartz glass (see Figures 14.9), which is transparent to ultraviolet light, that act as substrates for patterns etched in chromium, which are imaged onto the wafer to produce the chips. These patterns are made by first coating the quartz sheet with a photosensitive layer and then using an electron-beam writer to expose the pattern. Following this, the photosensitive layer is developed and the unexposed areas are removed by etching.

The photomasks are produced at a scale of 5:1 or 10:1 relative to the actual chip size, which allows image-enhancing reduction to be used during wafer exposure. The machines used to expose the wafers, which are called steppers, are high-precision optical instruments that can focus the image of the mask on the wafer with an accuracy of a fraction of a micrometer and reposition the wafer with the same precision. After the pattern has been exposed at one chip position using ultraviolet light, the wafer is moved by one step to the next chip position and the exposure is repeated. The wafer is thus exposed step by step until all the microcontrollers it will contain have been exposed (see Figures 14.10 and 14.11).

The entire wafer is coated with a light-sensitive lacquer called photoresist. In the areas where the photoresist is exposed to light, the lacquer is removed by etching and the underlying wafer surface is doped with impurity atoms. After the wafer has been cleaned several times and coated with a new photoresist layer, it is ready for exposure with the next photomask. Depending on the semiconductor manufacturer and the fabrication method that is used, producing a finished



**Figure 14.10** Operating principle of step-and-repeat exposure of individual chips on a semiconductor wafer using a stepper



**Figure 14.11** Cross-sectional image of a stepper for step-and-repeat exposure of individual chips on a semiconductor wafer (Reproduced with permission from ASM Lithography)

wafer involves around 400 process steps and an elapsed time of six to twelve weeks, although the actual processing time is only around three weeks. The significantly longer elapsed time results primarily from the queuing technique commonly used in the mass production of semiconductor devices.

To make the production process economically feasible, wafers must always pass through the fabrication process in batches. A typical batch consists of twelve wafers, which is usually the minimum production quantity. Quantities smaller than a full batch require considerable extra

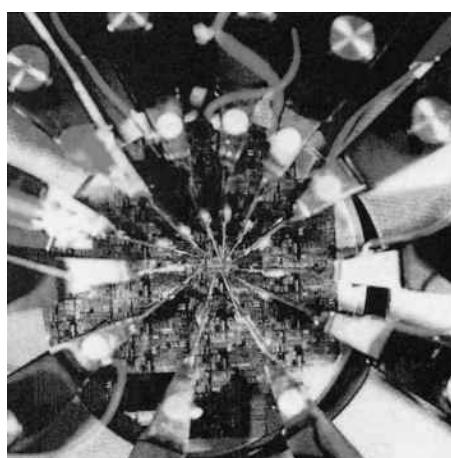
effort, with the result that the production cost is the same as for a full batch. However, many fabrication processes allow shared batches or multiproject wafers, so that microcontroller chips with different ROM masks can be produced on a single wafer. This allows smaller batches to be produced (around 1000 units).

#### 14.3.4 Chip testing on the wafer

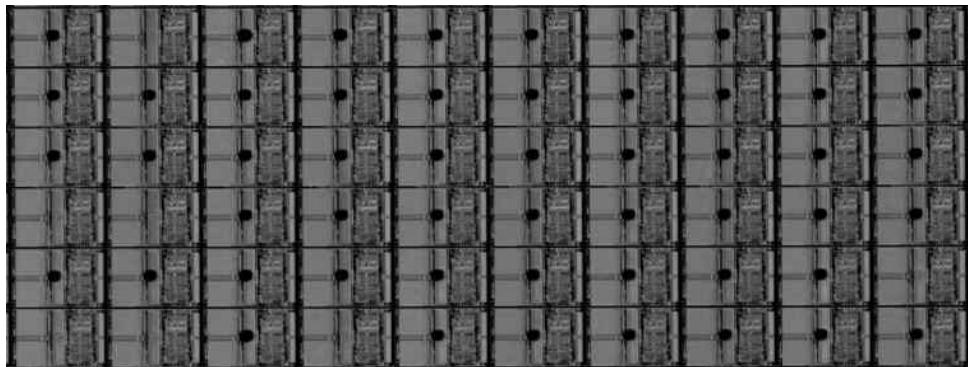
In the next production step, the microcontrollers on the wafer are contacted using metal probes and tested individually (see Figure 14.12). This involves making contact with the microcontrollers on the wafer in groups of up to 100 devices and testing them for correct electrical operation. As supplementary contacts for the microcontrollers are usually not available, even at this production stage, only the five contacts that will later be used in the smart card can be used for testing.

The functional units in the silicon are tested significantly more intensively and extensively at this point than later on, since the microcontrollers are still in test mode at this stage. In this mode, all of the memories (RAM, ROM, EEPROM and flash) can be read and/or written without any restrictions. In the past, microcontrollers that failed this test were marked with a small colored dot as shown in Figure 14.13 on the next page. This allowed defective chips to be identified visually in the following production steps and discarded after the wafer was sown into individual dice. In modern systems, the coordinates of the defective devices are simply sent to the next machine in the production sequence as data records, where they are taken into account as appropriate in further processing.

The ability to access the memory freely in test mode is also exploited to write chip-specific data to the nonvolatile memory. This can include a serial number that is used only once, so that it is unique. This individualizes each chip, and thereby the smart card in which the chip is subsequently implanted. In addition to certain security considerations, this has advantage that traceability based on the unique chip number is always assured.



**Figure 14.12** Testing a microcontroller on the wafer. Each IC is tested using needle probes that make contact with the device (Reproduced with permission from Philips)



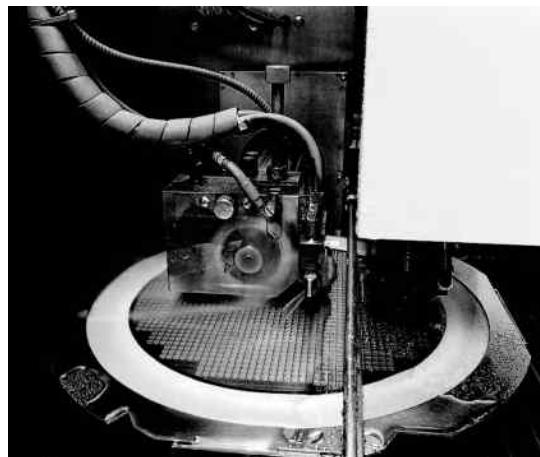
**Figure 14.13** A portion of a wafer with microcontroller ICs. The dots on some of the chips mark defective devices. The areas between the individual chips are called sawing lines

With smart card microcontrollers that have flash memory as nonvolatile memory, in principle it would be possible to load the entire operating system or part of the operating system into the flash memory at this time in addition to testing the chip hardware. Compared with loading the operating system after the chips are implanted in the card bodies, this has the advantage that it can be done faster at this point because the chips are still in test mode. This mode enables higher data transmission rates and faster write operations with nonvolatile memory than after the chip has been switched to user mode. This is possible because all the physical and electrical parameters are known precisely at this point, so it is not necessary to allow for the full range of parameter values allowed by the chip specifications.

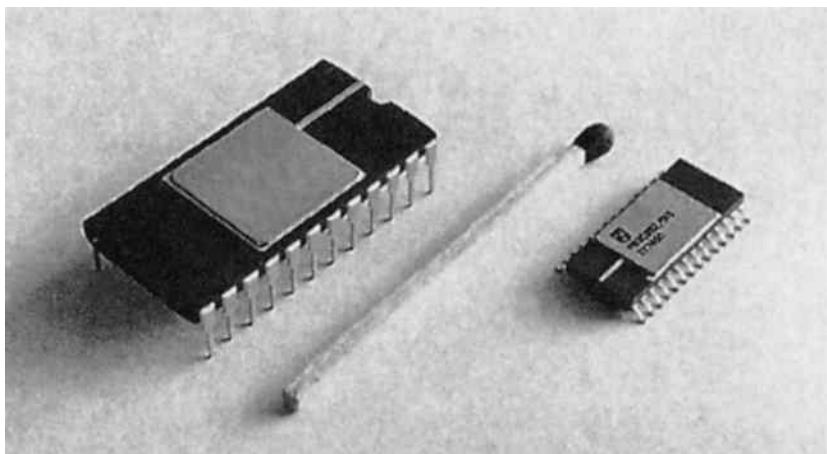
### 14.3.5 Wafer sawing

After the chips have been tested on the wafer, the next step is to separate them. A thin, self-adhesive film is first applied to the back of the silicon wafer to hold the individual chips in place after the wafer has been sawn. Special saws with a blade thickness of approximately 25 µm, operating at more than 30 000 rpm, are used to cut the wafer into pieces (see Figure 14.14). Sometimes high-pressure water jets are used instead for wafer cutting. The wafer is cut in a grid pattern so that each resulting piece holds a single microcontroller. These pieces of crystalline silicon with their microcontroller structures, which have a maximum size of 25 mm<sup>2</sup>, are called dice (die in the singular) individually (see Figure 14.16). After the wafer has been separated into dice, the defective dice identified by colored dots or data sets are separated from the good dice and destroyed.

Up to this point, it is not possible to determine whether the ROM software has been copied without any errors. For this reason, around ten dice are removed from the batch at this stage and mounted in ceramic DIL packages (see Figure 14.15). The software producer receives these first sample devices and uses its test facilities to determine whether the software in the ROM functions correctly. The entire chip can be tested in this manner. If an error in the software or hardware is discovered at this point, the production process must be stopped and the entire batch is worthless. After the error has been corrected, the production process must be started again from the first step at the semiconductor manufacturer's facility. The lost time cannot be recovered, even with accelerated processing in other production phases.



**Figure 14.14** Sawing a wafer into dice (Reproduced with permission from Hitachi)



**Figure 14.15** Examples of microcontrollers mounted in various types of ceramic packages for software testing



**Figure 14.16** An individual die with a match for size comparison. It has dimensions of  $2.2 \times 2.1$  mm, with a thickness of 0.12 mm

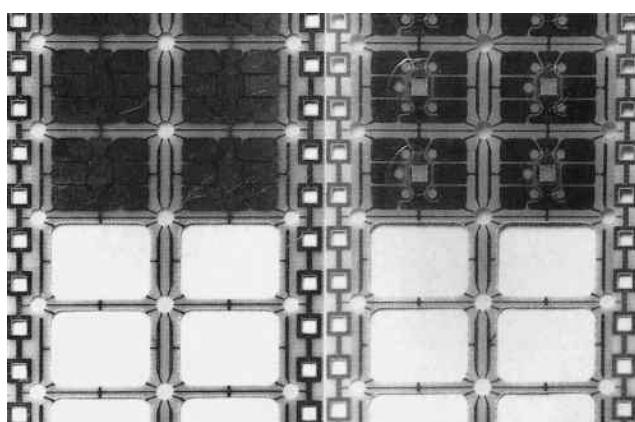
### 14.3.6 Packaging chips in modules

The next step in the process after the dice have been sawn from the wafer is to package them in modules. The modules increase the resilience of these very fragile bits of quartz crystal, and the electrical contacts on their top surfaces will later provide the connections between the card and the terminal.

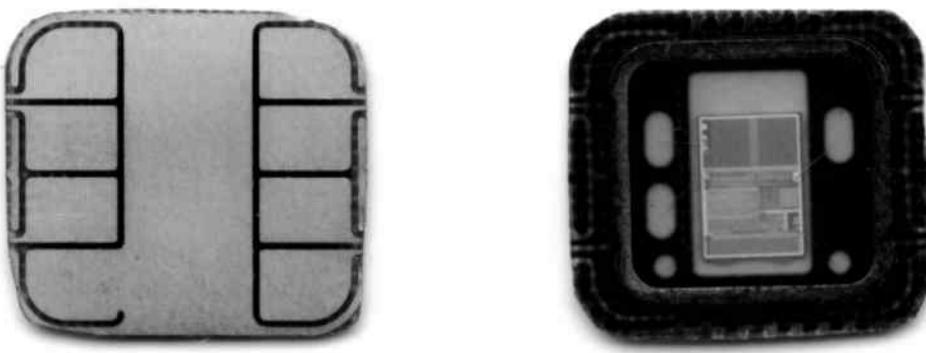
Chip modules are usually supplied on reels of 35 mm plastic film with perforated edges, which carry modules in adjacent pairs (see Figure 14.18). Depending on the size of the module, a single reel can hold 10 000 to 20 000 modules. The 35-mm carrier film is called tape in the trade jargon, and this form of packaging is called chip on tape (COT). A typical reel is shown in Figure 14.17.



**Figure 14.17** A reel of tape holding paired modules. Module producers typically use this form of packaging to supply modules to smart card producers



**Figure 14.18** Example of a 35-mm tape with attached pairs of modules. The front of the tape is shown on the left, while the rear is shown on the right. The holes left by modules that have already been punched out of the tape can be seen in the lower portions of the pictures



**Figure 14.19** Example layout of the front and rear surfaces of a module. The chip has not yet been encapsulated in black epoxy resin

Incidentally, the tape has the same width as 35-mm photographic film used in still and motion-picture cameras. This tape size originates from the early days of smart card manufacturing. At that time, the 35-mm film format was chosen for module tape to minimize the need to develop new equipment for transport and packaging, since standard commercial film spools and winding equipment could be used for module tape. Changing to a different format was no longer economically feasible after this format came into general use, so it is still used today.

The dice are usually glued permanently into the modules with the silicon substrate facing up (away from the rear surface of the module), as illustrated in Figure 14.19. This allows the die to be electrically connected to the contacts on the rear of the module in a subsequent production step. If flip-chip technology is used, the dice are glued into the modules with the opposite orientation, which means with their top surface facing the rear surface of the module, with the electrical connections being made at the same time.

### 14.3.7 Chip bonding

After the dice have been glued into the modules, the next step is to make the electrical connections to the rear surfaces of the contacts. This is done using very fine gold wire, which is welded to the aluminum pads on the die and the corresponding contact surfaces on the back of the module.

To prevent the bonding wires from being broken by strong temperature fluctuations, each wire is formed into a loop. However, the loops must not be too large, as otherwise the wires would not be fully covered when the chip is subsequently encapsulated in resin and would thus be subject to corrosion. This technique is illustrated in Figure 3.33 on page 53. An alternative to wire bonding is flip-chip technology, in which the die is connected directly to the contact surfaces without using bonding wires.<sup>6</sup>

<sup>6</sup> See also Section 3.6.1, ‘Electrical connections between the chip and the module’, on page 51

### 14.3.8 Encapsulating the chips in modules

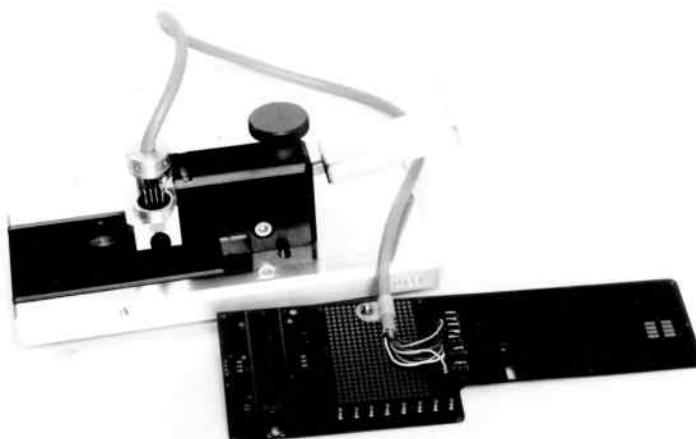
After the chip has been bonded, it is encapsulated in black epoxy resin poured over the chip and the back of the module. This resin protects the fragile chip against environmental influences such as humidity, twisting and bending. An opaque encapsulating resin is used because semiconductor devices are usually very sensitive to light and electromagnetic energy in the near-visible spectral regions. This encapsulation is also called ‘glob top’.

After the chips have been encapsulated, the carrier tapes holding the modules are wound onto film reels and packed in cardboard boxes. With small production batches, the modules can also be separated and packaged in plastic bins. However, this is avoided with large production volumes because it would be difficult for the module implanter to use automated production equipment.

If the objective of the production activities is launching a new microcontroller or testing modified chip hardware, encapsulation of the modules is the last step of the production process, which is then followed by suitable tests and qualification stages. In this case, mass production with suitably modified hardware and/or software starts only after testing and qualification are completed with no errors. A similar situation exists with an entirely new operating system, in which case hardware production also ends at this point. It is followed by the necessary qualification testing, which can take weeks or even months. If any errors are discovered, the process must be repeated with an improved version of the operating system or modified hardware.

### 14.3.9 Module testing

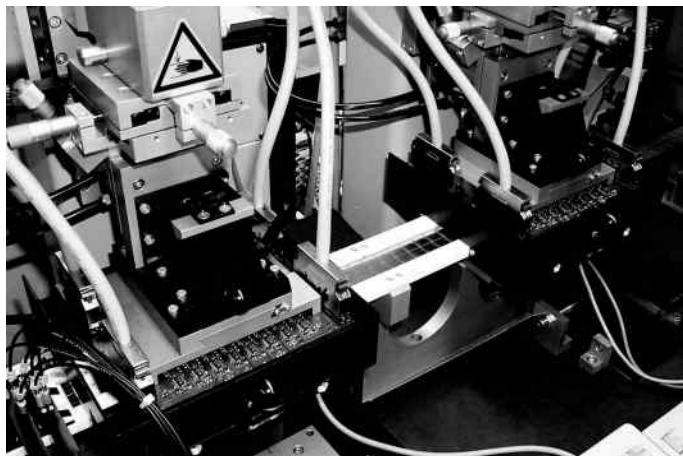
As a consequence of the production steps up to now – sawing the wafer into dice and packaging the dice in the modules – a certain number of dice become defective. Accordingly, additional testing is usually performed before the modules are packaged and shipped. This requires connecting each module to a tester via the contacts on the top surface of the module (see Figures 14.20, 14.21 and 14.22).



**Figure 14.20** An adapter for connecting a module on tape to a card in ID-1 format

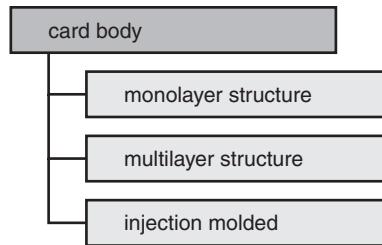


**Figure 14.21** An machine for testing modules on tape

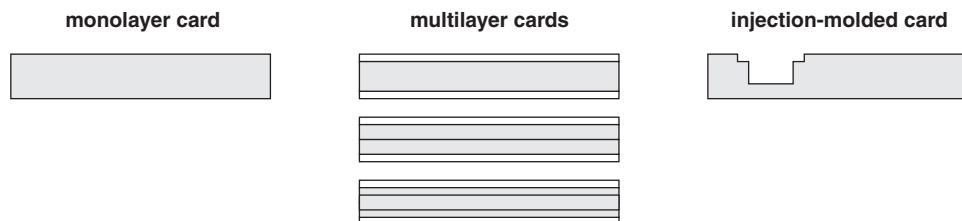


**Figure 14.22** Detail of a chip module tester. Each of the two contact heads makes connections to 16 modules at a time, and these modules are tested in parallel. Machines of this type can also be used to initialize smart card microcontrollers

The test computer then performs an ISO activation sequence and attempts to detect a valid ATR. If this is possible, it tests the chip hardware using the commands integrated in the mask-programmed software. If all these tests are successful, the module has not been damaged by any of the previous production steps and it can be implanted in a smart card. After completing the tests, the test computer switches the microcontroller from test mode to user mode by blowing a polysilicon fuse or writing a special code to a specific location in nonvolatile memory. In user mode, it is no longer possible to externally access the memory for reading or writing without first satisfying defined security conditions.



**Figure 14.23** Basic production methods for plastic card bodies



**Figure 14.24** Overview of commonly used card structures. A multilayer card consists of two overlay foils (outer layers) and one or more core foils (internal layers)

## 14.4 CARD BODY PRODUCTION

The card body is the substrate for the module and the various card components. Up to now, card bodies have always been made from various types of plastic, although a wide variety of production methods are used.<sup>7</sup>

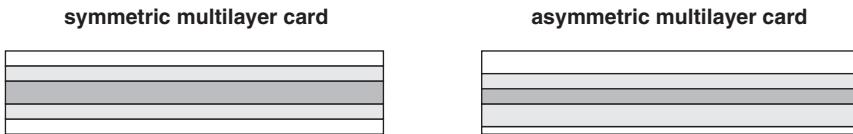
In principle, card bodies for smart cards can be produced using three different methods, as shown in Figures 14.23 and illustrated in Figure 14.24. They differ in terms of the durability of the card, the surface quality, and the card components that can be used. Many card producers offer only one method instead of the full range of production methods.

Laypersons often regard the production of card bodies as an uncomplicated, easily mastered technology that essentially only amounts to punching out a few pieces of plastic foil and gluing them together. However, this is by no means true. Mass production of high-quality card bodies involves a multitude of complex manufacturing steps, and it demands outstanding mastery of the chemical processes associated with the plastic materials and accompanying inks. This is the only way to achieve high-quality cards that are subjected to high stresses.

### 14.4.1 Monolayer card

A monolayer card structure, which consists of a single 800-µm foil called a monofoil, can be regarded as a simplified version of a multilayer structure. This method is less expensive, but the cards are less durable than cards with a multilayer structure, and they allow significantly fewer options for the design and configuration of the card components. For example, monolayer cards

<sup>7</sup> See also Section 3.3, ‘Card Body’, on page 38



**Figure 14.25** The figure on the left shows a symmetric card structure, while the figure on the right shows an asymmetric card structure. The asymmetric structure is prone to warpage and is thus undesirable

do not have laminated transparent overlay foils to protect printed elements against scratching and rubbing.

#### 14.4.2 Multilayer card

The most elaborate method is to construct the card body from several plastic foils that are thermally bonded. This is called a multilayer structure, and the process of bonding the foils using heat and high pressure is called lamination. The thickness of the core foils used to form the inner part of the card ranges from 100 to 600 µm, while the thickness of the overlay foils ranges from 25 to 300 µm. Card bodies with this structure allow considerable freedom in the design and layout of the card components, are very durable, and also allow security features to be inserted between the foils. For example, this technique is used for the MM technique used with German Eurocheque cards.<sup>8</sup>

The number of layers in a multilayer card can vary over a wide range. Depending on the specific requirements, such cards can have as many as nine layers, as illustrated in Figure 14.26.

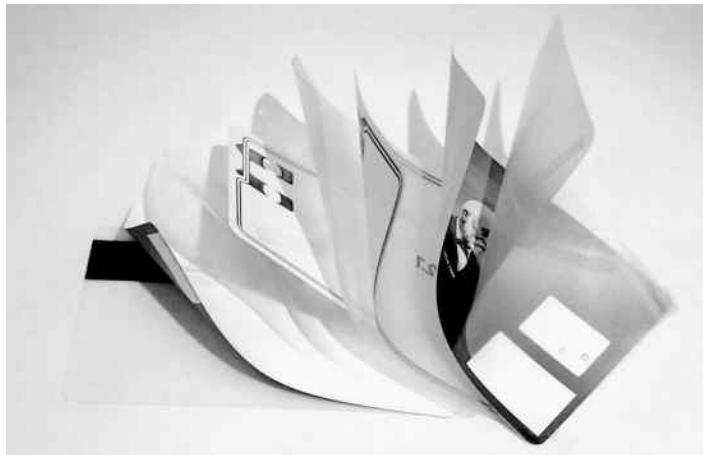
One of the important design requirements for multilayer cards is that they should have a symmetric structure relative to the central layer (see Figure 14.25). This means that the foils on the front and rear surfaces must have the same thickness and be made from the same material, as shown in Figure 14.25, as otherwise there is a good chance of card warpage. This is very similar to the effect observed with bimetallic structures.

An alternative way to produce multilayer cards is to use coextruded foils instead of laminated foils. Coextruded foils are composed of two or three different types of plastic that are bonded together during extrusion while in the liquid or plastic state. The overall thickness of the foil lies in the range of 20 to 200 µm. Foils with various properties can be obtained by using suitable combinations of plastics. Coextrusion makes it possible to reduce the number of layers of complex multilayer cards while still maintaining overall properties nearly the same as multilayer cards produced with conventional methods.

#### 14.4.3 Injection-molded card bodies

The third option for producing plastic card bodies is injection molding (see Figures 14.27 and 14.28). It essentially results in a monolayer card body, with all of its advantages and disadvantages. However, there are some small but significant differences. Most injection-molded cards are made from ABS, since it is well suited to this form of processing.

<sup>8</sup> See also Section 3.5.13, ‘Moduliertes Merkmal’, on page 48



**Figure 14.26** The individual foils of an eight-layer contactless credit card (Reproduced with permission from Giesecke & Devrient)



**Figure 14.27** Example of an injection-molded card body with a two-level cavity and a preformed plug-in card that can be broken free of the main card (CircleSmart Card)

Printed features can be applied to injection-molded cards directly during molding by placing a thin printed foil (thickness approximately 80 µm) in the mold. Although this technique, called in-mold labeling, has certain limitations compared with offset or screen printing in terms of design and printing inks, it has the advantage that the card bodies do not have to pass through a single-card printing machine after molding.

Another feature of the injection molding method is that the cavity for the chip module can be formed in the molding process as illustrated in Figure 14.27, so it does not have to be milled out afterward. In the case of smart cards for telecommunication applications, it is also possible to form a breakaway plug-in or mini-UICC card directly during the molding process. This eliminates the otherwise unavoidable punching step for cards of this sort. Typical injection molding machines can produce around 3000 card bodies per hour with a four-cavity mold, or around 5200 card bodies per hour with an eight-cavity mold.



**Figure 14.28** Example of an injection molding machine for producing card bodies (CircleSmart Card)

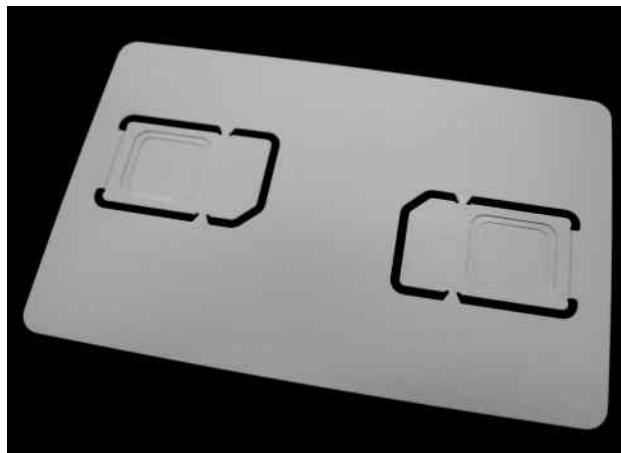
There are also techniques available that allow the chip module to be placed in the mold so that it is implanted in the card body during the molding process. This technique eliminates the need for some of the production steps described below.

With injection molding, relatively little retooling is required to make cards in ID-000 (plug-in) format, as needed for the telecommunication sector, instead of the usual ID-1 format. This increases the throughput of the injection molding machines and thus reduces the manufacturing cost of the card bodies.

#### 14.4.4 Direct plug-in production (plug-in only)

Although telecommunication is by far the largest application area for smart cards and by now only the ID-000 (plug-in) form factor is used in this sector, almost all smart card production lines are designed for the ID-1 card format, since this provides flexibility in line utilization for different application areas. If a uniform card size is used in production, the same machines can easily be used to produce smart cards for payment systems, health care, and so on, instead of only SIMs for mobile telephones. However, a slow trend away from this scenario has appeared in recent years as a result of high price pressure. Overall, producing SIMS by first producing ID-1 cards and then punching them to yield the plug-in format (see Figure 14.29) is slightly more expensive than producing plug-in cards directly. For this reason, there is a shift toward dedicated production machines for plug-in only production. With such machines, injection-molded card bodies are usually used to produce the SIMs.

A possible intermediate alternative to direct plug-in production is to package four plug-in units in each ID-1 card instead of only one. This allows the continued use of existing production equipment. It has the advantage of increasing printing capacity by a factor of four and reducing the unit cost of the card bodies, since four plug-ins can be produced from each ID-1 card. A difficulty with this approach is dealing with defective modules (reject handling) in these four-up cards. It must be possible to identify an individual module in a specific card as defective and exclude it from further production steps. In addition, production of additional four-up cards to replace the defective modules must be initiated. This problem is even more



**Figure 14.29** Two plug-in units in a single ID-1 card body, produced by injection molding. With this technique, card body production capacity and printing capacity can be doubled without any additional expense. With this form of production optimization, the end user receives a card in plug-in format instead of a card in ID-1 format (CircleSmart Card)

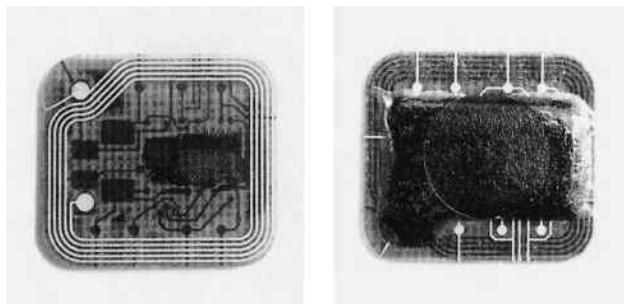
challenging if the delivered cards must be numbered sequentially. In this case, a sorting process is needed after the actual production process, and this sorting process is rather complicated and thus expensive.

In any case, with this form of process optimization it is still necessary to separate the individual plug-in units from the ID-1 cards. This is not necessary if the entire production chain is designed to produce cards in plug-in format directly instead of cards in ID-1 format. The advantage of the latter approach is that the smart cards can be processed end to end in plug-in format without any additional production steps. The cost savings compared with conventional production are only a few cents per smart card, but the investment can be recovered reasonably quickly due to the very large production quantities in the telecommunication sector. The drawback is reduced flexibility, since only plug-in cards for telecommunication applications can be produced on production lines optimized in this manner; they cannot be used to offload production of cards for other types of cards, such as payment cards.

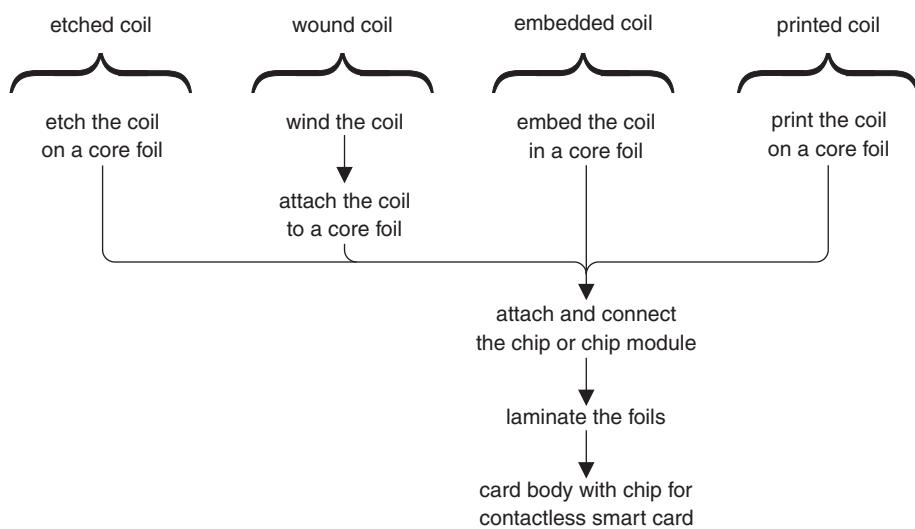
#### 14.4.5 Card bodies with integrated antennas

Contactless smart cards need antenna coils for transferring power and data. At high frequencies in the gigahertz range, these coils can be made so small that they can be integrated in the chip module. Figure 14.30 on the following page shows an example of this sort of module. With contactless cards of this type, the production process is nearly the same as for cards with contacts. The chip module with the coil is simply laminated between two or more plastic foils or inserted in a cavity.

However, most present-day contactless smart cards operate at much lower frequencies, which means they need antennas with larger coils. These coils are usually rectangular with rounded corners and measure approximately  $75 \times 45$  mm. They are thus only slightly smaller than a card body in ID-1 format. These coils normally have four turns, an inductance of



**Figure 14.30** Front and rear views of microcontroller chip module with an integrated coil



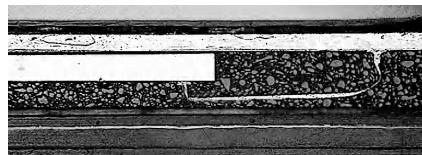
**Figure 14.31** The four usual methods for producing card bodies with antennas and their major process steps

approximately 4  $\mu$ H, and a resistance of a few ohms. Printed coils are an exception; they have a resistance of around 300 ohms.

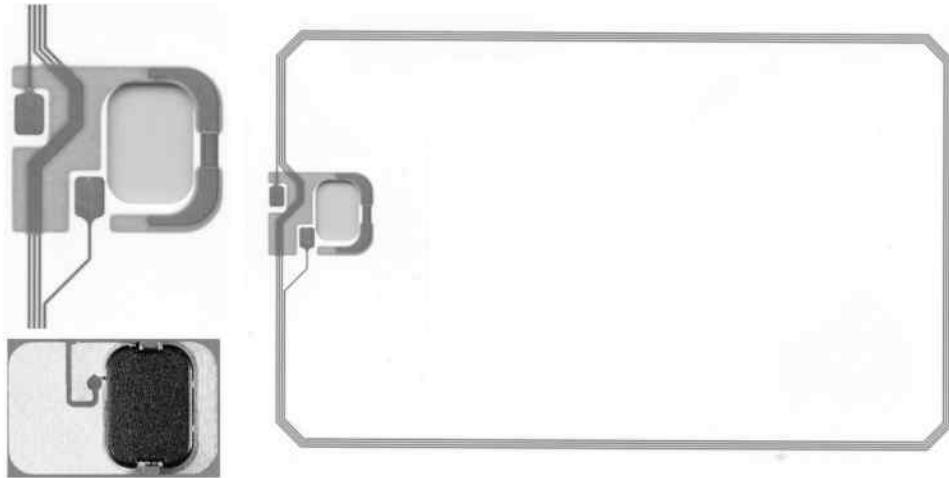
Production of card bodies with integrated coils requires certain changes to the standard production process in order to meet the modified requirements. However, some basic aspects remain the same. For example, both types of cards are usually produced using large sheets of plastic holding 48 cards each instead of producing cards one at a time, since the latter approach would be considerably more expensive.

#### **14.4.5.1 Etched antennas**

As depicted in Figure 14.31, there are several ways to integrate a coil in a card body. In the first process developed for producing contactless smart cards, a plastic foil coated with a 35- $\mu\text{m}$  layer of copper is etched to produce a coil in the copper layer. The width of the etched copper



**Figure 14.32** Cross section detail of a chip module for contactless smart cards implanted in a card body. The bonding wire connecting the chip to the coil is clearly visible (Reproduced with permission from Giesecke & Devrient)



**Figure 14.33** Right: inlay foil for a contactless smart card with an etched coil. Top left: enlarged detail of the module cavity and contacts for connecting the coil to the module. Bottom left: the corresponding module with the encapsulated chip

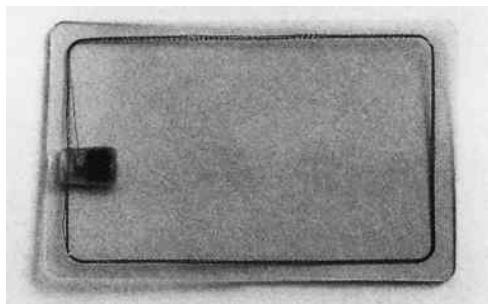
trace is approximately 100 µm. After the coil is etched, a chip is attached next to the terminals of the coil and connected to them (see Figures 14.32 and 14.33). This assembly is then used as a core foil, with overlay foils laminated to its front and rear surfaces. The smart card is now ready. This type of card is called a contactless smart card with an etched coil.

#### 14.4.5.2 Wound coils

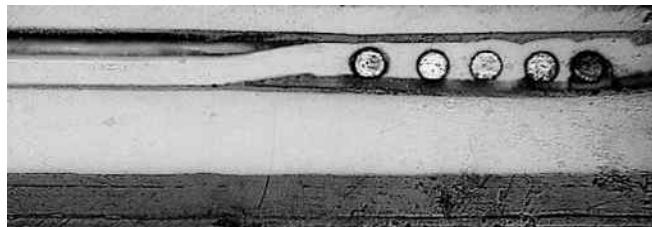
Another way to produce an antenna for a smart card is to use a wound coil (see Figure 14.34). Copper wire with a diameter of 150 µm is first wound on a tapered form and then slid from the form onto an internal foil, to which it thermoplastically welded using heat and pressure. Following this, the chip or module is attached and the foils are laminated.

#### 14.4.5.3 Embedded antennas

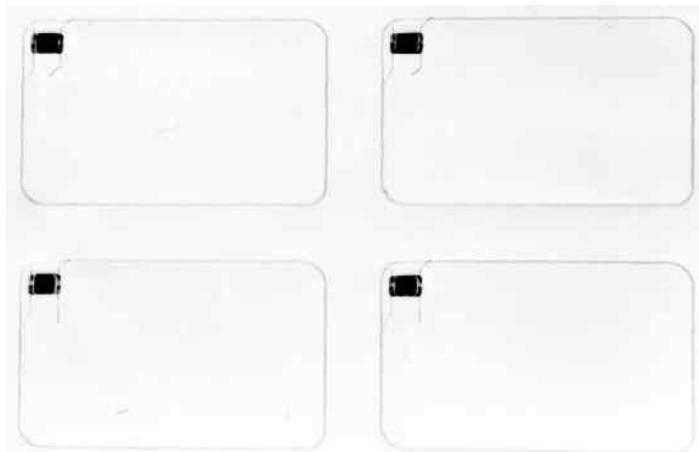
Another technique for producing wire coils is called the embedded coil method (see Figures 14.35 and 14.36). This works in a relatively simple manner. A coil made from 150-µm copper



**Figure 14.34** Example of a wound coil antenna and a lead-frame module in a card body that has been kept transparent for demonstration purposes



**Figure 14.35** Cross-section detail of a contactless smart card with a wound coil embedded in the card body (Reproduced with permission from Giesecke & Devrient)



**Figure 14.36** Antenna coils for contactless smart cards made using the embedded coil technique, located on a multicopy sheet



**Figure 14.37** A printed antenna for contactless smart cards

wire is formed on a plastic foil and simultaneously bonded directly to the foil using ultrasonic welding. This is done using a device called a Sonotrode, which mechanically guides the wire and welds it to the plastic foil. After this, the chip or module is connected to the coil and an overlay foil is laminated on top.

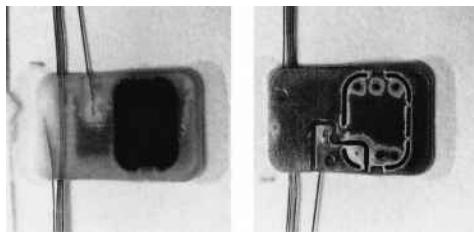
#### 14.4.5.4 Printed antennas

Of all of the possible methods, the printed coil method is the most advanced and the least expensive for mass production. The windings of the coil are produced on an internal foil by screen printing with a conductive ink (see Figure 14.37). Screen printing is suitable for this because it is fairly easy to achieve the required ink thickness of around 50 µm with this method. The resistance of the coil would be too high with a thin ink layer. After the coil has been printed, the chip is connected by die bonding and encapsulated in epoxy resin. The final step is laminating a protective overlay foil on top of this assembly. The main advantage of this method is that it permits high throughput, thanks to the simplicity of the printing process. It is thus exceptionally well suited to producing large numbers of cards. However, considerable expertise is necessary to achieve the necessary quality in antenna printing and die bonding.

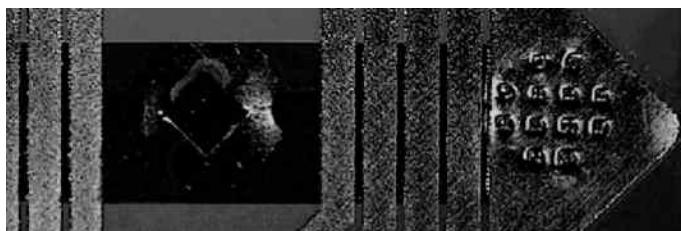
#### 14.4.5.5 Connecting the antenna to the chip

In the currently standard technology, the die to be connected to the coil is attached to a lead frame<sup>9</sup> and electrically connected to two contacts of the lead frame (see Figure 14.38). The reason for using a lead frame is that it relaxes the accuracy requirements for positioning the connections to the antenna coil, compared with direct connection to the coil. However, this technique is more expensive than direct connection of the die to the coil.

<sup>9</sup> See also Section 3.6.4, ‘Lead-frame modules’, on page 57



**Figure 14.38** Front and rear views of a lead-frame module for a contactless smart card, with connections to a wound coil



**Figure 14.39** Detail of the electrical connection between a small memory chip and an etched coil using die bonding

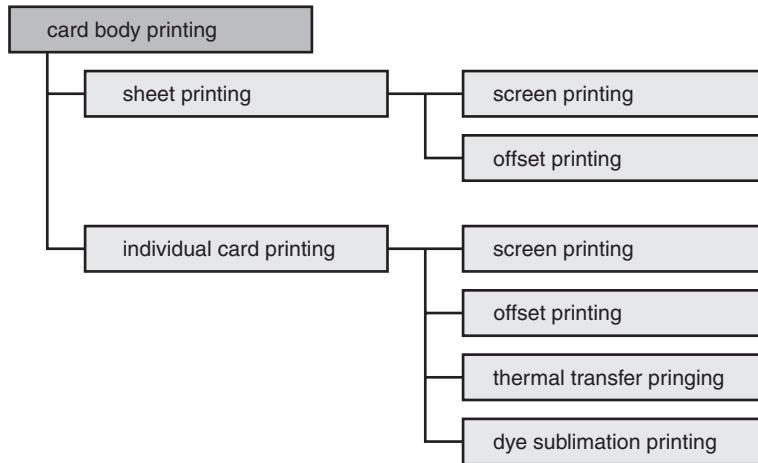
In the case of antennas integrated in the card body, there are two options for connecting the chip to the coil or the lead frame. With the widely used wire bonding method, the chip is connected by fine bonding wires. A significantly more elegant and less expensive solution is die bonding, in which the chip is pressed against the lead frame or coil to make direct electrical contact and then glued in place (see Figure 14.39). This requires the chip to be flipped over (relative to the wire bonding technique), so this method (and sometimes the chip itself) is called flip-chip. With both of these methods, the chip is covered by a protective layer of plastic resin after the connections are made.

#### 14.4.6 Printing the card bodies

The printed surfaces of a card are perceived directly by the user. In many applications, the printing portrays the brand concerned, and it is often used to provide information to other parties, such as with ID cards. In case of cards with personalized printing, the printing also expresses the individuality of the person concerned. For all these reasons, high-quality printing on smart cards is an important aspect of production, even with cards that spend many years concealed in mobile telephones as plug-in cards. As shown in Figure 14.40, various methods are used to print card bodies. Their key characteristics are listed in Table 14.3.

##### 14.4.6.1 Sheet printing of card bodies

With regard to printing the card body, there is usually no difference between multilayer and monolayer cards. In the sheet printing process, large sheets of plastic are printed with multiple

**Figure 14.40** Basic methods for printing card bodies**Table 14.3** Summary of the most commonly used card printing methods

Properties	Offset printing	Screen printing	Thermal transfer	Thermal dye sublimation
Sheet printing possible for mass production	yes	yes	no	no
Resolution	very good	moderate	good	good
Surface coverage of the ink or dye	good	very good	satisfactory	satisfactory
Printed surface suitable for lamination	no	yes	uncommon	uncommon
Printing scratch-resistant	no	only with a laminated overlay foil	no	yes
Card-specific printing	no	no	yes	yes
Cost	low	low	high	high

copies of the card body on each sheet (see Figure 14.45), and the card bodies are then stamped from these sheets. The multicopy sheets are usually large enough to hold 21 to 48 card images on each sheet, which is fed one or more times through the individual inking stations of an offset or screen printing machine. The front and rear surfaces of the card bodies must be printed separately.

#### 14.4.6.2 Printing single card bodies

The alternative to sheet printing of card bodies is single-card printing, in which the cards are printed one at a time. If the cards are printed singly, this always takes place before the cavity is milled. The throughput of single-card printing machines ranges up to 12 000 cards per hour. Other forms of single-card printing are thermal transfer or thermal dye sublimation printing



**Figure 14.41** An offset printing machine for sheet printing

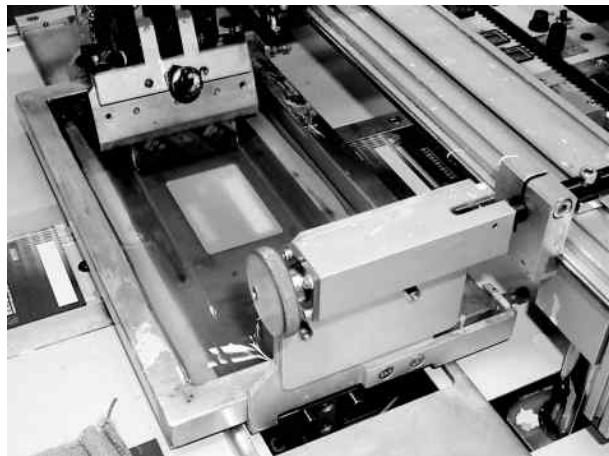


**Figure 14.42** A row of five single-card screen printing machines

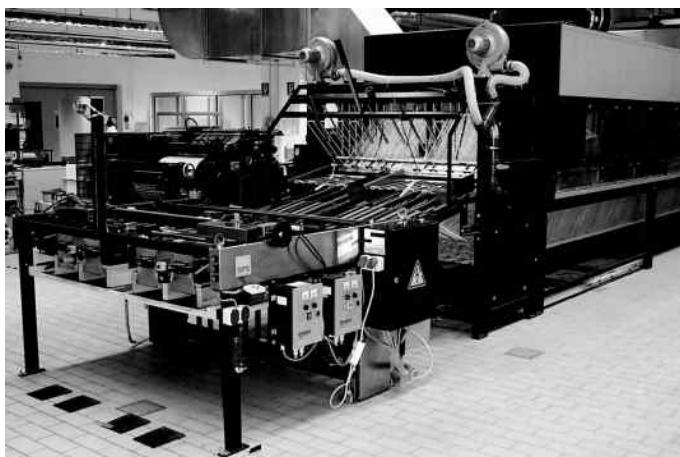
in desktop personalization machines. The throughput of these machines is significantly lower (around 300 cards per hour).

#### **14.4.6.3 Offset printing**

The two basic card printing methods are offset printing (see Figure 14.41) and screen printing (see Figures 14.42, 14.43 and 14.44). Finer details can be printed on the card with offset printing than with screen printing. In addition, the inks used in offset printing are dried under ultraviolet light. This occurs immediately after the actual printing operation, which has the advantage that the printed cards can be stacked directly after printing. However, it is not possible to affix holograms or magnetic stripes on top of UV-cured coatings using the hot-stamp method because these coatings are not thermoplastic and hot stamping requires a thermoplastic substrate.



**Figure 14.43** Detail of a single-card screen printing machine. The screen is in the middle, with the squeegee at the top left



**Figure 14.44** A screen printing machine for sheet printing, which can print multicopy sheets with 48 card bodies on each sheet as shown in Figure 14.45 on the following page

Offset printing is well suited to color images with high resolution and large production quantities. However, if additional functional elements such as holograms must be permanently affixed to the surface of the card, either the entire card or the substrates of these elements must be printed with a thermoplastic ink using screen printing.

#### 14.4.6.4 Digital printing

Digital printing encompasses all printing methods that do not use a static printing form, such as printing plates with offset printing or screens with screen printing. With digital printing, large



**Figure 14.45** A stack of printed multicopy sheets, each holding 48 card bodies

numbers of items can be printed directly from a computer, just as documents can be printed with a laser printer. Digital printing is used for smart card production in the same way as offset printing, and it has the same properties as offset printing. It can be used for sheet printing or single card printing. The main advantage of digital printing is that it allows personalized data and imagery to be printed on the cards at no additional effort or expense.

However, reprinting of individual cards can be difficult in some cases if personalized cards are produced using sheet printing because the cost is prohibitive. Consequently, digital sheet printing is primarily used with small production quantities.

#### **14.4.6.5 Screen printing**

Internal foils for laminated card bodies must be printed using screen printing because the lamination process requires the foils to be thermoplastic. With screen printing, the inks dry by evaporation of a solvent and remain thermoplastic. Consequently, additional card components can easily be laminated onto screen-printed surfaces. In practice, offset printing and screen printing are often used in combination. For example, large single-color areas and the substrates of the magnetic stripe and hologram are first screen printed, and the details are applied in a second step using offset printing, since screen printing cannot achieve such a fine level of detail.

In case of multilayer card bodies produced using sheet printing, the foils are laminated at 100 to 150 °C and the necessary features are integrated. The printed foils are protected against scratching and wear by supplementary transparent overlay foils laminated on the front and rear surfaces. Depending on the requirements, elements such as signature panels, magnetic stripes, and security features can be embedded or laminated in multilayer cards in this production step.

Cards can be produced with different foil thicknesses on the front and back surfaces, but for mechanical reasons it is generally better to use a symmetrical structure. This means that foils with the same thickness should be used for the front and rear of the card. This avoids potential problems with card warpage due to a ‘bimetallic’ effect.

#### **14.4.6.6 Thermal transfer and thermal dye sublimation printing**

Another printing method is thermal transfer printing, which is typically used for single-card printing and can also be used for printing personalized cards. With this method, pieces of colored foil are heated to release them from a substrate and applied to the card body, which is usually white. The colored foils bond to the surface of the card. A monochrome

(black-and-white) version of this method is used to apply serial numbers to cards, but it can also be used to reproduce all colors and color gradients. However, it is slow and expensive, so it is used primarily in desktop personalization machines for producing small quantities of cards. The resolution is typically 300 dpi. A disadvantage of this method is that the applied colors are only bonded to the surface of the card, so they can be scratched off.

This drawback can be avoided by using thermal dye sublimation printing. With this method, a print head with a temperature of nearly 200 °C presses hot dye into the top plastic layer of the card body. The maximum penetration depth is 5 µm, which is sufficient to make the printing scratchproof. Thermal dye sublimation printing otherwise has essentially the same characteristics as thermal transfer printing, including high cost, so it is primarily suitable for small quantities of cards. These two methods are definitely worthwhile additions to just-in-time printing for small and medium-sized card quantities.

#### 14.4.6.7 *Inkjet printing*

Inkjet printers are sometimes used to apply serial numbers to cards with modest printing quality requirements. The advantage of this method is that very high throughputs, extending to 40000 cards per hour, can be achieved with some machines. This method can also be used to mark cards with fairly simple characters that are only visible under ultraviolet light.

#### 14.4.7 Stamping the foils

After the individual plastic sheets have been laminated, they must be converted into single cards. This is done by stamping the cards from the sheets. The stamping machines have an hourly throughput of 4000 to 8000 cards. The burr that can be seen or felt on the edges of some cards is due to worn stamping tools.

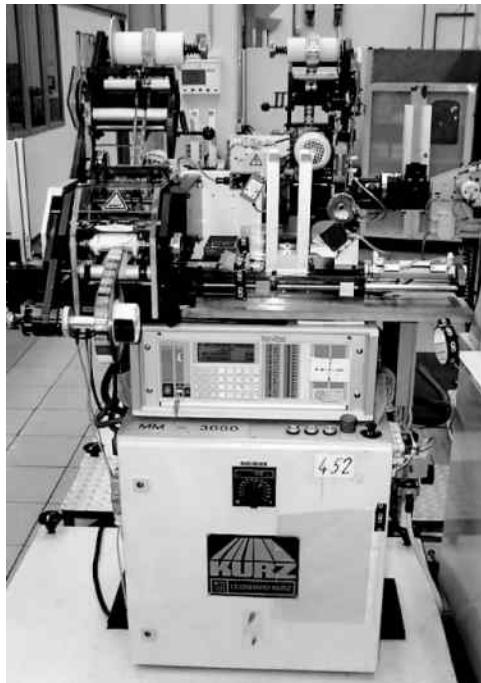
#### 14.4.8 Applying card components to the card body

After the card bodies have been stamped out, various elements such as holograms and magnetic stripes are applied to them. Holograms, which are supplied in rolls, are permanently attached to the card body by thermal bonding using the hot stamp or roll-on method (see Figure 14.46).<sup>10</sup> Any subsequent attempt to remove this card component will destroy it. Magnetic stripes are also applied to the card bodies by lamination or hot stamping.

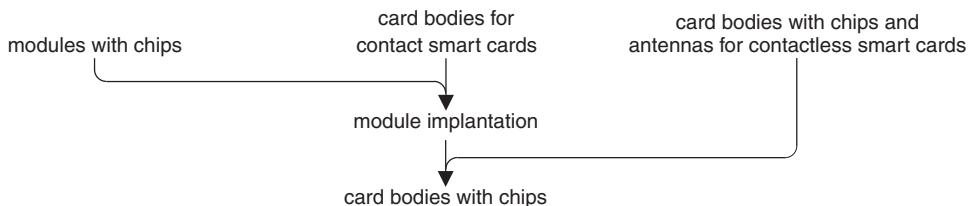
### 14.5 COMBINING THE CARD BODY AND THE CHIP

The final step in the production process is implanting the modules from the semiconductor manufacturer or module manufacturer in the prefabricated card bodies from the card producer (see Figure 14.47). Mechanical factors are the primary consideration in this step. A certain amount of expertise is needed to durably fit modules into the cavities of the card bodies. This is not as easy as pasting clippings into a scrapbook.

<sup>10</sup> See also Section 3.5, ‘Card Components and Security Features’, on page 42



**Figure 14.46** A machine for applying holograms to card bodies using the hot-stamping method

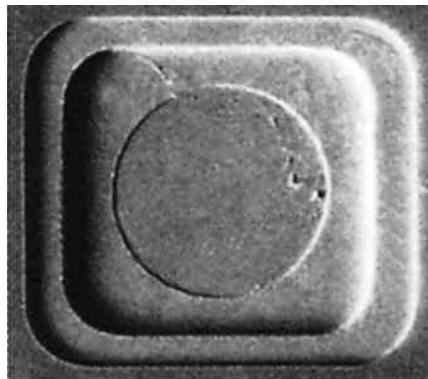


**Figure 14.47** Process steps for implanting modules in card bodies for contact smart cards

### 14.5.1 Milling the module cavity

After the card body has been produced, a recess for the module must be milled in the card. This recess is called a cavity (see Figure 14.48). There are also processes in which the foils are pre-punched to provide a recess for the module when the card body is laminated, but they are rarely used. With injection-molded card bodies, the cavity is formed during the molding process. However, with monolayer card bodies the module cavity is usually produced by milling.

The rear surface of the module has a bump containing the encapsulated die, so a matching recess must be milled in the card body. A single-level cavity is generally unsatisfactory with modern types of modules, so the cavity is usually milled with two or even three levels. This provides a larger contact surface between the card body and the module, so the module can be durably bonded to the card body. In mechanical terms, it is also significantly better if only the



**Figure 14.48** Example of a milled module cavity in a card body

rim of the module is bonded to the card body, with no physical contact between the die on the back of the module and the card body. This is called floating module implantation.

The first step in making the cavity is milling a recess with the same size as the contact layer of the module and the same depth as the contact layer. Following this, another recess is milled in the middle of the first recess to provide room for the encapsulated die. The result is a two-level cavity.<sup>11</sup>

The milling must be performed very precisely, since the thickness of the remaining card material below the deepest part of the cavity is only 0.15 mm. If the milling machine vibrates or rocks, the card body could be milled all the way through and thus be rendered useless. If the cavity is not deep enough, the module will stand proud of the surface of the card, which is only allowed within very narrow limits. This tricky production step is performed by fully automated machines, with the card bodies fed in from one bin and passed out to another bin (see Figures 14.49 and 14.50). The throughput of a single machine is around 6000 cards per hour.

### 14.5.2 Implanting the modules

Regardless of the method used to produce the card body and form a cavity for the module, the module must be implanted in the card body in the next step of the production process.

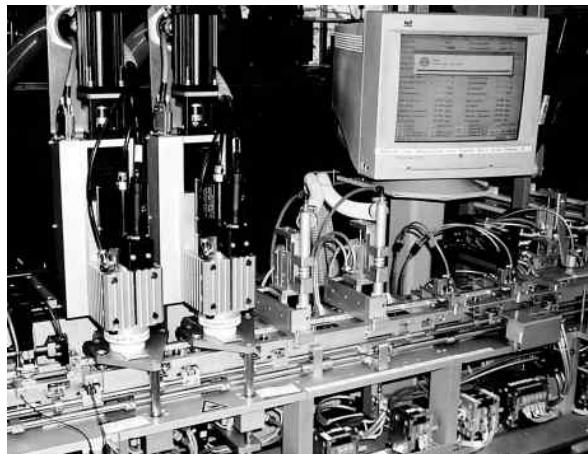
A piece of glue tape with hot-melt glue on both sides is normally used to attach the module to the card body. Only the support surface at the rim of the module is glued to the card body, with the encapsulated die in the middle of the module remaining free, so the module floats in the card body. To achieve this, the glue tape is first stamped to the right shape and then applied to the modules on the 35-mm carrier tape so it covers only the rims of the modules (see Figure 14.51). The individual modules are then separated from the carrier tape and glued into the card bodies with the attached glue tape. The durability of the glue bond depends on the proper combination of heat, pressure and time.

A difficulty with the gluing process is that the modules are heated to around 180 °C for a few seconds. The modules can be destroyed by overheating if this hot-gluing operation, which requires a certain amount of expertise, takes too long. In any case, this brief heating

<sup>11</sup> See also Section 3.6, ‘Chip Modules’, on page 50



**Figure 14.49** A machine (Ruhlamat MS-310) for milling cavities in card bodies (Reproduced with permission from Ruhlamat)

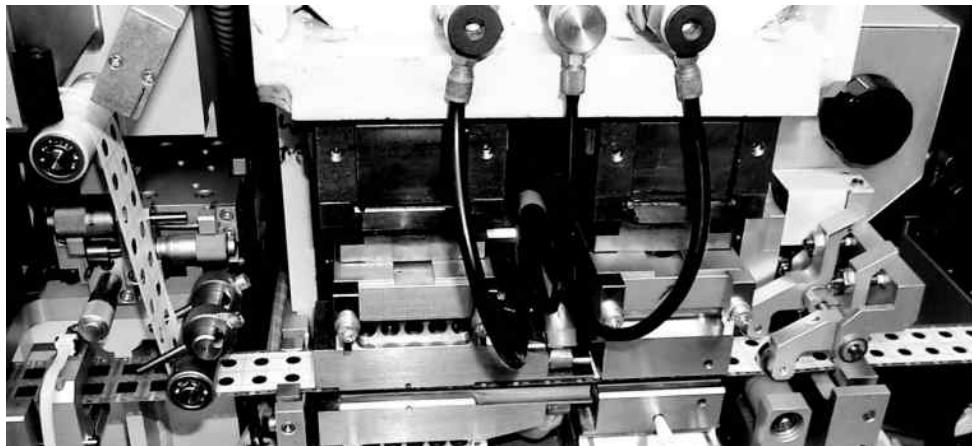


**Figure 14.50** A machine for milling cavities in card bodies. The two upright components on the left are electrically driven milling spindles with integrated swarf extraction

artificially ages the chips, but this normally does not have any negative consequences. The implanting machines used in card production (see Figures 14.52, 14.53, 14.54 and 14.56) can process around 6000 modules per hour, which corresponds to a cycle time of 1.6 s for the implanting operation.

Other attachment methods, such as liquid cold-setting glues, are also used (see Figure 14.55), but hot-gluing is still regarded as very reliable. The main problems with liquid glues dispensed into the milled cavity are glue curing and the lack of a clearly defined bonding surface.

Once the module has been implanted in the card body and all the nonpersonal features and printing have been applied to the card, mechanical production of the smart card is complete.



**Figure 14.51** Detail of a machine for applying glue tape to modules on a 35-mm tape. The module tape is fed in from the left, while the glue tape is fed in from the top left and applied to the modules in the middle



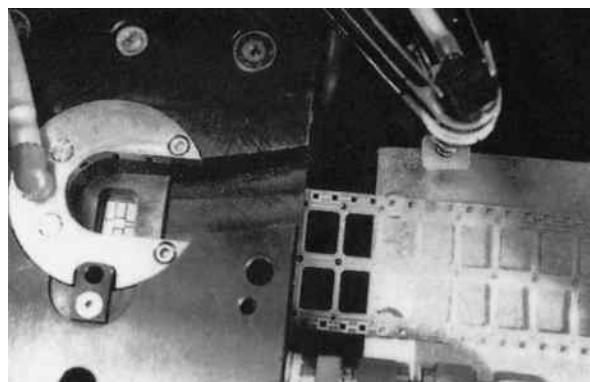
**Figure 14.52** A rotary-table machine for implanting chip modules. Modules arranged in pairs on 35-mm tape are fed in along the front, punched free, and placed in the card body cavity by a suction gripper

### 14.5.3 Module printing

A relatively uncommon design option that is nevertheless quite attractive for some applications is color printing on modules. This does not involve card-specific printing, which must also satisfy certain conditions. Electrically conductive inks must be used to ensure that electrical connections can still be made between the contacts on the smart card and the terminal. Bridging of the insulating regions between the individual contacts must also be avoided, as otherwise short circuits will occur. In addition, the inks must have high scratch resistance, as otherwise the printing will quickly become unsightly.



**Figure 14.53** A combo machine (Ruhlamat MS-330) that automatically punches the module from the tape, mills the cavity in the card body, and implants the module in the card (Reproduced with permission from Ruhlamat)

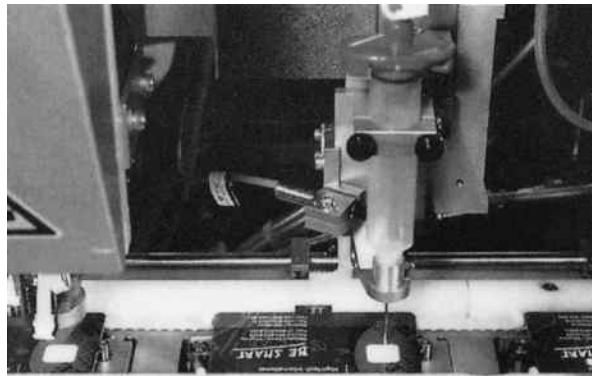


**Figure 14.54** Punching a module from the tape in an implanting machine. The punch can be seen on the left, and the suction head that transports the punched-out module to the card body can be seen on the right (Mühlbauer)

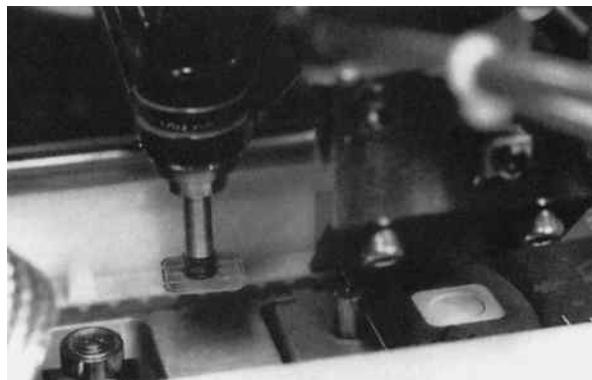
#### 14.5.4 Plug-in stamping

Only smart cards in plug-in format are used in modern mobile telephones. Some network operators are already issuing plug-in cards that can be converted to mini-UICC format by breaking the smaller form free from the card.<sup>12</sup> As the larger ID-1 format is used almost exclusively in conventional smart card production, an additional production step is necessary with mobile telephone cards. This step consists of stamping individual ID-1 cards such that card

<sup>12</sup> See also Section 3.1, 'Card Formats', on page 29



**Figure 14.55** Dispensing liquid glue into a milled module cavity in an implanting machine (Mühlbauer)

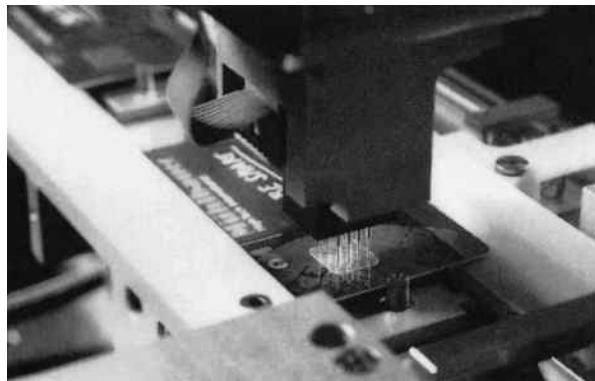


**Figure 14.56** Inserting a module in a module cavity in an implanting machine (Mühlbauer)

users can later convert them to plug-in or mini-UICC format by breaking the smaller format free from the ID-1 card body (see Figure 14.58). With the relatively recent injection molding method, the smaller format can be preformed in the molded card body, which eliminates the need for the punching step in the production process.

## 14.6 ELECTRICAL TESTING OF MODULES

The first production step in this phase is an electrical test of the smart card. A basic test is performed by generating an ISO smart card activation sequence, to which the card must respond with a valid ATR. If an ATR is received and it meets expectations, it is certain that at least the core of the microcontroller is working properly. This is followed by special tests of the hardware components, such as the ROM, EEPROM/flash memory, and RAM. Special



**Figure 14.57** Electrical testing of an implanted module in an implanting machine (Mühlbauer)

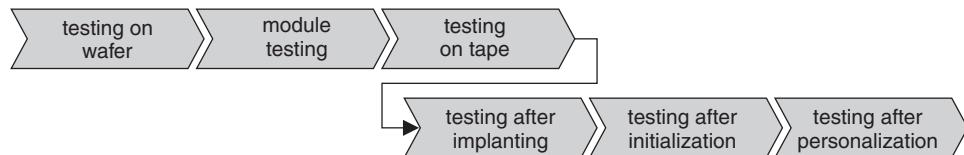


**Figure 14.58** A stamping machine (Ruhlamat MS-340) for forming plug-in cards in card bodies  
(Reproduced with permission from Ruhlamat)

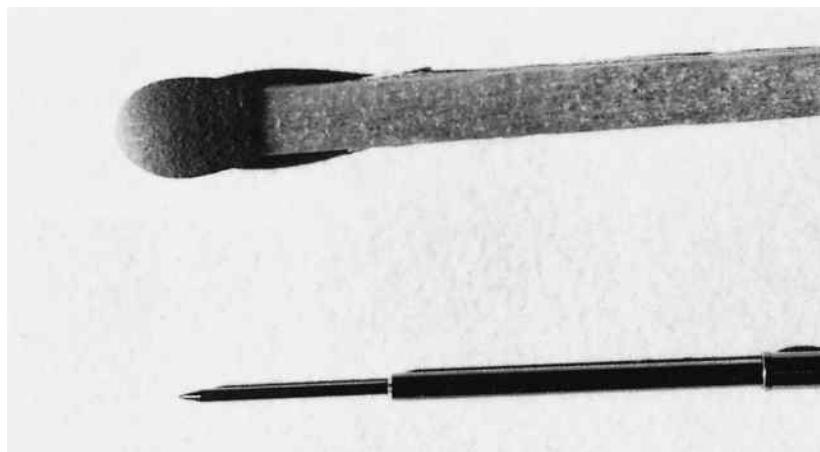
machines (see Figures 14.57, 14.61, 14.62 and 14.63) that can process several cards at a time are used to achieve high throughput with these tests, some of which can take up to several seconds. High-performance machines have a throughput of up to 100 000 cards per hour.

The preferred way to test the EEPROM is to write a checkerboard pattern, such as 'AA' (1010 1010) and '55' (0101 0101), to the individual bytes. As this would take a long time with large EEPROMs, a special technique is sometimes used to shorten the test. Instead of using the specified EEPROM page write time, such as 3.5 ms per page, a shorter time is used (such as one-tenth of the specified time – 350 µs in this example). The data retention time of the EEPROM is only a few minutes when such a short write time is used, but this does not create any problems in this case because the checkerboard pattern is checked a few seconds after the data is written.

The advantage of this dynamic EEPROM programming is that it significantly speeds up testing without reducing the quality of the test. This technique is also used occasionally when



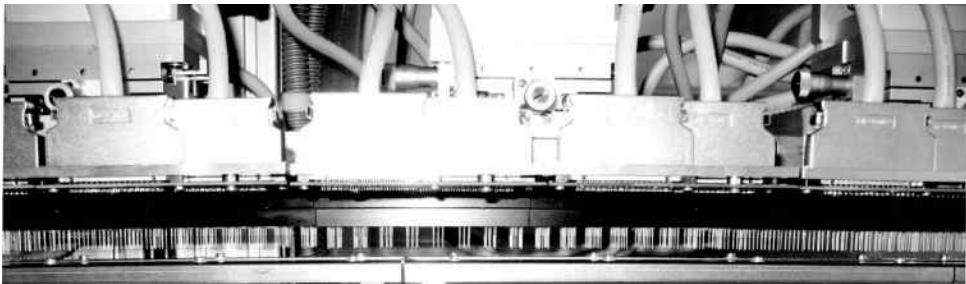
**Figure 14.59** Electrical testing of smart card microcontrollers in the production process. Wafer tests are performed by the semiconductor manufacturer, while module tests are performed by the module implanter. The remaining tests are performed by the card producer



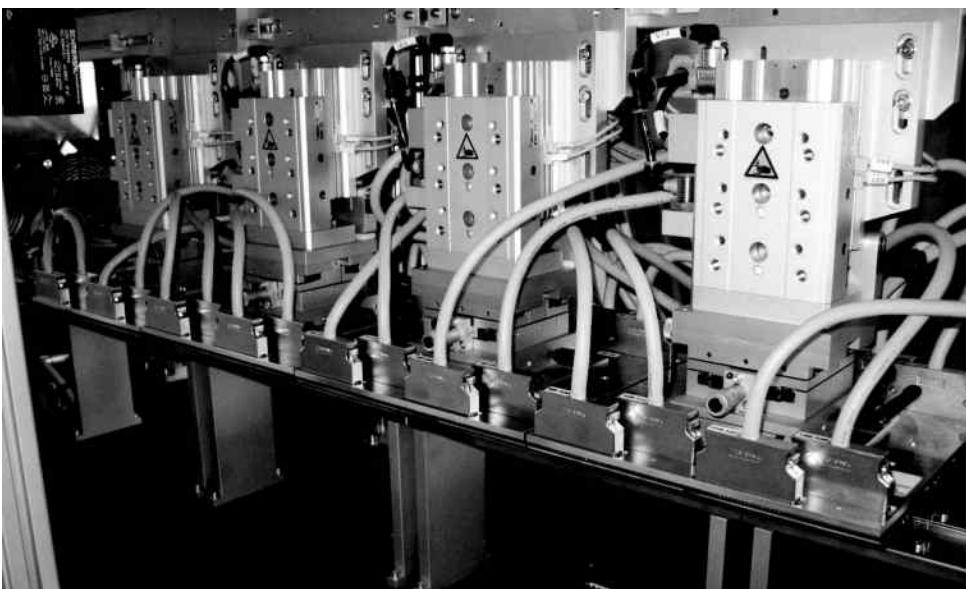
**Figure 14.60** Detail view of a spring-loaded probe needle for the contact unit of a high-performance smart card tester. Although the shape of the probe needle does not conform to the ISO/IEC 7816-2 standard, it is often used in production due to its reliability



**Figure 14.61** A massively parallel tester (Mühlbauer CMT 6530) for chip modules on tape (Mühlbauer)



**Figure 14.62** Detail view of a the tester shown in Figure 14.63 with a contact head that can communicate simultaneously with 128 chip modules on tape (Mühlbauer)



**Figure 14.63** The 128-module contact head of the module tester shown in Figure 14.61 on the preceding page (Mühlbauer)

the transmit and receive buffers of the I/O manager are located in EEPROM instead of RAM. In this case, the reduced write time yields a marked increase in the effective data transmission rate.

There is another interesting technique that is used in electrical testing. In order to reduce the time required to load data during subsequent production steps, a final test pattern (such as '00') is written to the entire EEPROM using the normal EEPROM write time. As the value already stored in EEPROM is known in the subsequent steps (completion, initialization, and personalization), only the data that differs from this value has to be written. A similar technique is to set the contents of the EEPROM to a value that makes it unnecessary to first erase the page to be written before performing subsequent write operations. Both of these techniques distinctly reduce the time needed to perform subsequent production steps that involve writing data to EEPROM.

## 14.7 LOADING STATIC DATA

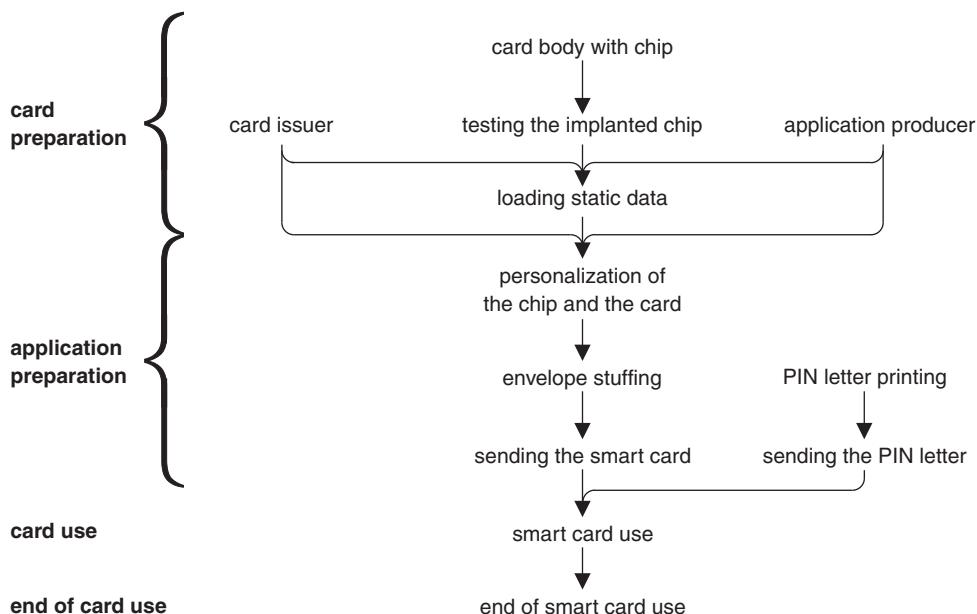
According to the ISO 10202-1 standard, the card preparation phase of the smart card life cycle includes not only implanting the chips in the prepared card bodies, but also loading all data that is not card-specific (see Figures 14.64 and 14.65). This phase and the next phase (application preparation) are often carried out by the same company, although for security reasons they are usually separated physically and organizationally.

A production planning and control (PPC) system is often used to coordinate these complex production processes. The various production machines receive their order data and processing data from this system, and they report current production status to a central control station. This allows mass production of smart cards to be managed to minimize the production time and cost. An additional benefit of a PPC system is that the networking of the production equipment enables near real-time evaluation of all quality assurance and test data.

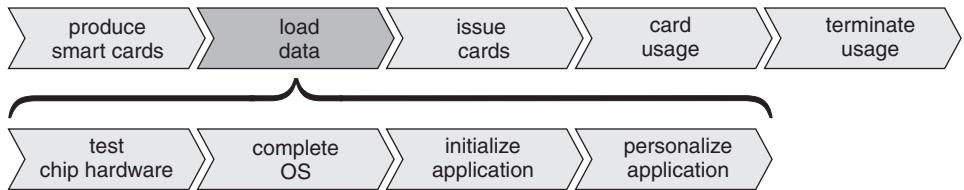
### 14.7.1 Completing the operating system

Many smart card operating systems are only partially located in the mask-programmed ROM of the smart card. The link tables and some portions of the program code are loaded into the smart card EEPROM after authentication using a secret key. The process of loading the EEPROM portion of the operating system is called completing the operating system.

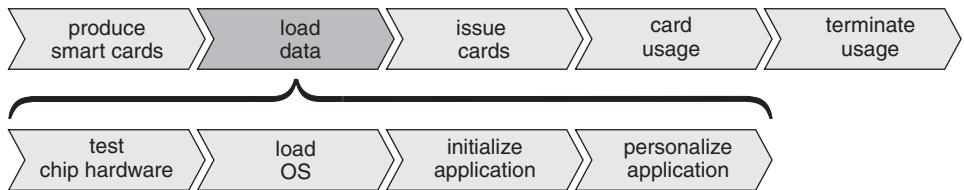
This approach allows minor changes to be made in order to correct errors or expand the functionality of the program code in ROM without being forced to generate a new ROM mask.



**Figure 14.64** Card testing, loading static data, card initialization and personalization (individualization), smart card usage, and termination of card usage in the life cycle phases from card preparation to the end of card usage



**Figure 14.65** Loading data into a smart card with a ROM-based operating system as part of the overall smart card life cycle. If more than one application must be loaded, the last two steps of this process are repeated for each application



**Figure 14.66** Loading data into a smart card with a flash-based operating system as part of the overall smart card life cycle. If more than one application must be loaded, the last two steps of this process are repeated for each application

The operating system in the smart card is not complete until the operating system data has been written to EEPROM. After this, all application commands, such as SELECT and READ RECORD, can be executed.

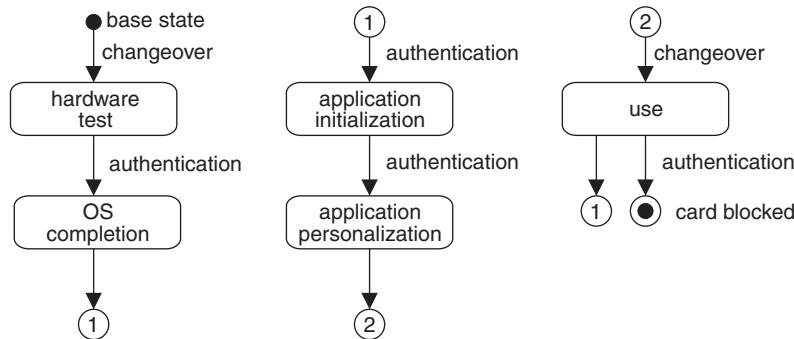
As with the incoming tests, completion (which involves data that is the same for all cards in a particular application) is performed using high-throughput machines that process several cards in parallel.

If the smart card microcontroller has flash memory instead of ROM, the entire operating system is loaded into nonvolatile memory at this point, instead of only the completion portion (see Figure 14.66). Any necessary adjustments to the operating system can be performed directly at the same time. The necessary boot loader is already present in the microcontroller in a small ROM, which is primarily provided to hold hardware test routines. The boot loader is used to load the operating system in this production step.

### 14.7.2 Collaboration of the card producer and the card issuer

The card issuer or application provider sends the card personalizer all the data related to its application. This includes data such as the name of the application, the structure of the file tree, the necessary files, and the file structures. This data is loaded into the cards during initialization. In addition, the personalizer needs all customer-specific and system-specific data, such as secret card-specific keys and the names and addresses of the cardholders. This data is transferred on data storage media or by data telecommunication.

Personalization data is almost always sensitive with regard to security, so the transport path and handover are suitably protected (the data is encrypted). The associated decryption key is transported separately to the personalizer. As a result, the personalization data is worthless if



**Figure 14.67** Smart card operating system state machine for implementing a five-phase life cycle

it is lost because it cannot be decrypted without the key. This sensitive data is always stored and processed inside hardware security modules (HSMs), whose communication with each other and the smart cards is cryptographically protected.

Some smart card applications do not involve transferring card-specific data. The best-known example is SIM cards for the GSM mobile telecommunication system, which are not made for a particular card producer, but instead contain only individual data and keys. The data sets needed for this purpose are usually generated directly by the card producer, and part of this data is reported back to the application operator so the latter knows which cards have been produced. The data that is reported back is called the response data. Network operators use this data to activate the smart cards produced.

Data transfer occurs only to the extent that the card producer receives the data common to all cards and the initial and final values of the card-specific data. The data sets for the individual cards to be produced are then generated in hardware security modules located in the production facility.

The logical interface between the card producer and the card issuer is the profile description, which is often simply called the profile (see Figure 14.68). This document, which is usually an electronic document in XML format, describes all the microcontroller-related data that is necessary for the personalization of a smart card. This includes the configuration of the smart card operating system, the file tree with all its files and their contents, and if appropriate the application program code to be loaded in the smart card. This document also forms the basis for the contractual relationship between the card producer and the customer.



**Figure 14.68** One of the first steps in the smart card production process chain is profile specification

Sample cards that fulfill all the requirements of the profile are produced from the profile description. If these sample cards are only intended to be used to assess the information technology characteristics of the microcontroller, white card bodies are often used for this purpose. Otherwise card bodies with the graphic layout of the subsequent mass-production cards are used. These sample cards are called engineering samples. If necessary, they can also be used to check prior process steps such as data generation.

If these sample cards do not fulfill all the requirements, the necessary changes must be made and a new cycle of the engineering sample loop must be initiated. If they do fulfill the requirements, the next step is send the card producer notification of approval for producing the release samples.

The release samples are produced shortly before the start of volume production using the same production line and production process as will later be used for volume production. The intention here is to produce the release samples under the same conditions as the later mass-production cards, so that any production problems that may exist can be recognized before the start of mass production. The release samples usually consist of only three to five cards, which the card producer checks against the profile description. The functionality of the operating system is not checked at this point, since this would be too complicated. The card issuer receives the test report, and the release for volume production is issued after the card issuer gives its approval. This means that, depending on the order, several million cards may be produced based on the profile with only a few random sample tests for each order. For this reason, the engineering sample and release sample process is carried out with great care and in part using the ‘four-eyes’ principle (two people acting jointly).

Although the details of the implementation of this process vary from one card producer to the next, the basic steps are usually quite similar, which means that card issuers can generally expect the processes to be largely similar.

### 14.7.3 Initializing the application

Completion provides the software basis necessary for the next production step, in which all application data that is the same for all smart cards in a particular application is loaded in the cards. This consists of the application data that does not vary from card to card and the data that is not person-specific, which is also the same for every smart card. This step is called initialization.

At the file level, initialization consists of creating all necessary files (MF, DFs and EFs) and filling them as much as possible with the application data. In many cases, the file contents are defined by the applicable specifications such as TS 51.011. With modern operating systems, the CREATE, UPDATE BINARY, and UPDATE RECORD commands are often used for initialization. This is the last processing step in which all smart cards are treated the same. Consequently, initialization can be performed using fast parallel machines. Card-specific application data and personal data are not loaded into the smart card until the following step, which is called personalization.

The objective of this distinction between common or global data and specific or personal data in the production process is to minimize process costs. Personalization machines that can write specific data to individual smart cards under the required security conditions are technically complex and have throughputs of around 1200 to 6000 cards per hour. They are also usually equipped with relatively slow labeling units for the card bodies. This results in high unit costs for loading data into the cards. Consequently, producers always try to load all global data, which does not differ from card to card, into the cards using simpler and faster initialization machines.

The bottlenecks for initialization as well as personalization are transferring the data to the card and writing it to EEPROM. For technical reasons, it is presently not possible to reduce the write access time of EEPROM. However, the time required to transfer the initialization and personalization data can be reduced dramatically by increasing the clock rate and reducing

**Table 14.4** Elapsed production time for initializing smart cards at various data transmission rates. The underlying assumptions and conditions are described in the text

Data transmission rate	9600 bit/s	38400 bit/s	115200 bit/s
EEPROM write time	3584 ms	3584 ms	3584 ms
Data transfer time	5870 ms	1468 ms	489 ms
Resulting cycle time	10454 ms	6051 ms	5073 ms
Resulting production time	90.7 days	52.5 days	44.0 days

the divisor value. For example, many initialization and personalization machines operate at data transmission rates up to 625 kbit/s instead of the usual smart card rate of 9600 bit/s. This considerably reduces the initialization or personalization time.

The following example is intended to illustrate that even small timing improvements can be worthwhile in the mass production of smart cards. Here we assume that one million cards are to be initialized with 4 KB (4096 bytes) of data each, using two initialization machines operating in two shifts (16 hours per day). We also assume that initialization is performed using 40 commands and the  $T = 1$  transmission protocol is used with 12 data bits per byte of transferred data. In addition, the EEPROM write cycle time is 3.5 ms for a 4-byte page, with no need for prior erasing. The transport time on the initialization machines, which do not have terminals for parallel processing, is 1 s per smart card, and any setup time that may be necessary (such as for loading or unloading bins) is not taken into account. The resulting cycle time is thus the sum of the EEPROM write time, the data transfer time, and the transport time.

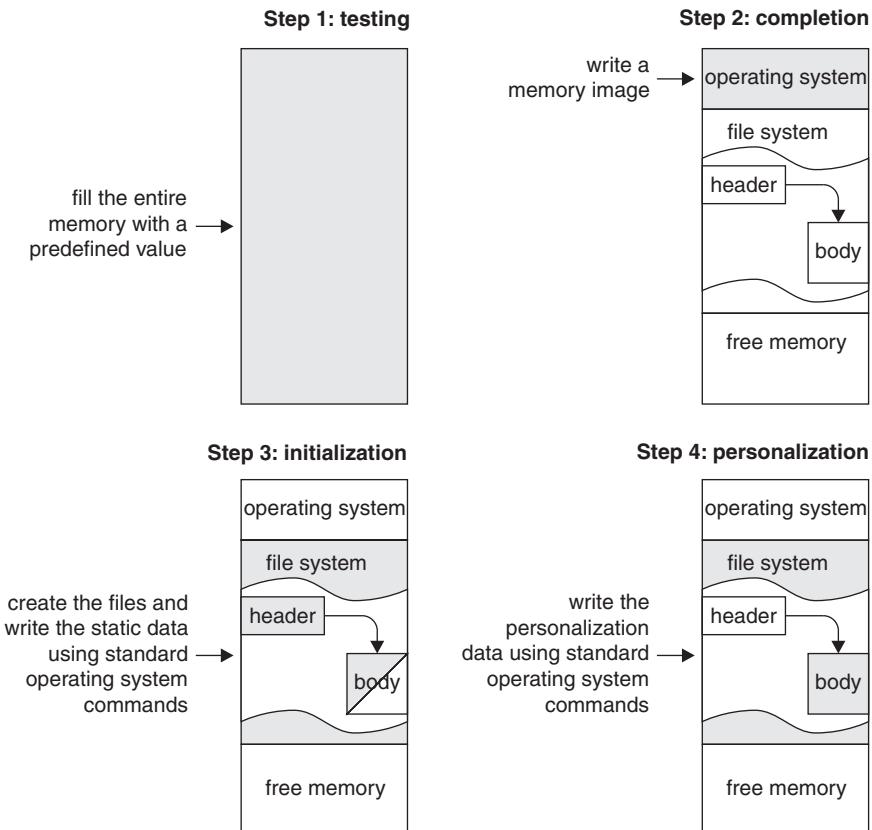
Using the formulas in Section 11.17.1, ‘Processing time estimation’, on page 407 with a data transmission rate of 9600 bit/s, we obtain an elapsed production time of 90.7 days (see Table 14.4). If the data transmission rate is raised to 38.4 kbit/s, the time required to produce one million cards drops to 52.5 days. A data transmission rate of 115 kbit/s would be ideal because it would allow production to be completed more than 46 days earlier than with a rate of 9 600 bit/s.

From this example, it is clear that especially when a large amount of data must be stored in the smart cards, it is worthwhile to invest time and effort in process optimization. The described increase in the data transmission rate depends on the smart card operating system and the underlying hardware. However, most current cards support a transmission rate of 115 kbit/s.

In principle, the static data could be stored in the microcontrollers at the end of the semiconductor fabrication process by using the previously described automated testers to transfer the data to the chip memory after completion of the tests. This could be done using other operating modes of the chip interface that allow even higher transfer and writing speeds, which in some cases can yield rates as high as 800 kbit/s. However, for logistical reasons this approach has been used only rarely up to now.

#### 14.7.4 Optimized mass data transfer to smart cards

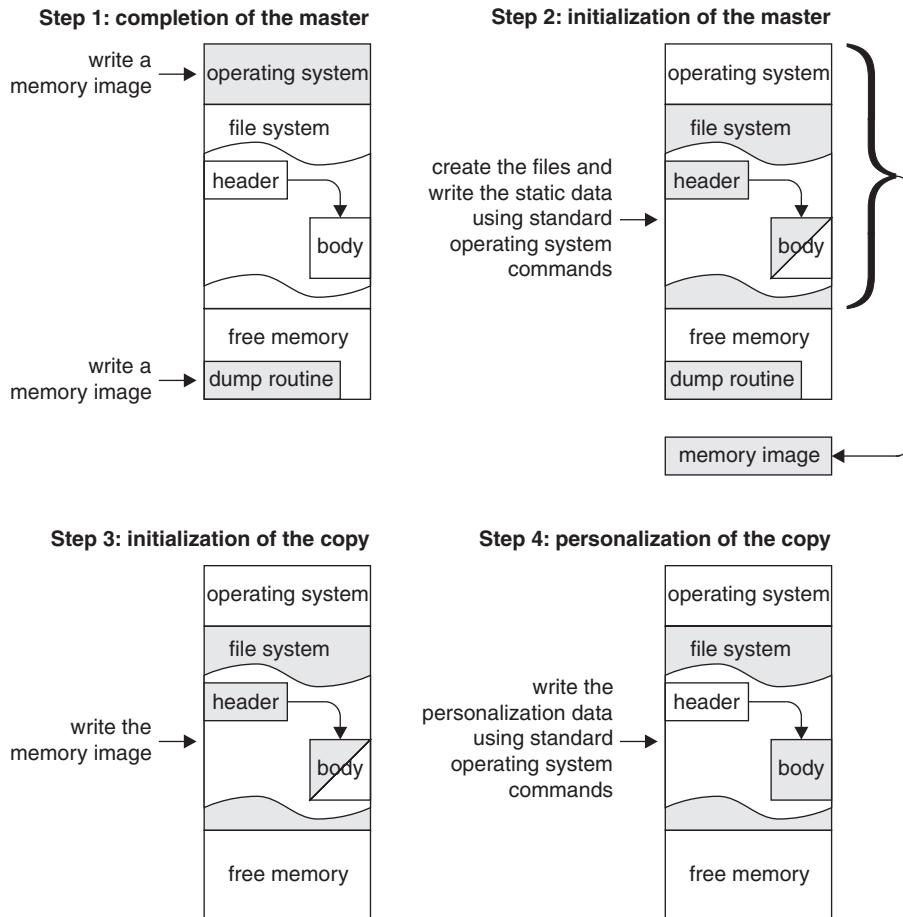
There are two fundamentally different methods for loading initialization data into the microcontroller memory. The first method is intended to avoid direct physical addressing of the



**Figure 14.69** Schematic representation of the main steps for loading common and individual data into a smart card using file management commands such as CREATE, UPDATE BINARY, and UPDATE RECORD. The shaded areas mark the data written in each step. For simplicity, this diagram only shows the process of loading a file system. A similar procedure is used to load other types of data, such as Java applets

memory, for which reason only logical addresses in the microcontroller are used (as much as possible) for initialization and personalization (see Figure 14.69). From a purely theoretical perspective, this is the preferred method because it avoids the need to use physical addresses outside the smart card. This automatically eliminates many potential sources of error, and to a certain extent it makes loading data into the smart card independent of the type of microcontroller that is used. The drawback of this method is that it significantly increases the time required for initialization and personalization, and time is an especially critical factor in mass production.

Consequently, a second method is also used in practice to load data into smart cards, in which the initialization data is written to microcontroller memory using direct external physical addressing (see Figure 14.70). This yields significant time savings compared with a method using logical addresses. Unfortunately, this method requires physical addresses to be used outside the card, with the associated drawbacks with regard to error vulnerability and



**Figure 14.70** Schematic representation of the main steps for loading common and individual data into a smart card by physically copying the data from a master version previously generated using logical commands. Further details of the processes used to generate the master version and the copy are shown in Figure 14.69 on the preceding page. The numbering scheme used here also corresponds to this figure. The shaded areas mark the data written in each step. For simplicity, this diagram only shows the process of loading a file system. A similar procedure is used to load other types of data, such as Java applets

general-purpose use. In practice, the method is usually selected on a case-by-case basis. If the number of smart cards to be produced is sufficiently large, the higher cost of the software for the initialization machines and the necessary complicated tests can be justified.

In order to write data directly to physical addresses, the data must be suitably prepared in advance. This requires generating a simulation of the entire file management system of the smart card operating system. A conversion program can then be used to load the data in suitably coded file bodies in the simulation and generate the corresponding file headers. After this, the files constructed in this manner can simply be relocated to the right addresses in memory. Naturally, the entire process must be performed without errors and in a manner corresponding to the version of the operating system that is used. Following this, the data can be read from the

simulation and written directly to physical memory addresses in the smart card, using the standard commands. Unfortunately, this approach has not proven worthwhile in practice because its cost far exceeds its benefit. In addition, the cost of testing using complicated black-box tests to ensure freedom from errors would be excessive. Consequently, this method is rarely used.

The method commonly used now is much simpler. A smart card containing a dump routine in an otherwise unused memory region is first initialized using logical commands. The initialized memory is then read out using the dump routine, and the resulting data is written to the physical addresses of the smart cards to be initialized and personalized. This allows the initialization and personalization times to be reduced significantly. In principle, this method can be regarded as a form of cloning. Its major advantage is that it is simple and robust. In addition, the data loading time is independent of the profile, which considerably simplifies utilization planning.

The only critical aspect is that the smart card containing the dump routine must never be allowed to leave the production facility. If this smart card were fully personalized, the dump routine could be used to read out all of the secret data. Consequently, this smart card has suitable mechanisms to prevent it from ever being misused to read out memory if it comes into the wrong hands, either inadvertently or as the result of an attack. One relatively simple way to achieve this is to have the dump routine automatically delete itself the next time the card is reset. This way the smart card can only be used during one session, since it loses its dump capability as soon as its power is cut off.

#### 14.7.5 Accelerating data transfer to the smart card

The technique described here for accelerating data transfer to the smart card is called pipelining or interleaving mode. It is sometimes used with microcontrollers having more than 32 KB of nonvolatile memory, in order to increase the data transfer rate.

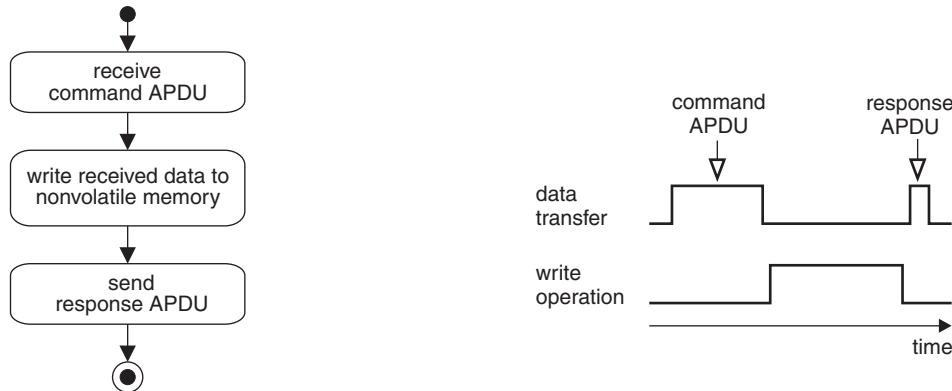
The data loading time is a significant cost factor in the production of smart cards. Reducing this time increases throughput and thus allows more cards to be produced. As the volume of data that needs to be loaded into smart cards increases with each microcontroller generation, continual optimization is necessary to avoid the need to constantly procure new production machines.

There are four major factors in the time required to load data into a smart card: the time required to transfer the data, the time required to write the data to nonvolatile memory, the computation time in the smart card, and the time required by the terminal to provide the data for the smart card.

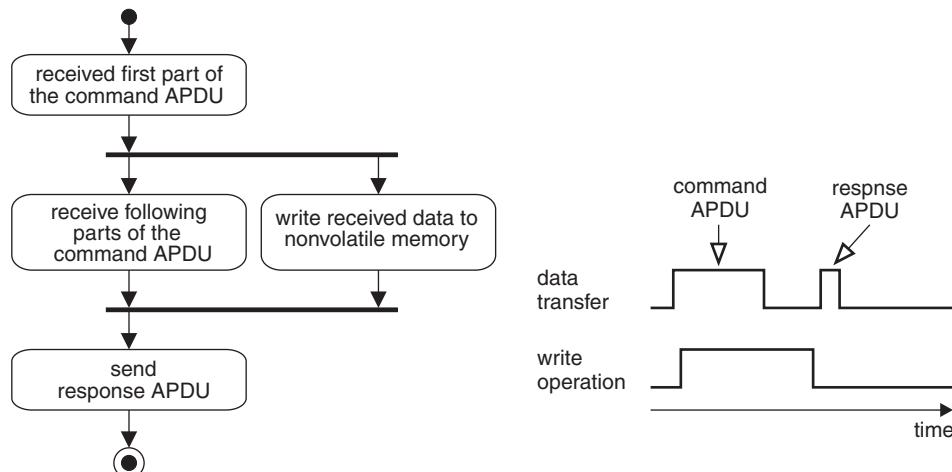
The time required to provide the data can be reduced by using a fast production machine infrastructure. The computation time in the smart card can be reduced by using a higher clock rate, and the data transmission rate can be increased dramatically by using a smaller divisor. The only factor that cannot be improved is the write time of the nonvolatile memory, since a certain minimum time is required to store data in a page in order to ensure that the data is stored reliably for the specified data retention time and over the specified temperature range.

The only possible approach is to abandon the normal sequential command processing sequence. Figure 14.71 on the facing page shows a nonoptimized command processing sequence for storing data in nonvolatile memory. The smart card first receives the entire command, then the received data is written to nonvolatile memory, and finally the response is sent to the terminal.

The processing time of this strictly sequential process can be reduced significantly by starting to write the data to the nonvolatile memory as soon as the first data is received. This



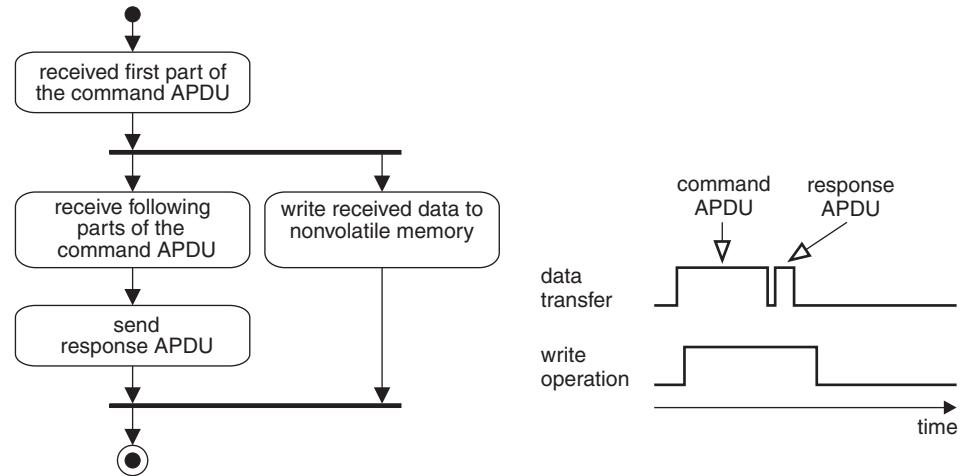
**Figure 14.71** Logical sequence and timing of nonoptimized command processing for storing data in the nonvolatile memory of a smart card



**Figure 14.72** Logical sequence and timing of optimized command processing for storing data in the nonvolatile memory of a smart card

only requires the smart card to accumulate enough data to write the first page of memory in a single operation. While this write operation is taking place, the data for the next memory page can be received from the terminal. This procedure is repeated until all the data has been received and shortly thereafter stored completely in nonvolatile memory. This is followed by sending the response to the terminal, as with a nonoptimized process. The optimized process is shown in Figure 14.72.

This approach is only possible if the smart card operating system supports this simple form of multitasking, since data reception and writing data to the nonvolatile memory occur in parallel. However, practically all modern microcontrollers support this functionality. Significant production time savings can be achieved with this optimization, with the amount of savings depending on how much data must be written with each command.



**Figure 14.73** Logical sequence and timing of command processing for storing data in the nonvolatile memory of a smart card, radically optimized for production purposes

An even more radical optimization approach is shown in Figure 14.73. Here the response is returned to the terminal immediately after the complete command has been received, regardless of whether all the data has already been written to nonvolatile memory at this point. If an error occurs during data writing, it may not be possible to report this to the terminal until the response to the next command is sent, which means that this approach violates the provisions of the relevant standards (such as ISO/IEC 7816-3). However, in certain situations this approach can accelerate production, which is reason enough for it to actually be used in production environments.

## 14.8 LOADING INDIVIDUAL DATA

The application preparation phase primarily involves the visual and electrical personalization of the smart cards. Like the prior production steps, this phase takes place in a highly automated production environment designed for processing large numbers of cards.

### 14.8.1 Generating card-specific secret data

Individual data for personalization is usually provided by the card issuer on data storage media or via data telecommunication. However, special methods are often used to provide secret data such as PIN codes and keys, since this data must remain confidential under all circumstances and it can only be generated in a highly secure environment. Four methods are used in practice with PIN data.

The simplest option is to generate a trivial PIN, which the cardholder must change to a PIN of his or her choice the first time the card is used (before it is used for the first valid transaction). However, for a variety of reasons this option cannot be implemented in all systems, even though it has the advantage of not requiring the printing and mailing of PIN letters.

A somewhat more elaborate option for generating PINs is for the card issuer to generate PIN codes using a good random number generator, followed by secure transfer of the PIN data to the card personalizer. The personalizer then writes the PIN data to the cards to be personalized using the usual secure mechanisms and generates the associated PIN letters. A variant of this option is to generate the PIN codes directly in the cards, followed by secure transfer of the PIN data to the personalization machines for further processing.

The third option is generation of random PIN codes by the card personalizer. These PIN codes, which are generated in a secure environment, are written to the appropriate data fields in the smart cards as in the previous option. In parallel with this, PIN letters are generated and sent to the cardholders. The associated smart cards reach the cardholders via a separate path. If the card issuer needs the PIN data that has been generated in this manner, it can be provided to the card issuer in a secure manner. Otherwise it is generally not necessary to store the generated PIN codes anywhere other than in the smart cards.

Another way to generate PIN codes is to use an algorithm, such as a cryptographic algorithm, to compute card-specific PIN codes using data present in the cards and a master key. The drawback of this approach is that the master key and in some cases the algorithm must be kept secret.<sup>13</sup>

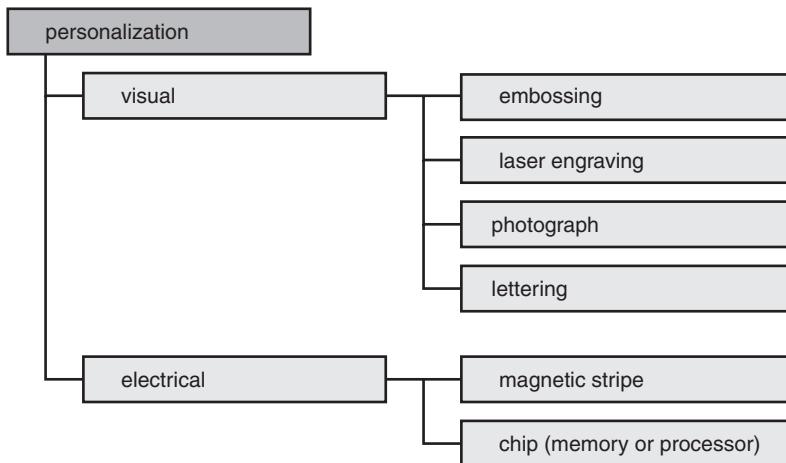
Essentially similar methods can be used if the secret data to be generated is not PIN codes but instead keys for cryptographic algorithms. The principal difference is that in this case it is not necessary to generate PIN letters, although the keys must be provided to the system operator in a secure manner. This is done using the response data, which the party that generates the keys sends to the system operator in a cryptographically secured form using data telecommunication or a physical data storage medium.

In accordance with the principle that all production steps must be checked for correct execution afterward, at least on a random-sample basis, it is essential to check whether the keys loaded in the smart cards are stored correctly and can be accessed readily by the operating system. Reading the secret keys out directly is naturally prohibited for security reasons. Testing is performed by using a special smart card command to initiate the computation of a hash value or CRC using the key concerned, which is specified externally. Although the received checksum does not allow any conclusions to be drawn regarding the actual key, it is suitable for checking whether the key concerned can be accessed properly by the smart card operating system. This only requires comparing the checksum received from the smart card with the corresponding checksum from the key management system. The advantage of this approach is that this comparison does not require any knowledge of the actual key. This technique is generally called the key check value (KCV) technique.

## 14.8.2 Personalization (individualization)

The next step in producing a smart card ready to be sent to the user is personalization, which is sometimes called individualization. In a broader sense, personalization means loading all the data associated with a particular person or a particular card into the smart card. This could be a name and address, for example, but it could also be card-specific keys. What is important is that the data is specific to a particular card.

<sup>13</sup> See Section 7.8.2, ‘Testing a secret number’, on page 188 for an example of this PIN generation method



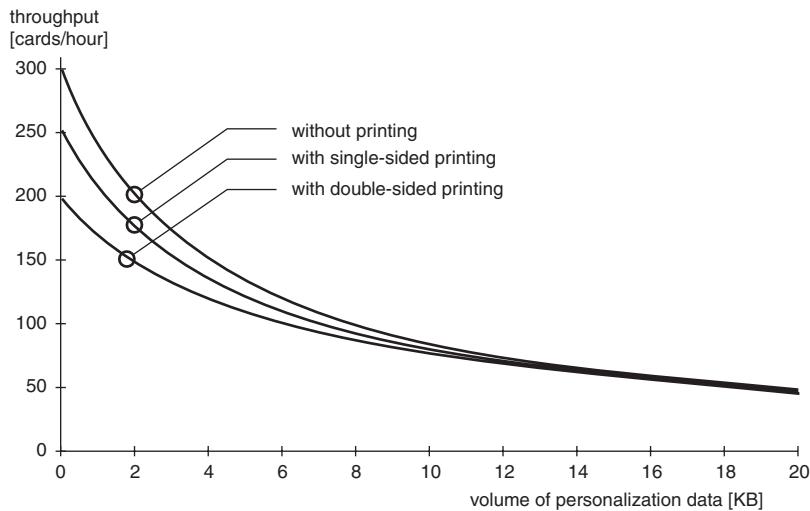
**Figure 14.74** Elements of a smart card that can be personalized



**Figure 14.75** A modular machine (Ruhlamat MS350) for visual and electrical personalization of smart cards. It can be equipped with modules for laser engraving, thermal transfer printing, inkjet printing and/or embossing, depending on specific requirements (Reproduced with permission from Ruhlamat)

A basic distinction is made between visual and electrical personalization (see Figure 14.74). The visual part of personalization consists of embossing characters on a card or laser-engraving text and imagery on the card body. The electrical part consists of loading personal data into the microcontroller and writing data to the magnetic stripe. The processing time for visual personalization depends very strongly on the specific features and cannot be generally stated. Electrical personalization usually takes 5 to 20 s, depending on the volume of data to be loaded (see Figure 14.76).

Embossing names and similar card-specific, character-based data is performed by a machine in which metal letter punches are hammered against the rear of the card at great speed and with considerable force. As this is a relatively noisy process, the embossing machines are usually



**Figure 14.76** Electrical personalization throughput with single-sided or double-sided card printing using a desktop personalization machine

separated from the rest of the production equipment. Laser engraving, which uses a laser beam to darken regions just below the overlay foil of the card body, is often used instead of embossing. This technique is also suitable for applying a black-and-white photo to the card body.

The data for the chip is written to the memory in the same way as for initialization. If this involves secret keys, cryptographically secured data transmission<sup>14</sup> is often used to prevent an attacker from deriving any advantage from tapping the data line. An even more complex method is sometimes used with cards used in payment systems. This consists of using a special security module in the personalization machine to re-encrypt the encrypted personalization data supplied by the card issuer and then loading it directly in the smart card. The advantage of this method is that the personalizer does not know the secret data in the card and cannot ferret it out by tapping the data line.

The trend in smart card personalization is increasingly oriented toward a process with full cryptographic protection. With smart cards whose card bodies do not have any security features, this allows personalization to be performed by relatively inexpensive service providers in nonsecure facilities. Other methods now in use allow the personalizer to receive the card-specific data on CD-ROM or DVD. With this approach, each production data set with its associated card-specific key is inseparably linked to the unique chip number of a particular microcontroller. Among other things, this prevents the production of duplicate smart cards by the personalizer, unless the personalizer can manipulate the operating system.

However, this method has the disadvantage that some of the supplied data sets cannot be used if some of the chips are defective, since the chips concerned are no longer available. Accordingly, with this method the personalizer must always report the numbers of the chips actually processed to the party that generates the personalization data. This is not necessary with the personalization methods presently in common use, since it is easy to produce a new

<sup>14</sup> See also Section 8.4, ‘Secure Data Transmission’, on page 225

Terminal (IFD)	Smart card (ICC)
<b>Identify Smart card</b>	
GET DATA [tag for ICCID]	————→ ————← Response [ICCID]
<b>Authenticate terminal</b>	
$KV_{IFD} = f(KMV, ICCID)$	————→
VERIFY [ $KV_{IFD}$ ]	————→ ————← IF $KV_{IFD} = KV_{ICC}$ THEN terminal authenticated Response [OK]
<b>Load data</b>	
$D_{1...n} = D(KMD, DEM_{1...n})$	————→
$KE_{ICC} = f(KME, ICCID)$	————→
$DE_{1...n} = E(KE_{ICC}, D_{1...n})$	————→
WRITE DATA [ $DE_{1...n}$ ]	————→ ————← $D_{1...n} = D(KE_{ICC}, DE_{1...n})$ store $D_{1...n}$ in nonvolatile memory Response [OK]
<b>Check stored data</b>	
COMPLETION END []	————→ ————← $EDC = H(D_{1...n})$ IF $EDC = OK$ THEN data stored correctly Response [OK]
ICCID stored in response database	————←

**Sequence Diagram 14.1** A typical process for secure loading of data into a smart card. The cryptographic functions used for this purpose and the associated secret keys are normally executed and held in a security module instead of directly in the terminal. The following data is used in this process: ICCID (unique smart card identifier), KMV (master key for generating card-specific PINs),  $KV_{IFD}$  (PIN for enabling the load function),  $KV_{ICC}$  (card-specific PIN stored in the smart card), KMD (master key for decrypting the data supplied by the card issuer), KME (master key for generating the card-specific key  $KE_{ICC}$  used for data encryption),  $DEM_{1...n}$  (data from the card issuers for loading in the smart card, encrypted using the KEM),  $D_{1...n}$  and  $DE_{1...n}$  (the plaintext data of the card issuer or the data encrypted using  $KE_{ICC}$  and the error detection code (EDC) forming the checksum of the loaded data)

card to replace a defective card. Incidentally, this is also why the personalization facilities of card producers are always secure areas.

The cryptographic algorithms and security methods used for personalization are largely confidential, so it is not possible to describe a specific application here. However, Sequence Diagram 14.1 shows an example of initialization followed by personalization as seen from a cryptographic perspective. For the cryptographic protection to be effective, these two production steps must take place in different rooms and be performed by different employees.

The basic operation of these steps is as follows. In the initialization process, a card-specific key is derived in a security module using a master key and a unique chip number. This key is sent to the smart card as plaintext, and the card stores the key. Naturally, a lot of other data must be written to the smart card during initialization, but generating and storing the card-specific key is the only cryptographically relevant operation.

Following this, the card is personalized. This can be done immediately after initialization or several weeks later. The only essential aspect is that personalization must be completely

isolated from initialization to prevent the possibility of using a card-specific key obtained illicitly during initialization to decrypt the card-specific data during personalization.

Before the start of personalization, the terminal authenticates itself by means of a secret PIN derived using a master key and a unique smart card identifier. In the next step of the personalization process, the personalization data previously encrypted using a shared key is decrypted for each card individually by the security module. This encryption/decryption process is necessary because the producer of the personalization data does not know the individual chip numbers, which are independently generated by the semiconductor manufacturer. The security module computes the card-specific key from the supplied card number using the master key.

Now the security module and the smart card have a shared secret, and they use it to encrypt the personalization data. This data is transferred in encrypted form to the smart card, where it is decrypted and written to the appropriate locations in nonvolatile memory. This method provides complete cryptographic protection of the personalization process. It secures the personalization data against spying as long as the card-specific key written to the card during initialization remains secret.

Authorization for completion or loading data can be obtained by authentication using a PIN. Alternatively, a signature method can be used here quite well. With this approach, the public key of an asymmetric cryptographic algorithm is stored in the smart card, and the data to be loaded is then signed using the corresponding private key. It is then relatively easy for the operating system to check the signature during the loading process to determine whether the data can be safely loaded. This is a quite useful application of the signature method, which is used especially for completion of the operating system.

At the end of the personalization process, the personalization machine performs several optical and electrical quality control tests on the finished smart card. For example, the visual personalization can be checked by scanning the smart cards with a camera and using a computer to analyze the acquired images and check them against the production database. If an error is detected, the card concerned is deflected to a reject bin and a new copy of the card is produced automatically. Sometimes the personalization data in the microcontroller is also checked. However, this is technically difficult because many of the files are no longer freely accessible for reading. Consequently, personalization machines have special security modules containing master keys to enable them to perform these personalization tests, so that they can check the correctness of the personalized keys in the smart cards. Indirect methods such as authentication are normally used for this purpose.

Another method that can be used is to provide the personalizer with command strings and corresponding response strings for each card. The personalizer then sends these commands in the correct sequence to the smart card and compares the responses received from the card with the proper responses to the commands. If they do not match, the smart card is not behaving as expected and a personalization error must have occurred. With this method, it is not necessary to have a special security module to perform the tests in the personalization machine.

Once a smart card has been personalized, it is generally not possible to reverse the process, which means that an incorrectly personalized smart card is worthless. Of the various processes, electrical personalization is the most prone to errors, and errors in the personalization of a large batch of cards would result in major financial losses and significant lost time. Consequently, there are a few smart card operating systems that allow all of the personalization data to be deleted after suitable authentication. In terms of the operating system, a depersonalized smart card behaves the same as after semiconductor fabrication or completion. This capability is

sometimes used with test cards because it allows the software in the smart cards to be modified without scrapping the cards. These mechanisms of the smart card operating systems are occasionally enabled for regular production cards so that depersonalization can be performed if necessary.

## 14.9 ENVELOPE STUFFING AND DISPATCHING

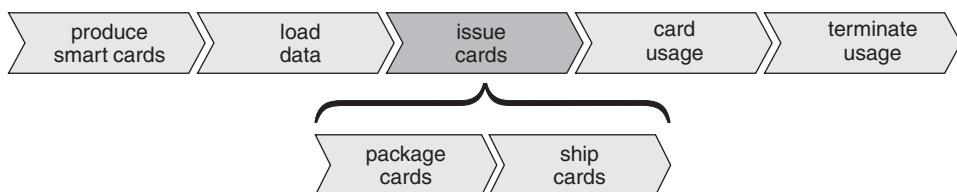
The final step in the production of smart cards consists of putting the cards in envelopes and dispatching them (see Figure 14.77). This is not necessary with some types of cards, such as prepaid phone cards, which are often delivered en masse to the card issuer. However, with more sophisticated and expensive cards the cardholder usually receives a personalized letter accompanied by his or her new card. With some applications, such as credit cards, the cardholder also receives a letter with the PIN code. For security reasons, this is sent separately and a few days later than the card. The area where these card production activities take place is often called the lettershop.

The envelopes used for PIN letters usually have a carbon-paper coating on the inside so a slip of paper in the envelope can be printed from the outside using a dot-matrix impact printer. The envelope is fashioned such that unauthorized reading of the printed PIN code is not possible without visible damage to the envelope. These measures ensure that PIN codes cannot be spied out without being noticed, even during the printing of the PIN letters. High-performance printing systems for PIN letters can print up to 34 000 documents per hour (see Figure 14.78).

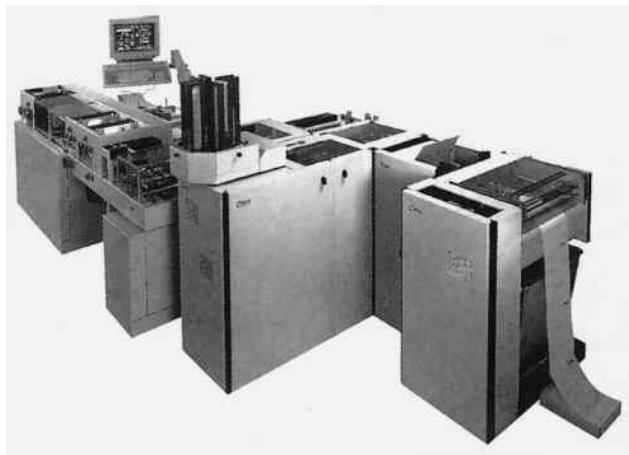
For card dispatching, the personal data such as the cardholder's name and address is either read from the card or retrieved from the production database, depending on the card type. Using a high-throughput laser printer, this information is printed on preprinted letters called card carriers. The letter may have two stamped slots to hold the corners of the card. Alternatively, a strip of easily removable adhesive tape is often used to attach the card to the letter. Following this, the card carrier is folded and placed in an envelope, possibly accompanied by other inserts. After the envelope has been franked, the smart card with the personalized letter is ready to be sent to the cardholder. High-performance envelope stuffing machines can process around 8000 letters per hour.

The final quality control step is to automatically weigh the finished letters containing the cards. The weight of the card, at around 6 g, is easily sufficient to ensure reliable verification that each envelope actually contains a card.

To minimize postage costs, the letters are usually presorted by postal code before they are handed over to the post office. This postal optimization is most easily achieved by producing the cards in the order necessary to satisfy the postal sorting criteria (such as a regional code followed by a local code).



**Figure 14.77** Card dispatching as part of the overall smart card life cycle



**Figure 14.78** A machine for attaching cards to their associated letters, which are then stuffed into envelopes along with any desired inserts. This machine can process up to 7000 envelopes per hour (Reproduced with permission from Böwe Systec)

Practical experience with even such simple matters as sending cards by post repeatedly brings new and interesting problems to light. For instance, a major producer of smart cards was once confronted with a large number of sudden failures of smart cards sent by post. After a few days of searching for the cause, it was found that the postal distribution center concerned had changed the arrangement of the feed rollers in the sorting machine. With the new arrangement, the letters containing the smart cards were bent so severely during sorting that with some cards the chips broke inside the modules. The problem was solved by shifting the position of the card on the card carrier by a few centimeters.

Due to incidents of this sort, before a mass mailing takes place a few hundred test letters are often posted in the target region and then examined in order to ensure that the smart cards will not be damaged during transport or sorting. Cards that are defective when they reach the customer are called ‘dead on arrival’ (DoA) cards.

## 14.10 SPECIAL TYPES OF PRODUCTION

As previously mentioned, smart card production is a typical mass-production process, and the production volume is probably the highest in the world for personalized products with independent computers. However, smart cards are also largely standardized products that are easily interchangeable. They generally do not have any unique identifying features. For this reason, a few card producers attempt to set themselves apart from the competition by offering special production-related services. The following topics – production on demand, single-card printing, and direct on-site card issuing – are examples of such services.

### 14.10.1 Production on demand (PoD)

Due to economic constraints, smart card production is necessarily a form of mass production. However, there is sometimes a need for selective production of single personalized cards, such as in case of lost cards or defective cards. This need is dependent on the extent to which the

card issuer wishes to selectively address individual customers. Production of single cards as needed is called production on demand.

It would be technically difficult and economically unreasonable to use regular production facilities and processes to produce single cards. In practice, a separate production line is set up for this purpose, using machinery that is optimized for this form of production. Although mass production at the lowest possible cost is not possible with this arrangement, it is suitable for the flexible production of single cards at an acceptable cost.

The card bodies used for this purpose, with the modules already implanted, are first produced in the usual quantities on production lines for mass production and then procured individually as needed. Individualization is performed on a production line optimized for single-card production. This can easily include elaborate four-color digital printing. After the card is produced, it is placed in an envelope with an accompanying letter and sent directly to the customer.

The modified process for single-card production and the special production line needed for this purpose require very close long-term cooperation between the card issuer and the card producer. This is ensured by means of suitable contracts, which also provide the card producer with the assurance of adequate return on investment. These contracts include service level agreements (SLAs) that specify matters such as the maximum response time. For example, a typical service level may be that the customer will receive the new card by 12 noon if the card producer receives the data set for the card to be produced no later than 3 pm on the previous working day.

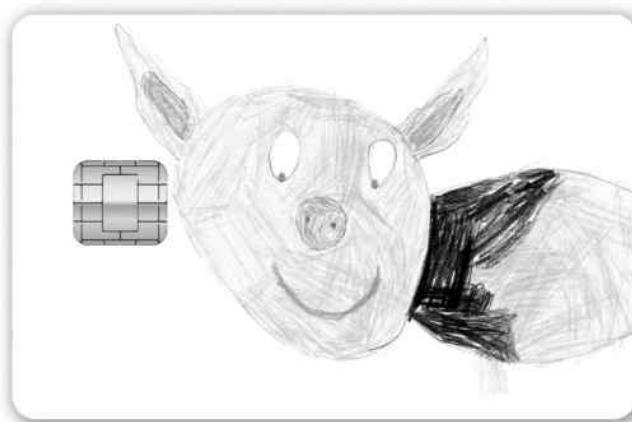
The desired short response times with production on demand are only possible with a direct link between the card management system of the card issuer and the production system of the card producer. In addition, all associated processes must be fully automated so that the card issuer can directly trigger the production sequence for a single card and receive feedback when the order is placed, with advice of the time of arrival of the card at the customer.

A typical scenario for using production on demand could take the following form. While paying for her lunch, a bank customer notices that her credit card is cracked, and she is afraid that it will shortly become unusable. She calls the service number of the bank and advises the customer support desk that she needs a new credit card. The customer support employee can access the customer database from his work station and trigger the production of a new card directly from his computer.

The card producer immediately receives the personalization data for the new card, at say 2:30 pm. Production of a new single card is initiated in response, using a card body with the right layout and a suitable microcontroller taken from a store of cards. After completion of visual and electrical personalization, the card is placed in an envelope and taken to the post office along with other cards that have been produced in the same way. The post office transports the card overnight to the distribution center for the target destination, and at 10 am the next day the letter carrier drops the letter with the customer's new credit card in her mail box. In accordance with the instructions in the accompanying letter, the customer destroys the old card by cutting it in two. At 1 pm, she can again pay for her lunch with the new credit card.

### 14.10.2 Picture cards

In line with the global trend to personalization of all sorts of everyday objects, many issuers of credit cards offer their customers cards with personalized features. Using a Web interface



**Figure 14.79** Example of a picture card produced individually for a specific customer

on the Internet, customers can select one of several photos or images or upload their own photos. The personalized credit card is then produced and sent to the customer by post. This special type of card is often called a picture card or a vanity card (see Figure 14.79). The major credit card companies, such as Visa and MasterCard, have guidelines for the specific layout of cards of this sort. It must always be ensured that cards personalized in this manner are reliably recognized as genuine credit cards when the customer uses them to make purchases. It is thus clear that all typical credit card features, such as embossing, a hologram, a magnetic stripe, and the logo of the credit card company must be present on the card body in their usual locations.

If the customer uploads his or her own photo for the card, the photo must be checked by the card producer, since it will ultimately be printed on an official credit card. This checking (screening) presently cannot be performed entirely by automated image recognition, but instead requires at least occasional manual checking. The screening criteria are broadly formulated and include sexually and politically offensive images as well as logos subject to copyright or that are undesirable on cards of this sort. If the customer uploads an image, the customer must also confirm that he or she is entitled to use the image.

After the customer's image has been suitably scaled and prepared for printing, the credit card data can be generated. Electrical personalization is then performed, and the photo is printed on a previously produced blank card using the thermal transfer method. The blank card is a credit card with all the necessary card components, and it is only necessary to print the customer's image on the card. As this is a form of just-in-time production, small desktop machines are normally used for personalized printing and electrical personalization instead of the large personalization machines used for mass production.

Nowadays single-card production is suitable not only for individualists among credit card users, but also for company credit cards in small volumes. Producing company credit cards in small volumes is usually too complicated and too expensive because the card layout must always be approved by the card issuer. However, with the picture card method it is possible to produce credit cards with company-specific printing with volumes as low as ten cards or one hundred cards.



**Figure 14.80** A desktop personalization machine (CLP 54) with integrated laser engraving and a capacity of up to 150 smart cards per hour (Mühlbauer)

#### 14.10.3 Direct smart card issuing (instant issuing)

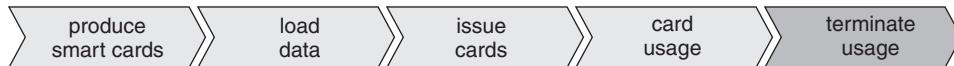
Issuing smart cards directly at the bank counter is a new development in the payment cards area. With this arrangement, the customer receives a personalized card immediately after submitting an application for a card. This concept requires the local bank branch to be able to perform complete electrical and visual personalization on site. It is essentially similar to producing picture cards using desktop personalization machines (see Figure 14.80).<sup>15</sup>

The customer service employee receives all the personal data needed for the card from the customer and enters this data in a computer. The data is then transferred to a central location where it is checked according to the type of card concerned. If it passes these tests, the associated personalization data for the card is generated in the same central location and sent to the production computer in the local branch. Naturally, suitable cryptographic measures must be used to protect all communications between the local branch and the central location against interception and manipulation.

The local branch has a desktop personalization machine that uses blank cards with all the necessary card components for the type of card to be issued, such as a hologram, magnetic stripe, and a chip. Desktop machines are now available that can perform laser engraving as well as embossing, in addition to the usual thermal transfer printing. They can be used to produce elaborate types of cards directly on site. An especially important aspect of issuing smart cards directly is secure storage and handling of the blank cards. For example, in the case of credit cards the blank cards already have all the security features, such as a hologram and microtext. Counterfeiters would find it relatively easy to use these blank cards to produce cards with virtually the same appearance as genuine credit cards.

With this arrangement, it is no longer necessary to produce cards exclusively in a central location and send them to customers by post. With this technology, they can be produced in the local branch or by a cash dispensing machine and used immediately by the customer.

<sup>15</sup> See also Section 14.10.2, 'Picture cards', on page 626



**Figure 14.81** Termination of card usage as part of the overall smart card life cycle

## 14.11 TERMINATION OF CARD USAGE

The final phase of the smart card life cycle according to the ISO 10202-1 standard defines all measures related to the termination of card usage (see Figure 14.81). Specifically, these measures consist of deactivating the applications in the smart card, followed by deactivation of the smart card itself. However, with most smart cards these two processes are purely theoretical. In practice, cards are either tossed into the waste bin or carefully labeled and filed away by collectors for an indefinite length of time. Cards are rarely returned to the card issuer.

### 14.11.1 Deactivation

Suitable commands are available for deactivating individual applications and the entire smart card. For instance, the ISO/IEC 7816-9 commands DELETE FILE, DEACTIVATE FILE, TERMINATE DF and TERMINATE CARD USAGE are explicitly intended to be used to herald the final phase of the life cycle of an application.<sup>16</sup>

These commands are especially important for managing individual applications in multiapplication cards; they are rarely used with current smart cards, which generally have only one application. The easiest way to end the life of a smart card is to cut it into pieces with a pair of scissors. Anyone can do this, and some card issuers recommend this method for ‘terminating’ smart cards.

However, in some cases it would certainly be justified to return the smart cards to their issuer for security reasons. Some of the secret keys in the cards are still valid, and if a potential attacker can acquire several hundred or even thousands of cards, he has significantly more data available for analyzing the hardware and software than if he has only a few cards. Statistical analyses based on a large number of cards will always yield more information than those based on single cards.

For this reason as well as environmental considerations, some card issuers have started collecting expired cards when new cards are issued. Collection bins for used telephone cards are sometimes placed near card phones. Collecting the cards is an essential prerequisite for effective recycling.

As an exception to the usual practice of treating smart cards as disposable objects, there are now a few practical examples of intentional deactivation of smart cards. This often results from a limited number allotment and the fact that increasing the size of the existing allotment would incur costs that the card issuer would rather avoid. A typical example is provided by mobile telephone systems in which the number of issued dialing numbers has reached the limit of the number allotted to the system operator and new numbers would have to be purchased from a regulatory authority. In such situations, the operator often appeals directly to the customers and sometimes offers small incentives to entice customers to return their surplus SIM cards

<sup>16</sup> See also Section 11.8, ‘File Management Commands’, on page 384

to the system operator. The SIM cards are collected and sent to a central location where their unique identification numbers (usually the ICCIDs) are read from the cards.

The associated dialing numbers can be determined from the ID numbers by consulting the database of the appropriate background system. The smart cards are then deactivated, and the associated dialing numbers can be issued anew in the system. With this method, it is essential to be able to read the unique identification number from the smart card, even if the PIN is not known. With this identification number, additional data contained in the smart card can be determined by consulting the database of the background system.

### 14.11.2 Recycling

With regard to the recycling of smart cards, it must be noted that little progress has been made. At present, the volume of smart cards that is collected is simply not enough for effective recycling. In 2007, approximately 15 000 metric tons of plastic were used worldwide for smart card production. Even under the fully idealistic assumption that a volume of cards with the same weight is discarded and the entire volume can be fed back into a recycling process, this volume is negligible compared with the total worldwide production of plastics, which amounted to around 230 million metric tons in 2005.

Recycling is a difficult subject with smart cards. With card bodies laminated from several layers of different types of plastic, smart cards form a highly heterogeneous material (see Table 14.5). In addition, the cards are printed with different kinds of ink and include holograms, signature panels and magnetic stripes, all of which increase the number of materials in the mix. Essentially homogeneous materials can only be collected during card production, for instance as scrap when cards are stamped from monolayer sheets. It is relatively easy to reuse these materials, and many card producers already do so.

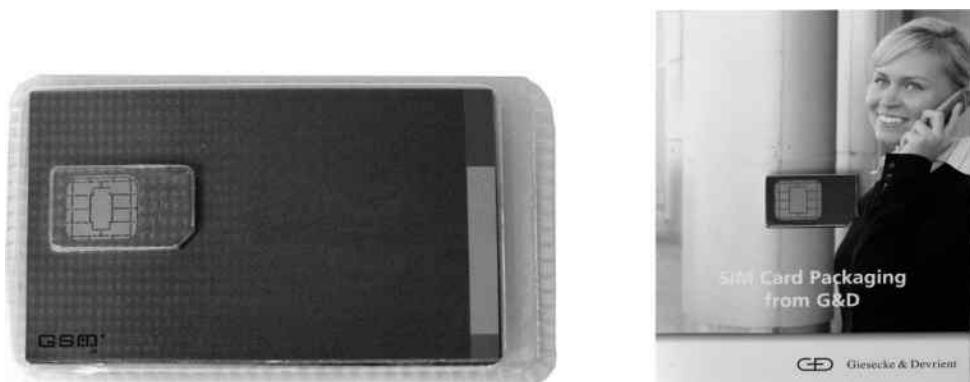
With smart cards that are no longer used, by contrast, it is practically impossible to separate the cards into homogeneous materials. The currently suggested recycling method is to punch the modules out of the cards and then shred the card bodies (see Figure 14.82). The plastic shavings can then be used to produce low-quality plastic items (garden ornaments are a

**Table 14.5** The major components of smart cards by weight

Component	Material	Weight
ID-1 card body	various plastics (e.g. PVC, PC, and ABS)	4.400 g
ID-000 card body (plug-in)	various plastics (e.g. PVC, PC, and ABS)	0.359 g
Inks on the card body	resins and pigments	very low
Magnetic stripe	iron oxide and similar materials, inks, and glue	very low
Hologram	aluminum and glue	very low
Microcontroller ( $10 \text{ mm}^2$ )	silicon with various doping elements	0.009 g
Bonding wires	gold or aluminum	very low
Microcontroller encapsulation	epoxy resin	0.010 g
Glue for bonding the module to the card body	epoxy resin	very low
Module with six contacts	epoxy resin, glass fibers, nickel, aluminum, gold	0.170 g
Module with eight contacts	epoxy resin, glass fibers, nickel, aluminum, gold	0.180 g



**Figure 14.82** Smart cards at the end of their life cycle after physical destruction by shredding



**Figure 14.83** Examples of packaging of smart cards in plug-in format. Left: fully protected in a plastic blister pack; right: a paper package with the plug-in protected by a transparent film (Reproduced with permission from Giesecke & Devrient)

typical example of this type of recycling). The modules can also be ground, and the metals they contain can be recovered using electrolytic processes. However, this method is presently not used anywhere on a large scale. In addition, it is not clear that this sort of complex recycling is truly better for the environment than simple incineration or landfill disposal.

In the case of contactless smart cards with coils of copper wire or conductive ink embedded in the card body, it is effectively impossible to separate the card materials into individual types of plastic.

Particularly with multilayer cards, the only practical approach is high-temperature incineration, which is sometimes given the grandiloquent name ‘energy recycling’. If the temperature is sufficiently high, relatively few harmful substances are produced. It remains to be seen whether this solution will be considered acceptable in the long term. After all, even though a single smart card weighs only 5 g, the net weight of one million cards is 5 metric tons, and with one billion cards the figure is a tidy 5 thousand metric tons.

Some countries now take a different approach to reducing the volume of waste. By means of legislation, operators of mobile telephone networks are encouraged to issue SIMs exclusively

in plug-in format and avoid using cards in ID-1 format with breakaway SIMs. This reduces the weight of the smart card from around 5 g to around 0.4 g, which can certainly contribute to protecting the environment. However, this is only true under the right conditions, which are not always present. Due to their small size, plug-ins are easily lost, so they are commonly packaged in transparent blister packs (see Figure 14.83). These blister packs nicely offset the weight reduction achieved by producing SIMs exclusively in plug-in format.

From an environmental perspective, the best form of packaging is a paper sleeve with the SIM attached by a removable adhesive. However, this is not an acceptable solution in countries where SIMs are sold in stalls alongside dusty roads. Blister packs are clearly the best form of packaging in such conditions. These examples clearly show that it is not possible to find a single solution that is satisfactory in all regions with regard to environmental protection considerations; instead, individual solutions are necessary for different countries and cultures.

# 15

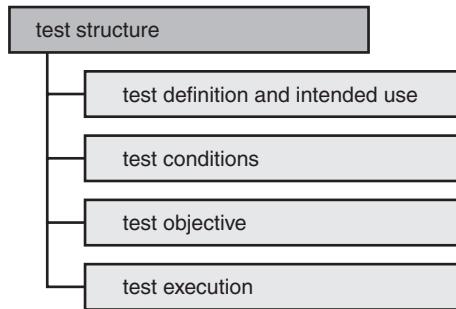
## Quality Assurance

Quality assurance and the associated test procedures and methods are especially important with smart cards. Smart card manufacturers must produce very large product volumes at high quality and low cost. In contrast to other industries, these products also contain relatively complex and sensitive microcontrollers and software that generally cannot be modified after the product has been delivered to the customer.

Compared with standard PC software, for example, the basic difference is obvious. With PC software, for many years it has been common practice to replace initial releases of new software (usually identified by a '0' at the end of the version number) with updated and improved versions at relatively short intervals ranging from a few weeks to one or two months at most. This would be impossible with smart cards. Their mask-programmed software is unalterable due to the nature of the memory that is used, and it is not feasible to replace a large number of issued cards using any sort of recall campaign. Even in case of cards that are not used in the especially sensitive payment sector, such a campaign would cause lasting damage to the reputation of the card issuer and the cost would be immense.

This is why quality assurance and testing are of fundamental importance in the production of smart cards. After the cards have been produced and distributed, it is simply not possible to update them with an improved version of the software a short time later. This naturally means that considerable effort must be expended to generate products that have as few defects as possible. With regard to the various tests, a basic distinction must be made between qualification tests and production tests. Qualification tests are used to decide whether the smart card in question can be used at all. These tests are usually performed before the market launch of a new card body, chip, module, or operating system. If the new or modified product meets the specified requirements, it is qualified for production and can be produced in large quantity. Qualification tests are performed in production only on random samples at relatively long time intervals.

Other test methods are used for production testing. These tests can generally be performed quickly and easily, in order to meet the inescapable demand of mass production for short process times and high throughput. They primarily involve only simple measurements of basic mechanical and electrical parameters, along with sending suitable test commands to the smart card microcontroller.



**Figure 15.1** Basic structure of TS 51.017 tests. This structure has been kept fairly general so it can be used for all smart card tests

Many test specifications for large smart card applications are primarily designed to ensure interoperability between smart cards and terminals. A good example of this is the TS 51.017 specification, ‘Subscriber Identity Module (SIM) Test Specification’, which occupies around 100 pages. It describes detailed tests for GSM smart cards that address aspects such as the card body, general electrical parameters (including supply voltage and current consumption), data transmission protocols, commands, and files. The TS 51.017 tests are organized as follows (see Figure 15.1):

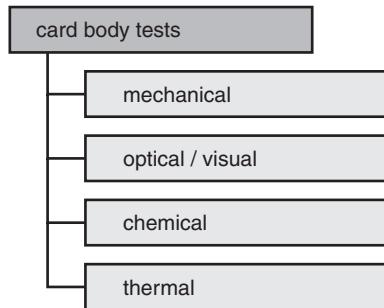
- physical properties;
- electrical signals and transmission protocols;
- logical model;
- security functions;
- functions;
- commands;
- file contents.

The organization of the individual tests in this specification is equally clear and practical. Each test consists of four parts. The first part contains a formal definition of the test and specifies its purpose. The second part lists the requirements to be satisfied, and the third part describes the test objective in detail. The final part specifies the actual test procedure.

## 15.1 CARD BODY TESTS

The ISO/IEC 10373 standard is the most important reference for testing cards with and without chips. At the European level, there is also the EN 1292 standard, but it deals exclusively with smart cards and terminals, including their general electrical requirements. Standards for cards also often include individual tests and test methods for verifying the properties specified in the standards.

Many of the usual tests and verifications for smart cards (see Figure 15.2) are described briefly below in alphabetic order. The testing laboratories of card manufacturers usually have a repertoire of 120 to 150 different tests for cards.



**Figure 15.2** A selection of commonly used card body tests. A set of specific tests is necessary for each of the individual card components (hologram, magnetic stripe, chip, etc.)

Standard ambient conditions are a fundamental requirement for the test environment, which means that a temperature of  $23 \pm 3^\circ\text{C}$  and a relative humidity of 40 to 60 % must be maintained in the test laboratory. The cards to be tested must be acclimatized to these conditions for at least 24 hours before the actual testing takes place.

### 15.1.1 Adhesion (or blocking)

Basis: ISO 7810; test specification: ISO 10373. This test checks whether the card characteristics change when it is stored under certain ambient conditions. Five unembossed cards are stacked together and subjected to a uniform pressure of 2.5 kPa at  $40^\circ\text{C}$  and 90 % relative humidity for 48 hours. After this, the cards are inspected for delamination, discoloration, surface changes, and other visible changes.

### 15.1.2 Amplitude measurement

Basis: ISO 7811-2; test specification: ISO/IEC 10373. This measurement verifies the signal amplitude and resolution of the magnetic stripe recording material. The measurement is made using a standardized read/write head that is passed along the magnetic stripe at a precisely defined speed.

### 15.1.3 Bending stiffness

Basis: ISO 7810; test specification: ISO 10373. To determine whether the card has the required bending stiffness, one end of the card is clamped to a depth of 3 mm with the card facing downward and the amount of bending is first measured with no load. A load of 0.7 N is then applied to the other end of the card, and the difference between the amount of bending under load and the amount with no load is measured. The result indicates the bending stiffness of the card.

The bending stiffness test is often also performed at temperatures below or above the usual test temperature of  $23^\circ\text{C}$ .

### **15.1.4 Card dimensional stability and warpage with temperature and humidity**

Basis: ISO 7810; test specification: ISO 10373. The shape and size of certain types of plastic change markedly in response to variations in relative humidity. Consequently, compliance with the standards must be tested under these conditions using suitable methods.

For this purpose, the card is placed flat on a surface and the temperature and relative humidity are varied. Four different test conditions are used:  $-35$ ;  $+50$ ;  $+25^{\circ}\text{C}$  at  $5\%$  RH; and  $+25^{\circ}\text{C}$  at  $95\%$  RH. The dimensions and warpage of the card are checked against the standard values after it has been exposed to each of these conditions for 60 minutes.

### **15.1.5 Card dimensions**

Basis: ISO 7810; test specification: ISO 10373. This test measures the height, width, and thickness of an unembossed card. A force of  $2.2\text{ N}$  is applied to the card and its height and width are measured using a profile projector. For measuring the thickness, the card is divided into four equal rectangles and the thickness is measured at the center of each rectangle using a micrometer with an applied force of  $3.5$  to  $5.9\text{ N}$ . The measured maximum and minimum values are compared with the specified standard thickness.

### **15.1.6 Card warpage**

Basis: ISO 7810; test specification: ISO 10373. This test measures the warpage of the card. The card is placed on a flat surface and the warpage is measured using a profile projector. This test is primarily intended for cards that are stamped from a base material supplied in roll form.

### **15.1.7 Delamination**

Basis: ISO 7810; test specification: ISO 10373. This test is only meaningful for multilayer cards, which are composed of several laminated plastic foils. The cover foil is separated from the core foil at one point using a sharp knife. Starting with this separation, the tester attempts to pull the two laminated foils apart. The necessary force is measured and compared with reference values.

### **15.1.8 Dynamic bending stress**

Basis: ISO 7816-1; test specification: ISO/IEC 10373. The dynamic bending test is illustrated in Figure 15.3 on the facing page, and a machine for performing bending tests is shown in Figure 15.4. The card is flexed 30 times per minute ( $0.5\text{ Hz}$ ) with a deflection  $f$  of  $2\text{ cm}$  across its length or  $1\text{ cm}$  across its width. The card must withstand at least 250 flexures in each of the four possible directions (a total of 1000 bending cycles) without damage.

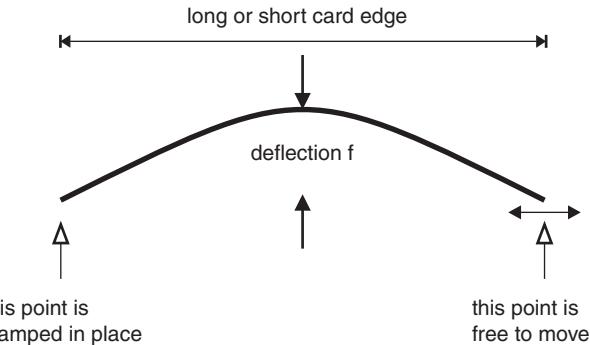


Figure 15.3 Schematic diagram of card loading in the dynamic bending test



Figure 15.4 A machine for performing dynamic bending tests on smart cards

### 15.1.9 Dynamic torsional stress

Basis: ISO 7816-1; test specification: ISO/IEC 10373. In the dynamic torsion test, the card is twisted  $\pm 15^\circ$  about its longitudinal axis at 30 twists per minute (0.5 Hz). The standard requires the card to withstand 1000 torsion cycles without functional failure of the chip or visible mechanical damage to the card.

### 15.1.10 Electrical resistance and impedance of contacts

Basis: ISO 7816-1/2; test specification: ISO/IEC 10373. The electrical resistance of the contacts is an important criterion for reliable provision of electrical power to the microcontroller in the card and data transmission to and from the microcontroller. The resistance is measured using two test probes, which are applied with a force of  $0.5 \pm 0.1$  N to two opposite corners of the smallest allowable contact rectangle. The resistance between the two test probe contacts, which are gold-plated and rounded with a radius of 0.4 mm, must be less than  $0.5 \Omega$ .

### **15.1.11 Electromagnetic fields**

Basis: ISO 7816-1; test specification: ISO/IEC 10373. In this test, the card is moved into a static electromagnetic field with a strength of 1000 Oe (79.6 H) at a maximum velocity of 1 cm/s. The memory contents of the card must not change as a result.

### **15.1.12 Embossing relief height of character**

Basis: ISO 7811-1; test specification: ISO/IEC 10373. In this test, the thickness of the embossed portion of the card is measured with a micrometer, using an applied force of 3.5 to 5.9 N.

### **15.1.13 Flammability**

Basis: ISO 7813; test specification: ISO/IEC 10373. The flammability of the card is measured by holding one edge at an angle of 45° in a specified Bunsen burner flame (diameter 8.5 mm, height 25 mm) for 30 s.

### **15.1.14 Flux transition spacing variation**

Basis: ISO 7811-2; test specification: ISO/IEC 10373. This test determines whether the magnetic flux transitions that encode individual bits in the magnetic stripe are uniform and sufficiently strong. For this purpose, a read head is passed along the stripe and the flux transitions are recorded. The measured results are then compared with the values specified in ISO 7811-2.

### **15.1.15 Height and surface profile of the magnetic stripe**

Basis: ISO 7811-2/4/5; test specification: ISO/IEC 10373. This test measures the height and smoothness of the surface of the magnetic stripe. It is performed using a special measuring device described in detail in the standard, which can record a height profile.

### **15.1.16 Light transmittance**

Basis: ISO 7810; test specification: ISO 10373. Some cards have an optical barcode on an embedded foil. This test is suitable for determining the optical transparency of the cover layer and the rest of the card body. One side of the card is illuminated with a light source, and amount of transmitted light is measured on the other side using a detector that is sensitive to light with a wavelength of 900 nm.

### 15.1.17 Location of contacts

Basis: ISO 7816-2; test specification: ISO/IEC 10373. This test is used to measure the contact locations. The card is placed on a flat surface and subjected to a force of  $2.2 \pm 0.2$  N. The positions of the contacts relative to the edges of the card are then measured using any desired method with a maximum error of 0.05 mm.

### 15.1.18 Resistance to chemicals

Basis: ISO 7810, ISO 7811-2; test specification: ISO/IEC 10373. The chemical resistance of the card body and the magnetic stripe are determined using these tests. Different cards are placed in the following precisely specified liquids at a temperature between 20 and 25 °C:

- 5 % aqueous solution of sodium chloride;
- 5 % aqueous solution of acetic acid;
- 5 % aqueous solution of sodium carbonate;
- 60 % aqueous solution of ethyl alcohol;
- 10 % aqueous solution of sugar;
- petrol (according to ISO 1817);
- 50 % aqueous solution of ethylene glycol.

Each card is left in the corresponding solution for 1 minute and then removed for visual examination and testing by reading the magnetic stripe data with a magnetic card reader.

### 15.1.19 Static electricity

Basis: ISO 7816-1; test specification: ISO/IEC 10373. This test, which is only meaningful for cards with embedded chips, checks the chip's resistance to damage from electrostatic discharge (ESD resistance). A 100-pF capacitor alternately charged to +1500 V and –1500 V is discharged through a  $1500\text{-}\Omega$  current-limiting resistor into the various contacts of the chip. There must be no damage to the functionality of the chip and no change to the contents of its memory due to the discharges.

### 15.1.20 Surface profile of contacts

Basis: ISO 7816-1 and ISO 7816-2; test specification: ISO/IEC 10373. This test compares the surface profile of the individual contacts with the surface of the rest of the card. It is intended to ensure that the contacts are in nearly the same plane as the surface of the card body.

### **15.1.21 Surface roughness of the magnetic stripe**

Basis: ISO 7811-2; test specification: ISO/IEC 10373. The surface roughness of the magnetic stripe is measured using the same device as for the height and surface profile. However, this is done using a special probe tip for measuring the surface roughness of the magnetic stripe. This test is important because surface roughness is one of the major wear factors for read/write heads in magnetic-stripe readers.

### **15.1.22 Ultraviolet light**

Basis: ISO 7816-1; test specification: ISO/IEC 10373. EEPROM and flash memories lose their contents when they are exposed to ultraviolet light, so a specific test is needed to ensure that the smart card is not sensitive to ultraviolet light. The card is exposed for 10–30 minutes to ultraviolet light with a wavelength of 254 nm and an energy density of 15 Ws/cm<sup>2</sup>. The data in the EEPROM or flash memory must remain unchanged.

### **15.1.23 Vibration**

Basis and test specification: ISO/IEC 10373. Cards are often exposed to strong vibration during transport and use (e.g. mobile phones in cars), so a suitable test specification is necessary. The card must be tested on a vibration table in all three axes with an amplitude not exceeding 1.5 mm over a frequency range of 10 to 500 Hz. The functionality and memory content of the chip must not be adversely affected as a result.

### **15.1.24 Wear test for magnetic stripe**

Basis: ISO 7811-2, test specification: ISO 10373. In order to determine the wear characteristics of the magnetic stripe, test data is first written to the stripe. A dummy read/write head with a hardness of 110–130 HV and a radius of curvature of 10 mm is then passed back and forth along the stripe 1000 times with an applied force of 1.5 N. Following this, the data is read back. The signal amplitude must lie within the limits specified in ISO 7811-2.

### **15.1.25 X-ray test**

Basis: ISO 7816-1; test specification: ISO/IEC 10373. The contents of EEPROM and flash memory cells can be modified by X-radiation as well as ultraviolet light. To test the X-ray resistance of the memory, the chip is exposed to X-rays with an energy of 70 kV. The memory contents are then examined for any changes and the memory is tested to see whether it can still be written.

There are naturally many other things that can be tested, such as the number of insertion cycles, the wear resistance of the inks, the stability of the plasticizer, and resistance to perspiration and saliva. Depending on where and how the card will be used, suitable tests must be selected and performed.

## 15.2 MICROCONTROLLER HARDWARE TESTS

Aside from ensuring the quality of the card body, one of the primary aims of quality assurance is to ensure that the microcontroller is in good working order. The microcontroller is the most important and most vulnerable component of a modern smart card.

The CPU and memory are subjected to a variety of tests, starting with the semiconductor fabrication stage. To support these tests, microcontrollers have a test ROM containing various routines that enable external access to the CPU and memory. In addition, there are sometimes special contact pads that provide direct access to the main buses of the microcontroller. During semiconductor fabrication, needle probes are used to contact the appropriate pads on the chip so the necessary test programs can be run. These test pads are cut off when the chips are sawn from the wafer, in order to prevent them from later being used for attacks. This means that it is no longer possible to access the internal buses of the chips.

After the dice have been packaged in the modules, they are also tested via the module contacts. This often only amounts to performing an activation sequence and seeing whether an ATR can be received. If this is possible, it is assumed that the chip has not suffered any serious damage while being packaged into the module and that all bonding wires are correctly connected. A similar ATR test is performed immediately after the module is embedded in the card body. This test checks whether the module was damaged by being briefly heated during the embedding process.

The microcontroller is tested meticulously before the smart card is initialized. Test commands specifically approved for this processing stage are used.<sup>1</sup> After successful completion of the test program, which lasts 10 to 100 s, these commands are permanently blocked against further use. These time-consuming tests can be performed at this stage without reducing throughput by using a large number of initialization machines in parallel so that the test duration does not have a significant effect. The tests performed at this stage check aspects such as whether all EEPROM or flash memory locations can be written and erased and whether the RAM is free of defects. If the chip is scratched during bonding, this could prevent some EEPROM or flash memory cells from being properly written, or cause certain regions of the ROM to have incorrect contents.

Various final tests are performed after the card has been initialized and personalized, depending on the manufacturer. This is usually done using fully automatic, self-calibrating testers that can configure their own parameters by reading all the data necessary for the tests from the smart card and performing the tests accordingly.

In addition to the relatively simple and quickly executable tests performed on all cards, there are random-sample tests that are only performed on isolated cards. These cards, which are taken from regular production, can also be subjected to destructive testing if necessary.

In addition to the ISO/IEC 10373 standard, qualification testing and continuous random sample testing are addressed by the EN 10373 standard, which defines a large variety of test methods for microcontrollers. Typical random-sample and qualification tests for microcontrollers are:

- signal rise and fall times at the I/O contact (EN 1292);
- number of possible write/erase cycles in EEPROM/flash memory;

<sup>1</sup> See also Section 11.11, ‘Commands for Hardware Testing’, on page 395

- EEPROM/flash memory data retention;
- clock overfrequency and underfrequency detection;
- Vcc overvoltage and undervoltage detection;
- I/O contact voltage (EN 1292);
- CLK contact current consumption (EN 1292);
- Reset contact current consumption (EN 1292);
- Vcc contact current consumption (EN 1292);
- Vpp contact current consumption (EN 1292).

Naturally, every card manufacturer also uses its own supplementary tests to cover special features of the embedded microprocessors. For example, there are specific tests for the various sensors on the chip.

### 15.3 TEST METHODS FOR CONTACTLESS SMART CARDS

The ISO/IEC 10373 standard contains a compilation of methods for testing ID-1 cards with and without embedded chips. They address the properties and parameters defined in ISO/IEC 7810 for ID cards in general; in ISO/IEC 7811 for magnetic-stripe cards; in ISO/IEC 7816 for contact smart cards; in ISO/IEC 11693, 11694, and 11695 for optical memory cards; and in ISO/IEC 10536, 14443, and 15693 for contactless cards. ISO/IEC 10373 consists of six parts:

- ISO/IEC 10373 Identification cards – Test methods;
- Part 1: General characteristics tests;
- Part 2: Cards with magnetic stripes;
- Part 3: Integrated circuit(s) cards with contacts and related devices;
- Part 5: Optical memory cards;
- Part 6: Proximity cards;
- Part 7: Vicinity cards.

The parts that are especially relevant to contact cards are Parts 6 and 7, which contain specific test methods for contactless cards, and Part 1, which contains test methods for the general properties of ID cards, including:

- card dimensions and warpage;
- bending resistance;
- chemical resistance;
- temperature and humidity resistance;
- UV resistance;
- X-ray resistance.

**Table 15.1** Summary of the current status of standards for proximity cards and vicinity cards. Here ‘Amd.’ stands for ‘Amendment’

Standard	Published	Current status
Proximity cards (PICCs)		
ISO/IEC 14443-1	2000	under revision
ISO/IEC 14443-2	2001	under revision
ISO/IEC 14443-2 Amd. 1	2005	
ISO/IEC 14443-3	2001	under revision
ISO/IEC 14443-3 Amd. 1	2005	
ISO/IEC 14443-3 Amd. 3	2006	
ISO/IEC 14443-4	2001	under revision
ISO/IEC 14443-4 Amd. 1	2006	
Vicinity cards (VICCs)		
ISO/IEC 15693-1	2000	under revision
ISO/IEC 15693-2	2006	second edition
ISO/IEC 15693-3	2001	under revision
Test methods for proximity cards		
ISO/IEC 10373-6	2001	under revision
ISO/IEC 10373-6 Amd. 1	2007	
ISO/IEC 10373-6 Amd. 2	2003	
ISO/IEC 10373-6 Amd. 3	2006	
ISO/IEC 10373-6 Amd. 4	2006	
ISO/IEC 10373-6 Amd. 5	2007	
Test methods for vicinity cards		
ISO/IEC 15693-7	2001	under revision

ISO/IEC 10373 describes test methods for ensuring compliance with the standard, but it does not specify any limits. They must be taken from the previously mentioned standards (see Table 15.1).

Practical experience in the past has shown that, especially with contactless cards, the test methods are very important for achieving interoperability. In particular, standard-compliant implementation of the coupling field for power and data transfer is nearly impossible to demonstrate without a precisely specified measuring method. In many cases, ambiguous aspects of the standard can only be resolved by means of clearly defined test methods. Particularly with the introduction of electronic travel documents (passports, visas and identification documents) in recent years, numerous problems with the interoperability of cards and terminals arose in practice. Extensive tests carried out by ISO/IEC and the ICAO have raised many questions that must be answered by working groups. In order to deal with this, the ISO/IEC 14443 standard and the relevant part of the ISO/IEC 10373 standard (Part 6) have been revised and corrected several times. As a result, there are many amendments to the standards, which unfortunately makes it more difficult for users to maintain an overview. The current status (spring 2008) and year of issue of these standards and related amendments are listed in Table 15.1.

As can be seen, technological progress with contactless proximity cards has resulted in many amendments to the standards. Additional amendments are currently being drafted in the

standardization committees. The revisions to the individual parts of the standards are provisionally expected to be published in 2008. At this time, most of the amendments will be integrated into the individual parts, which will reduce the structural clutter. We strongly recommend that readers consult the website of the ISO/IEC standardization committee ([www.wg8.com](http://www.wg8.com)) to determine the current state of standardization in order to ensure that they are using current versions of the standards. The contents of Parts 6 and 7 of the ISO/IEC 10373 standard are described briefly below.

### 15.3.1 Test methods for proximity smart cards

Part 6 of the ISO/IEC 1037 standard describes methods for testing the physical interfaces of proximity coupling smart cards compliant with ISO/IEC 14443. The test equipment necessary for this, consisting of a calibration coil, a test jig for measuring the load modulation, and a reference card (reference PICC), is defined in the standard.

Test methods for the following properties of the card or terminal are described in the standard:

- the resistance of the card to damage by electrostatic discharge;
- the amplitude of the load modulation and the functionality of the card within its defined working range, as described in the basic standard;
- the strength of the field generated by the terminal;
- power transfer from the PCD to the PICC;
- the modulation index and transient characteristics of the signal generated by the terminal;
- load modulation detection by the PICC.

It should be noted that exact, reproducible measurement of the load modulation signal is difficult due to the low signal level. Instruments and test equipment available from specialist companies can help here.

Amendment 1 of Part 6, ‘Protocol test methods for proximity cards’, defines methods for testing the initialization, anticollision, and data transmission characteristics of proximity cards. Correct implementation of the protocol specified in Parts 3 and 4 of the ISO/IEC 14443 standard is described by more than 60 scenarios. This amendment is a valuable aid for achieving correct implementation of the protocol.

Amendment 2, ‘Improved RF test methods’, essentially utilizes accumulated practical experience to improve the test methods.

Amendment 3, ‘Protocol test methods for proximity coupling devices’, describes test methods for verifying the standard-compliant performance of a terminal (PCD) in accordance with the Parts 3 and 4 of the ISO/IEC 14443 standard. The combination of this and the protocol tests for the card described in Amendment 1 provides a complete description of the test methods for protocol tests in both directions.

Amendment 4, ‘Additional test methods for PCD RF interface and PICC alternating field exposure’, describes test methods for a Class 1 PICC. A Class 1 PICC is a proximity card whose antenna is located inside a zone defined by two concentric rectangles. The outer rectangle has dimensions of 81 × 49 mm, while the inner rectangle has dimensions of 64 × 34 mm.

The standard allows considerable freedom in the locations and dimensions of antennas in proximity cards, which in practice results in large differences in the working ranges of different proximity card implementations with the same terminal. This can lead to perceptible problems in some applications. Restricting the cards to Class 1 vicinity cards creates the conditions necessary for obtaining similar results from cards made by different manufacturers when they are used with the same terminal. Restriction to Class 1 antennas can also help to achieve similar performance from products made by different manufacturers in case of ID devices with form factors other than ID1, such as electronic passports.

Amendment 5 describes test methods for proximity cards with elevated bit rates (fc/64, fc/32, and fc/16). This amendment was made necessary by the fact that elevated bit rates were only included retrospectively in the ISO/IEC 14443 standard.

It should also be mentioned that an International Technical Report with the title ‘Proximity cards – Requirements for the enhancement of interoperability’ (ISO/IEC TR 29123) was published in late 2007. As the name suggests, it describes supplementary test methods (in addition to those described in ISO/IEC 10373-6) for improving the interoperability of proximity cards and terminals.

### 15.3.2 Test methods for vicinity coupling smart cards

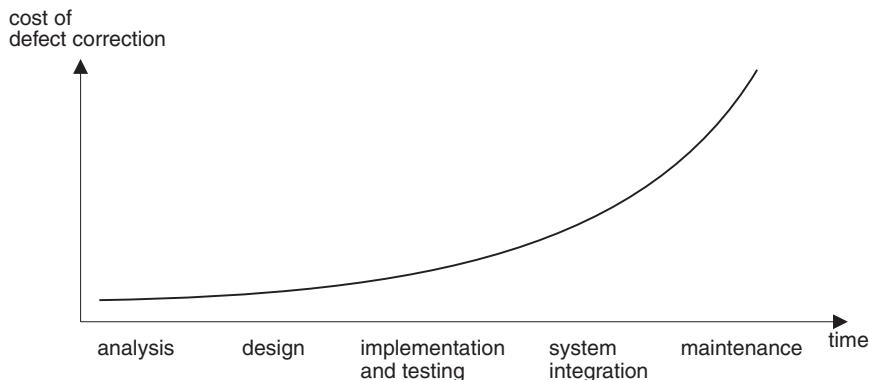
Part 7 of the ISO/IEC 10373 standard describes methods for testing the physical interfaces of vicinity-coupling cards and corresponding terminals compliant with ISO/IEC 15693. The test equipment and test methods largely correspond to those described in Part 6 of the standard. The only difference is in the structure of the reference card, due to the different subcarrier frequency. Test methods for initialization, anticollision, and the transmission protocol are currently not defined in the standard.

## 15.4 TEST METHODS FOR SOFTWARE

Physical components such as the bodies and modules of smart cards can generally be tested using conventional methods. Electrical characteristics can also be measured in a satisfactory manner using automated test equipment. However, the microcontroller software presents a somewhat different situation. Although the techniques for testing software for defects have been steadily refined since the early days of computer programming and there are many accepted techniques for producing software with low defect density, software defects are still encountered relatively often in everyday practice.

This is not a serious problem in most situations, since a revised version of the software can quickly be issued to correct defects. This cannot be done so easily with smart cards because the software cannot be modified after the cards have been issued to the users. Due to this severe constraint, software for smart card microcontrollers must have extremely low defect density. Software that is truly defect-free would be even better, but this is not feasible with the current state of software development.

As is well known, the subject of software testing is very broad, and it is described in all its forms and varieties in many books. Here we can only present a brief description of this subject, which has now become an independent field of information science. As representative examples of the extensive literature on this subject, we can mention books by Glenford J.



**Figure 15.5** The cost of correcting defects as a function of the time when they are discovered

Myers [Myers 95] and Peter Liggesmeyer [Liggesmeyer 02], both of which provide a good introduction to the subject.

Although it may sound obvious in some situations, we must say in advance that software testing does not mean combing through the software in search of defects, but instead performing tests defined by a test specification with the objective of discovering defects in the software under test. The test specification is based on the requirements of the item under test. Testing is entirely different from debugging, since tests are performed by testers while debugging is performed by software developers.

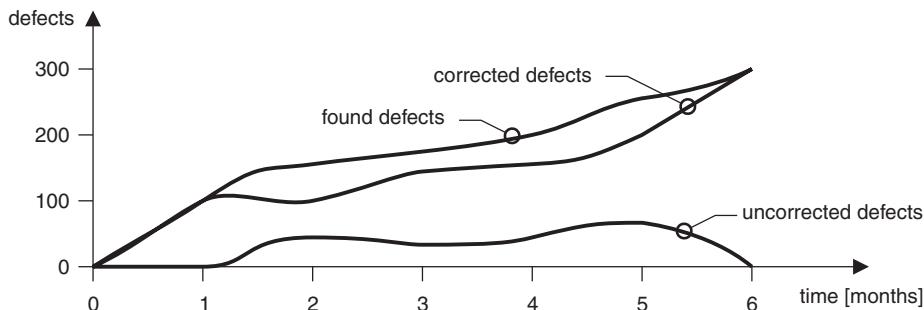
As we have repeatedly emphasized in other places, programs for smart card microcontrollers are relatively small compared with other types of software. However, they have their own special features. This can be illustrated by considering some of the general specifications of a typical operating system.

In case of a microcontroller with 16 KB of ROM and 8 KB of EEPROM, the operating system software needs around 20 KB of memory, which leaves 4 KB of EEPROM for application software. If the operating system software is programmed in assembly language, it will amount to approximately 30 000 lines of source code, which would fill 500 pages of paper printed at 60 lines to the page. The number of conditional branches will be around 2000, and even an experienced programmer will need at least nine months to generate the 20 KB of assembly language code. High-performance smart card operating systems currently have up to 300 000 lines of source code in C, corresponding to approximately 5000 pages on paper.<sup>2</sup>

This numerical example clearly shows that the software of a smart card operating system is rather complex. On top of this, the software is used almost exclusively in areas where security is a significant factor. This means that the requirement for low defect density cannot be fulfilled by simply performing a few improvised tests during or after software development. Instead, a suitable test strategy is necessary.

The programming language most often used for smart cards is C, which is relatively close to the hardware level; Java is also used for certain parts of the operating system in Java cards. Using a high-level programming language instead of assembly language is necessary in case of large smart card operating systems with more than 30 KB of code, not only due to considerations of implementation time but also due to the need for defect minimization.

<sup>2</sup> See also Section 13.1, ‘Evolution of Smart Card Operating Systems’, on page 442



**Figure 15.6** Typical curves of found, unresolved, and resolved defects versus time in a typical development project for a large smart card operating system

This can be illustrated as follows. It can be assumed that the number of defects per line of source code is nearly the same with almost all programming languages. As the functionality of a high-level language is significantly greater than that of assembly language, the resulting defect density is lower with the same amount of executable code.

For example, if we assume an entirely realistic value of 1.5 defects per 100 lines of code and further assume that only half of the defects can be found with an acceptable amount of effort, the tested program will still have 0.75 defects for every 100 lines of code (see Figure 15.6). With Java, the ratio of lines of source code to machine code (bytecode) is approximately 1:6. This means that one line of Java code has roughly the same functionality as six lines of machine code.

### 15.4.1 Fundamentals of smart card software testing

It is impossible to define a test strategy without first considering the life cycle of the smart card software. The waterfall model described by W. W. Royce, which has been known since 1970 and published in many forms, can be used for this purpose. It is relatively well suited to mask-programmed smart card operating systems. As it is also intended to be used with very large software projects for PCs and mainframe computers, here we use a simplified version specifically adapted to smart cards.

The five steps described here are normally performed in sequence. It is entirely possible for a problem encountered in a certain step to make it necessary to go back and repeat one or more previous steps. However, this should be avoided as much as possible because each iteration costs time and money.

In order to meet economic requirements such as time-to-market during software development, it is often necessary to overlap the steps to a certain extent instead of performing them in strict sequence. With this approach, which is called simultaneous engineering, portions of the software are split into individual modules as early as possible and the modules are developed in parallel using the waterfall model. As a result, it may happen that a smart card with only data transmission protocol capability is already in the system integration phase while the cryptographic algorithm for the same application is still being specified.

### ***15.4.1.1 Analysis***

The analysis phase includes specifying a basic definition of the objective and compiling the requirements into a requirements specification document, which defines all the requirements that must be fulfilled by the smart card software to be developed. The analysis phase also allows initial solution approaches to be described in the form of preliminary designs. Put simply, this phase defines what the finished software must do.

### ***15.4.1.2 Design***

The analysis is followed by the design phase, which specifies how the software will be fashioned. For this purpose, it is necessary to generate a precise specification that is not subject to interpretation and fully defines one of the various possible solutions to the requirements specified in the analysis phase. Formally structured specifications are best because they allow the features, functions and processes of the smart card software to be defined clearly and unequivocally. Specifications written in pseudocode, which can be checked for consistency and freedom from defects by computer programs, are well suited to this purpose. Unified Modeling Language (UML) is also a good choice for this phase of software development in case of graphically oriented specifications and representations.

### ***15.4.1.3 Implementation and test***

After the specification has been finalized and accepted, the program flowcharts can be generated. This is followed by programming and associated testing. The result of this phase is a fully programmed and tested smart card operating system.

### ***15.4.1.4 System integration***

As smart cards can only operate as part of a larger system, an integration phase is necessary to combine the various components of the system. The results are a fully functional and defect-free system and the final documentation of the overall system.

### ***15.4.1.5 Maintenance***

This final phase of software development can only be used to adjust the values of any general parameters incorporated in the software of issued cards. Large-scale software upgrades or changes are no longer possible in this phase.

It is a well-known fact of long standing that in all types of projects, and especially in software development projects, the cost of correcting defects increases as the project progresses (see Figure 15.5). Consequently, an appropriate amount of time and effort should be invested in the initial phases of the project as represented by the waterfall model. If the analysis is incomplete or the specification is faulty, the cost of remedying the situation increases exponentially during the course of the project.

## 15.4.2 Testing techniques and test strategies

It is nearly impossible to keep track of the wide variety of methods and techniques that have been developed for software testing. However, only a few proven techniques are necessary for testing smart card programs. Here it is possible to draw on decades of experience and a large number of publications on the subject of software testing.

Testing techniques can be classified as static or dynamic. With a static technique, the program code is analyzed and evaluated using various methods, either manually or automatically. The two most commonly used static testing techniques are statistical program evaluation using software tools and manual review of the source code. In contrast to static techniques, with a dynamic program analysis the program is tested while it is operating, either manually or with the aid of a computer. There are two basic forms of tests: blackbox and whitebox; as well as a mixed form called greybox tests.

### 15.4.2.1 Statistical program evaluation

Statistical program evaluation consists of analyzing the properties of program code, such as:

- number of lines of code (LOC);
- number of lines of comments;
- ratio of the volume of comments to the volume of program code;
- structure of the program code;
- number of functions;
- nesting depth;
- dead code;
- various complexity metrics.

### 15.4.2.2 Review

A review is a formal analysis and evaluation of a program module by a team of evaluators. In certain cases, this is called a code walkthrough or a code inspection. The basic objective is to check whether the coding, structure and data flow comply with the specification.

### 15.4.2.3 Blackbox test

A blackbox test assumes that the entity performing the test knows nothing about the internal processes, functions, or mechanisms of the program under test. Testing is thus limited to examining the input and output data, processes, and sequence of events and comparing them to the relationships defined in the specification.

Blackbox tests are the standard type of tests for smart card operating systems. They are also used to test security modules used in terminals and computer systems. It is often incorrectly

assumed that these tests can discover Trojan horses or similar items that may be present in the software, in addition to software defects. This assumption is used as a justification for omitting relatively time-consuming and expensive program code review. The main advantage of blackbox tests is that they are independent of the software implementation. There is thus no need to adapt them to the specific functionality of the program code.

Although a blackbox may be able to detect simple trap doors in smart card operating systems if they are inadvertent or not cleverly programmed, an experienced programmer can easily create access methods that can never be detected by a blackbox test. This can be illustrated by the following simple example. It is not meant to serve as a model for a Trojan horse, since it has anyhow been known for a long time, but instead to emphasize the need for code inspections as part of security analyses.

Almost all smart card operating systems include a command (GET CHALLENGE) for generating and issuing random numbers. This command could be modified such that only the first eight-byte number that it issues is actually generated by the pseudorandom number generator. Each of the subsequent supposedly random numbers could then be an eight-byte value read from the EEPROM and XORed with the initial random number. A simple external program could then be used to read out the entire memory contents, including all of the keys. Incidentally, this is a very good example of steganography applied to smart cards.

There is no way that a blackbox test can determine whether this command conceals a Trojan horse. Even a statistical analysis of the received random numbers would not detect any significant deviation from normal pseudorandom numbers. The only way to recognize such a manipulation is to inspect the entire program code of the operating system. This example illustrates only one of many ways to modify a normal smart card command so that it can be used to obtain the contents of the memory. Only a few lines of program code are needed for this purpose, so the only effective way to prevent it is to disclose and review all of the source code.

Special tests called ‘disturbance tests’ are used to test the functionality of atomic operations. These tests are also called recovery tests or tear save tests. The test consists of sending a suitable command to the smart card that causes it to initiate an atomic operation. At a certain point in time during the atomic operation, power to the card is interrupted (see Figure 15.7). After this, the processed data is checked to see whether the atomic operation maintained data consistency. In such tests, power is interrupted at various times during the course of command processing instead of only at one specific time. In order to obtain valid test results, in each successive test the interruption time is incremented by approximately half the EEPROM or flash memory page write time. This provides a very good test of the effectiveness of atomic operations. However, the number of tests required is fairly large. With a typical time increment of 1 ms, 1000 tests are necessary to fully test a command with a processing time of 100 ms.

#### 15.4.2.4 Whitebox test

Whitebox tests are sometimes called ‘glassbox tests’, which clearly describes the underlying concept. With this type of test, all internal data structures and processes are known and can be fully understood. The relevant program documentation is used to design and generate the tests, but the specification is always the decisive reference document. For decades, program flowcharts and Nassi–Schneidemann diagrams (structograms) have commonly been used to document programs, and they also form the basis for evaluating the internal functions of the

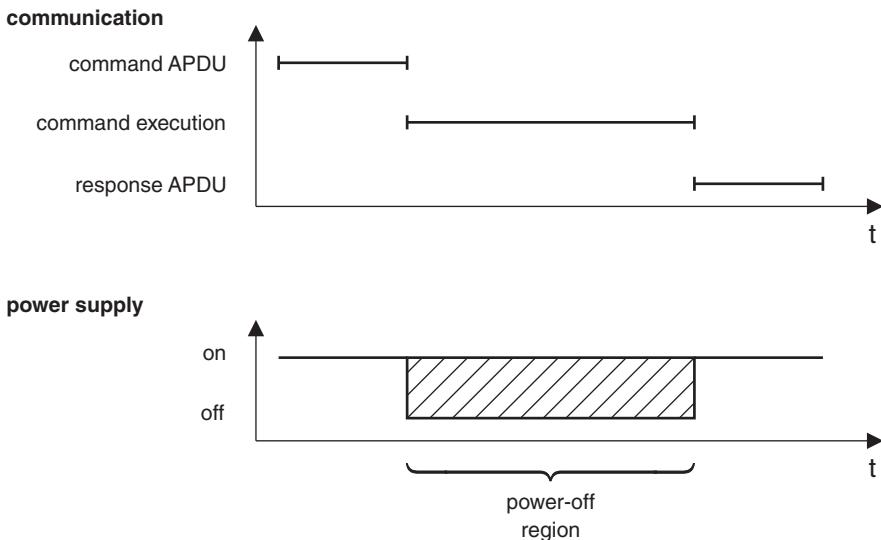


Figure 15.7 Timing diagram of disturbance testing for an atomic operation

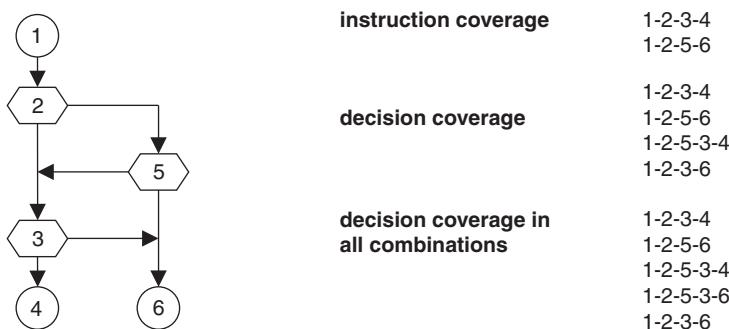


Figure 15.8 Example of the number of possible execution paths through a program flowchart for testing with statement coverage and decision coverage

software in a whitebox test. With object-oriented languages such as Java, the Unified Modeling Language (UML) has become the prevalent form of representation. It is also well suited to describing the architecture of smart card software.

As the exact program processes are known, it is reasonable to wish to test all possible execution paths of the software (see Figure 15.8). There are several ways to go about this. One of them is statement coverage, in which every instruction in the program is executed at least once. This is a very easy way to discover whether the program contains dead code that is never used, but it is not sufficiently strong to ensure that the desired functionality is present. A better method for this is decision coverage, which involves traversing all decision nodes in the program code at least once for each possible option.

Recognizing these internal program processes during dynamic testing requires either a sophisticated emulator for the smart card microcontroller concerned or instrumentation of the

**Table 15.2** Comparison of the number of possible test cases with various coverage methods in whitebox testing. A 200-byte smart card command interpreter is used here as an example. The calculated values are based on an assumed average command processing time of 30 ms, including data transfer

Coverage method	Number of possible test cases	Test duration
One million random input values	1000 000	≈ 8 h
Command coverage	10	≈ 0.3 s
Decision coverage	50	≈ 1.5 s
Decision coverage with all possible options	50 000 000	≈ 17 days
Decision coverage with all possible options and a five-byte header	$\approx 1.1 \times 10^{11}$	≈ 1 000 years
Input coverage with equivalent classes	15	≈ 0.5 s
Output coverage of return codes	6	≈ 0.2 s

program under test. Instrumenting a program means inserting special program code ahead of every jump instruction, branch instruction and function call in order to collect location and parameter information when the program is run. This information can be evaluated statistically and graphically by an analysis program. Unfortunately, the additional program code alters the timing characteristics of the program, and in the worst case, it can even alter the behavior of the program. This must be borne in mind when this method is used.

An extension of the decision coverage method is to traverse all program decisions in all possible combinations once for each combination. This covers all possible execution paths. However, due to time constraints this is only possible with very small programs having a code size of several hundred bytes. Even with programs with a size of around 1000 bytes, it is not possible to test all possible combinations in a reasonable length of time.

Table 15.2 illustrates this in summary form, using a typical smart card command interpreter as an example. The purpose of this module is to identify a command located in the card's receive buffer by means of its class and instruction bytes and then check the P1, P2, L<sub>c</sub> and L<sub>e</sub> parameters. This routine has a program code size of around 200 bytes with 18 branches. The possible output values consist of five return codes and calls to 26 different command processing routines.

Two other path coverage criteria are often used, in particular for testing smart card operating systems: input coverage and output coverage. The objective here is to generate all possible input or output values. The output values are often restricted to the available return codes, as otherwise the number of options would be too large.

However, a test strategy should not focus only on the various coverage metrics, since they are far from sufficient for discovering all defects in the program code. According to Peter Liggesmeyer [Liggesmeyer 02], even with 100 % statement coverage only around 18 % of the defects in the code will be discovered. Testing with 100 % branch coverage, which is nearly impossible in practice, would discover only around 34 % of the defects. This rather dramatically illustrates the fact that purely coverage-driven test methods will not lead to the desired low level of defects. As a rule of thumb, the degree of code coverage is typically assumed to be somewhat more than 90 % in operating system testing and approximately 95 % in application program testing. Significantly higher values are usually not feasible with tests

using APDU-driven commands, in part because the program code includes security functions that cannot be accessed via the APDU interface.

As the number of possible input values can easily achieve a magnitude that makes testing impractical due to the multitude of input values or the amount of time required, equivalence classes are commonly used. This approach reduces the very large number of possible input values to a few values that can be tested in a reasonable length of time. Equivalence classes are formed by selecting boundary cases on either side of the decision range, along with a value in the middle of the range.

For example, if the smart card command interpreter allows the P1 byte to have values in the range of 20 to 50, the equivalence class can be formed by taking 19, 20, 50 and 51 as the boundary values and 35 as the midrange value. This set of values verifies the essential query conditions of the program. After this test, it can be assumed with a relatively high degree of confidence that parameter range checking has been implemented correctly.

Especially when software is programmed using a language close to the hardware level, it is unfortunately necessary to take the properties of the target hardware into account when defining equivalence classes. For example, all arithmetic operations that can cause an overflow or underflow in the processor due to the computer architecture (8, 16, or 32 bits) must be taken into account when forming equivalence classes. This is the only way to ensure that underflows and overflows are handled correctly in the program.

Whitebox tests are often used for module testing during smart card development. In such tests, special test commands are used to supply input data to completed software modules loaded into a smart card, and the results of processing by the software modules are retrieved using other test commands. The actual results are then compared with the expected results. The disadvantage of whitebox tests is that they are dependent on the specific software implementation. Under certain conditions, this requires suitable adaptation of the tests when the source code is modified, even if the functionality remains unchanged.

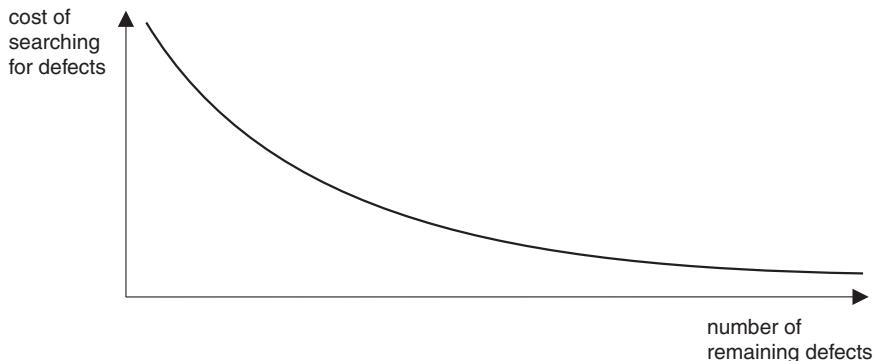
#### 15.4.2.5 Greybox test

Greybox tests are hybrid combinations of blackbox and whitebox tests. With a greybox test, some parts of the software are known, such as the internal program processes. In the smart card realm, greybox tests are used primarily in the integration phase, since they provide a quick and effective way to detect and correct defects in the interactions between individual system components. Naturally, this requires the key management system to provide suitable test keys, which are public. After this part of integration testing has been successfully completed, the results can be checked using the real keys (lifekeys).

#### 15.4.3 Dynamic testing of operating systems and applications

It is important to realize from the start that software testing can only prove the presence of defects, not their absence. Assuming that only roughly half of the defects that are present are actually found, it is apparent that programs typically have a number of weaknesses.

In practice, a defect rate of 0.7 per 100 lines of code after completion of testing is often assumed for assembly language programming. If we take the previously mentioned value of 30 000 lines of source code for a smart card operating system as a reference and deduct two-thirds of this value as comments, we can calculate that there will still be approximately



**Figure 15.9** The cost of searching for defects versus the number of remaining defects

70 defects in the fully tested and released operating system. In fields where security is critical, such as military and medical technology, it is assumed that, despite the enormous effort expended on testing and quality assurance, there are still four undiscovered defects for every 10 000 lines of source code [Thaller 93]. Although most undiscovered defects are latent defects that never manifest themselves in actual use, under the right conditions a single defect is all it takes to broach all the security barriers of a smart card operating system. It is highly beneficial to constantly bear this in mind as a reason for conducting careful and well-considered testing.

Of course, there are natural limits to testing. Particularly in commercial projects, as opposed to research projects, the available time and maximum affordable cost are strong limiting factors. In addition, testing becomes more and more difficult and expensive as the number of defects present in the software decreases (see Figure 15.9). At some point, the search for the last remaining defects must end due to fundamental limits on the available time and resources.

When a new version of the software is released, it can initially be assumed to contain fewer defects because there has been an opportunity to analyze and correct defects discovered in use. Interestingly enough, this reduction in the number of defects does not persist indefinitely. Instead, the number of defects is commonly seen to reach a minimum around the second version, after which it increases. This arises from the simple fact that the necessary corrections are based on the original specification and source code. From a certain time onward, which can vary, it is likely that correcting a defect will generate one or more new defects. Consequently, it is often significantly better to start from scratch after a certain number of versions than to continue building on an outdated high-level design and repeatedly upgraded program code. Incidentally, this is true in almost all areas of technology.

In accordance with the ‘Standard for Software Unit Testing’ (IEEE 1008), three test levels can be distinguished for dynamic testing. The first is the basic test level, which essentially tests the basic functions of the individual commands with successful execution. The second level, the capability test, encompasses boundary values and unsuccessful execution. The third level is the behavior test, in which commands are tested in various combinations.

#### 15.4.4 Test strategy

There is a major difference between testing a new operating system and testing a new application. When a smart card operating system is tested, the entire program code must be tested

in a wide variety of application scenarios. This requires a large number of different tests. With a new application, a smaller number of tests are necessary, depending on the data to be used and any program code in applets that is intended to be used for the application.

When a new operating system must be tested, several test applications similar to typical real applications are usually generated. This essentially amounts to creating equivalence classes for the usual applications. These equivalence classes form the basis for performing the individual tests.

The approach described below has become established over the course of several years in a wide variety of projects as a strategy for testing new smart card operating systems. Testing always begins with the data transmission functions, since they form the basis for all other activities. Following this, all available commands are tested. If an application is involved, the next step is file testing. If all of these tests are completed successfully, testing of the specified processes can begin.

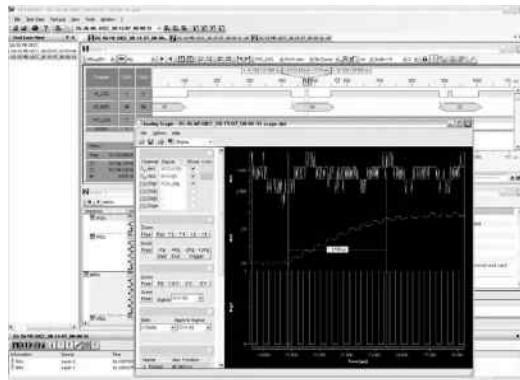
In order to provide an overview, a selection of possible tests in the customary sequence are described here. This list does not pretend to be complete, and it is only meant serve as a detailed illustrative example. The objective of the listed tests is to check the essential general parameters of a new operating system, including one or more applications.

### Data transmission tests

- ATR (parity error detection, character repetition if  $T = 0$  is available, ATR structure and contents);
- PPS (PPS structure and contents);
- data transmission test at OSI layer 2 (start bit, data bits, stop bits, divisor, data transmission convention);
- $T = 0$  transmission protocol test (parity error detection and character repetition, various processes);



**Figure 15.10** Screen display of a software tool for verifying communication between a terminal and a smart card at the physical and logical levels. The upper window shows the communication process at the logical level, while the two lower windows show the commands and associated parameters recognized in the communication process (Reproduced with permission from Comprion)



**Figure 15.11** Screen display of a software tool for verifying communication between a terminal and a smart card at the physical and logical levels. The open window in the foreground shows the communication process with the smart card at the electrical level (Reproduced with permission from Comprion)

- T = 1 transmission protocol test (CWT, BWT, BGT, resync, error mechanisms, various processes);
- secure messaging.

### Testing available commands

- test all possible class bytes;
- test all possible instruction bytes;
- test all available commands using equivalence classes for the supported functionality.

### Testing available files

- test whether all files are present in the proper locations (MF, DF, EF);
- test for correct file size;
- test for correct file structure;
- test for correct file attributes;
- test for correct file contents;
- test the specified access conditions (read, write, block, unblock, etc.).

### Testing available processes

- test the specified state machines (such as for the command sequence).

### Other tests

- typical processes during use (use case test);
- smart card lifetime with regard to wear of nonvolatile memory;
- compatibility of the operating system and application with existing smart cards;

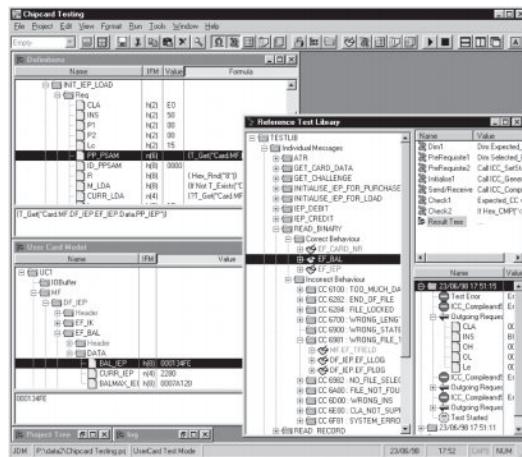
- measure the processing speed of commands;
- measure the processing speed of typical scenarios;
- disturbance testing with typical scenarios;
- combination of processes;
- nonvolatile memory usage (hot-spot analysis).

Incidentally, it is not sufficient to test command processing by simply checking the return code for correctness after the command has been executed. The test must verify correct implementation of the command in the smart card, as otherwise it is not possible to draw any reliable conclusions from the test results. For example, consider a simple command such as UPDATE RECORD, for which it is by no means sufficient to check for a return code of '9000'. A high-quality test must also check whether the data sent to the smart card with the UPDATE BINARY command is actually present in the written data, in the right location and without errors. Such considerations are what make this sort of testing rather complicated, since many mechanisms inside the smart card can only be tested indirectly from outside.

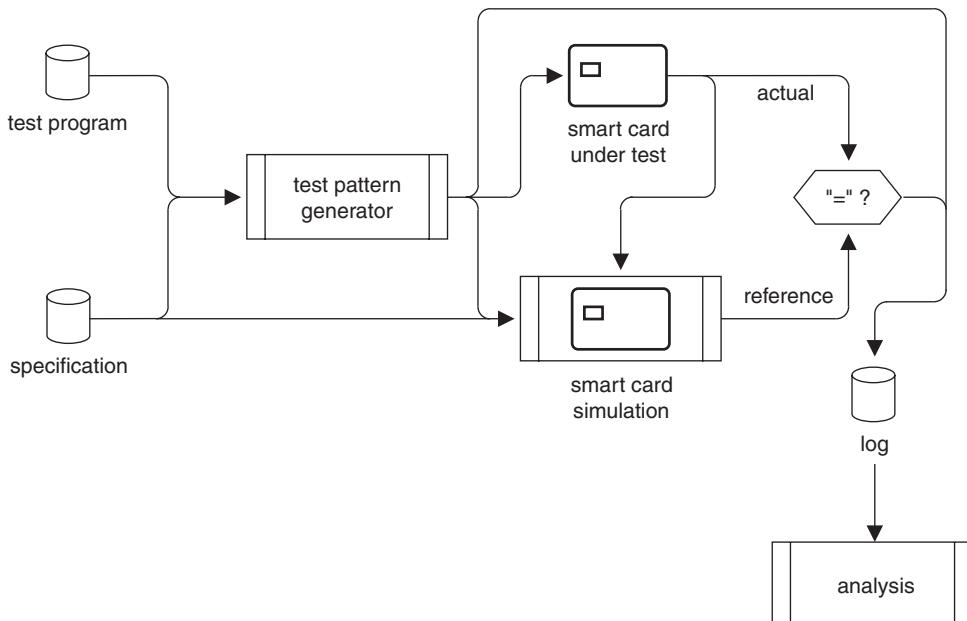
As can easily be imagined, even if equivalence classes are formed and some other minimization techniques are used, a relatively large number of individual tests are necessary. It can be assumed that 4000 to 8000 different tests must be specified to cover the essential test cases with a 20-KB smart card operating system, with tests that perform the same operation repeatedly in a loop (such as sending several hundred different values to the smart card) being counted as individual tests. The number of commands sent to the smart card in these tests can easily be around 40 000. The duration of this form of testing is in the range of one to two days. The only way to manage such a large number of tests with a reasonable level of effort is to use a suitable database, which can also store the test results.

The layout of a test tool for smart cards is shown in Figure 15.12 on the following page. As depicted in Figure 15.13, the specification of the software in the smart card, which is written entirely in pseudocode, is located in a suitable database. If the specification changes, the necessary modifications to the tests are made automatically. Another database contains all of the tests, which are defined in a high-level language that can also be read directly by a computer. The two databases supply data to a test pattern generator, which generates the commands (which means the TPDUs or APDUs as appropriate) for the card under test. A simulation of the real card, which is largely defined by the specification, runs in parallel. As some processes that run in the real card (such as random number generation) are not fully predictable, additional data must be sent to the simulated card. The real and simulated cards send their command responses to a comparator. If the responses are the same, the real card has provided the correct result, assuming that the simulation is a proper reference. All data generated during a test run is stored in a log database for subsequent manual evaluation.

The usual approach to generating tests is based on implementation of the test specifications. A relatively new approach to testing in the smart card realm is based on automatic test generation. This involves generating a full functional description of the smart card operating system or a portion of the operating system in a programming language, with this description being independent of the program code of the actual operating system. The descriptive source code is then analyzed by a program that generates tests from the code. This analysis can be configured by means of various parameters in order to generate tests with various depths and widths. This model-based approach is well suited to the generation of supplementary tests,



**Figure 15.12** Screen display of an object-oriented, database-supported tool for testing smart card operating systems and applications. The defined data elements of a command APDU (INITIALIZE IEP for Load) are shown at the upper left. Below this is the associated reference simulation of the smart card. Part of the tree structure that defines the individual tests is shown on the right (Reproduced with permission from Integri)



**Figure 15.13** Basic structure of a tool for testing smart card operating systems and applications

**Table 15.3** The usual four-part structure of an individual test case, using a test for a retry counter as an example (the counter allows at most three false attempts). This is just one test of a test suite that typically encompasses several thousand tests

Individual test case step	Associated working steps
Test preparation	<ul style="list-style-type: none"> <li>● activate the card</li> <li>● enter an incorrect PIN code twice</li> </ul>
Test execution	<ul style="list-style-type: none"> <li>● enter an incorrect PIN code</li> </ul>
Test results assessment	<ul style="list-style-type: none"> <li>● check for the correct return code</li> <li>● check whether a file access that requires this PIN is blocked by the operating system</li> </ul>
Test wind-down	<ul style="list-style-type: none"> <li>● reset the retry counter to zero</li> <li>● deactivate the card</li> </ul>

but it cannot replace tests developed manually by human testers. This is primarily due to the complex sequences necessary to discover typical defects in smart card software, which require considerable experience in defining test structures.

## 15.5 EVALUATION OF HARDWARE AND SOFTWARE

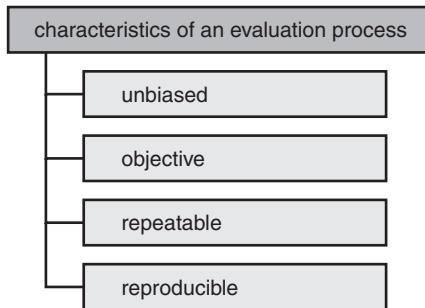
Smart cards are used in security-sensitive areas due to their ability to store data securely. However, smart cards can be used to advantage not only for the secure storage of data, but also for the secure execution of cryptographic algorithms.

In particular, the electronic payment sector is an important market for smart cards. The money flows in large systems in this sector are immense, which means that the application provider or card issuer must have a high degree of confidence in the semiconductor manufacturer, the producer of the operating system, and the smart card personalizer. The application provider needs to be absolutely certain that the software in the smart card performs the required financial transactions without any errors and that the software is free of security leaks, not to mention trap doors deliberately implemented in the software.

For example, suppose a secret command could be sent to the smart card to read out the PIN and all secret keys. In the case of a GSM or Eurocheque card, the attacker would then be able to clone any desired number of cards and sell them as fully functional cards.

The security requirements involve not only the production of the smart cards, but also card initialization and card personalization, since the secret keys and PIN codes are loaded into the cards in these stages. With regard to security, the card issuer must place considerable trust in the card supplier.

The same applies to the fundamental security of the software in the smart cards. Problems can arise even if a trap door has not been intentionally implemented in the software to allow data to be ferreted out. A software malfunction could easily make it possible to read data from the card or write data to the card using a combination of commands that is not used in normal operations. Although the likelihood of this is extremely low, it is nevertheless well known that with the current state of software technology it is impossible to guarantee that programs are unconditionally free of errors.



**Figure 15.14** The four characteristics of an evaluation process

There are only two ways in which the application provider can test the trustworthiness of a product. He can test all of the features and functions of the smart card and its software himself, or he can have them tested by a trustworthy entity. In many cases, the first option is possible only to a limited extent because the provider usually does not have all the necessary technical expertise and resources for the tests. The second option, which is to entrust the tests to another entity, is currently regarded as a solution that is acceptable to all parties concerned.

This sort of testing is generally called evaluation. As summarised in Figure 15.14, evaluation is the unbiased, objective, repeatable and reproducible assessment of information technology systems (hardware and/or software) by a trustworthy entity in accordance with a defined set of specifications (criteria catalog). The information technology system to be evaluated is called the target of evaluation.

This issue has existed for many years in the area of software and system development for military applications. It is thus not something new in the smart card world. In order to make the trustworthiness of software products objectively measurable, which means to specify a set of metrics for this property, in 1983 the US National Computer Security Center (NCSC) published a catalog of criteria for evaluating the trustworthiness of information technology systems. The NCSC was founded in 1981 by the US Department of Defense (DoD). *Trusted Computer System Evaluation Criteria* (TCSEC) was published in 1985. This book had an orange binding, so it is generally known as the ‘Orange Book’. These criteria act as guidelines to the NCSC for the evaluation of information technology systems.

The TCSEC was used worldwide as a model for practically all criteria catalogs in the information technology sector. Specifically European criteria were defined in Europe, although they were based on the TCSEC. They were first published as *Information Technique System Evaluation Criteria* (ITSEC) in 1990. Following this, the Common Criteria (CC) were generated to provide a uniform worldwide standard for testing the correctness of software. They can be regarded as a distillation of the essential aspects of the TCSEC and the ITSEC. The Common Criteria are better oriented to the evaluation of software than the TCSEC or the ITSEC. The first version of the CC was published in 1996, and since 1999 they have been available in the form of the international ISO 15408 standard.<sup>3</sup> In contrast to the ITSEC, which has six levels, the Common Criteria have seven levels of trustworthiness. The transition from an evaluation based on the TCSEC or ITSEC to an evaluation based on the Common Criteria is relatively

<sup>3</sup> The TCSEC, ITSEC and CC are available at no charge from many Internet sites (e.g. the CC at [NIST])

easy because all of these criteria catalogs have many features in common. The current version of the Common Criteria is 3.1 (issued in September 2006), and they have become the most important foundation for the evaluation of hardware and software in the smart card sector.

An evaluation process has four characteristics, regardless of the methodology that is used. First, the process must be unbiased, which means that the evaluator must not have any prejudices with respect to the target of evaluation or its producer. The second characteristic is that the evaluation process must be objective and structured to minimize personal assessments. The third characteristic is that the same result must be obtained if the evaluation process is repeated. The final characteristic is that the evaluation process must be reproducible, which means that a different evaluator or evaluation authority must arrive at the same conclusion.

One of the most important aspects of every evaluation is specifying the security targets for the target of evaluation (TOE). The target of evaluation is the object to be evaluated, and the security targets describe the mechanisms to be evaluated. Incidentally, an evaluation can be dramatically simplified by suitable selection of the security targets, since elements that are critical with respect to security can thereby be excluded. This is simply a subterfuge that can be used to achieve a high evaluation level as quickly and inexpensively as possible. Naturally, the actual security suffers as a result.

### 15.5.1 Common Criteria (CC)

The Common Criteria consist of three parts: Part 1, ‘Introduction and General Model’, describes the general basis of an evaluation. This includes specifying the target of evaluation (TOE), which is the information technology system to be tested by the evaluation. This part also presents the protection profile (PP), which is an implementation-independent set of security requirements adapted to the specific application area for certain targets of evaluation. For example, Eurosmart [Eurosmart] has its own protection profile for smart cards. The security requirements list the threats and corresponding defense requirements in a fairly generalized form.

The second part of the Common Criteria is the security functional requirements, which consists of a very extensive list of requirements on the functionality of the target of evaluation. The third part consists of detailed descriptions of the security assurance requirements. This part also contains a description of the evaluation assurance levels (EALs). These seven hierarchical levels designate the degree of trustworthiness of a target of evaluation within the context of an evaluation. Level 0 is the lowest level of trustworthiness, while Level 7 is the highest level. These levels are summarized in Table 15.4 on the next page.

The amount of effort does not increase linearly with each EAL level, but instead nearly quadratically. This means that it takes twice as much effort to go from level EAL2 to level EAL3 as it does to go from level EAL1 to level EAL2. The consequences of this are naturally most pronounced with evaluation levels EAL5 to EAL7. Complete evaluation of a medium-sized smart card operating system at the Common Criteria EAL7 level can easily take several years and cost several million euros.

In the evaluation process, a fundamental distinction is made between informal, semi-formal, and formal methods. An informal description of a function can readily be compared with a written description in this book. The description of the Small-OS operating system in Section 13.17 on page 521 is rudimentarily semiformal. By contrast, a formal description can be tested logically and is generated using formal notation.

**Table 15.4** The seven trustworthiness levels (evaluation assurance levels) of the Common Criteria

Criterion	Description
EAL1	Functionally tested
EAL2	Structurally tested
EAL3	Methodically tested and checked
EAL4	Jethodically designed, tested, and reviewed
EAL5	Semiformally designed and tested
EAL6	Semiformally verified design and tested
EAL7	Formally verified design and tested

**Table 15.5** The Common Criteria function classes. Each class describes a certain basic function of the security architecture, which must be separately assessed in the evaluation

Function class	Description
FAU	Security audit
FCO	Communication
FCS	Cryptographic support
FDP	User data protection
FIA	Identification and authentication
FMT	Security management)
FPR	Privacy
FPT	Protection of the TSF
FRU	Resource utilization
FTA	TOE access
FTP	Trusted path/channel

In all of the function classes (see Table 15.5), the formal requirements on the development process and the development environment are specified in a very abstract manner. In addition, the function classes contain information and specifications regarding the operating documents and the eventual operating environment. This information is generated in a form that is suitable for use with all possible forms of software development in information technology systems.

In Common Criteria evaluations, the mechanism strength is often stated in addition to the evaluation level. This indicates the robustness against attacks. Three mechanism strengths are defined: low, medium, and high.

'Low' characterizes protection against random, unintentional ingress into a secure environment. 'Medium' means that there is protection against attackers with limited resources. The highest mechanism strength ('High') means that there is protection against attackers with very good technical knowledge and resources. High mechanism strength is usually used at Common Criteria level EAL4 and above. This is indicated by a + sign in the designation, such as EAL4+.

In case of microcontroller hardware used in the smart card environment, the trustworthiness level that is typically aimed for in evaluations is EAL4+ or EAL5+. The level that is usually aimed for with operating systems is EAL4+.

**Table 15.6** Security evaluation of the specifications and source code of a smart card operating system in accordance with the ZKA criteria (based on Stefan Rother [Rother 98a])

● Operating system commands	● Encryption and decryption
● Application commands	● Key derivation
● Communications mechanisms	● Signature generation and verification
● Memory management and resource management	● Key management
● File attributes and file access privileges	● Authentication mechanisms
● Checksum algorithms	● Random number generator

Evaluation of a slightly modified version of a product that has already been evaluated is often necessary in certain cases. This can be handled by a ‘delta evaluation’, in which the new version is evaluated on the basis of the original evaluation. This saves substantial time and expense with the same level of quality.

In place of separate evaluations of the hardware and software, a composite evaluation can be used to assess them as cooperating components of a single entity that can complement each other in defending against attacks. This form of evaluation is sometimes used in the smart card environment.

### 15.5.2 ZKA criteria

In addition to international evaluation standards such as TCSEC, ITSEC and CC, several specific evaluation standards have become established for large smart card applications. Two examples of such standards are the Visa criteria for security tests and the German ZKA criteria (see Table 15.6). Compliance with the ZKA criteria is mandatory in Germany for Eurocheque cards with chips, and all smart card operating systems for this application are tested against these criteria by authorized evaluation bodies.

An evaluation based on the ZKA criteria consists of reviewing the provided documentation as well as examining the software and the hardware. The main advantage of the ZKA criteria relative to the Common Criteria is that the ZKA criteria are specifically applicable to smart cards, while the Common Criteria apply to any desired software. Table 15.7 on the following page lists the 13 ZKA criteria with associated explanations.

In summary, the ZKA criteria probably offer the best possible security for smart cards at present because they fulfill the stringent requirements of a large financial transaction system and are specifically tailored to the particular interests of systems based on smart cards.

### 15.5.3 Additional evaluation methods

In addition to the methods described above, there are a variety of specialized methods used in the smart card domain. For example, certification using the MasterCard compliance assessment and security testing (CAST) program is used in the payment systems sector. Visa has similar specifications for testing smart cards.

The requirements of the FIPS 140-2 standard are occasionally taken into account in evaluations, alongside the Common Criteria. This standard specifies four possible security levels for security modules, which can also include smart cards, and provides detailed descriptions

**Table 15.7** The ZKA criteria for evaluating a smart card system (based on Stefan Rother [Rother 98b])

Criterion	Explanation
Component authentication	It must be possible to authenticate the security components of the system
Message integrity	Security-related data that is exchanged between the components must be protected against manipulation
User authentication	Certain functions may be executed only after the user's PIN has been entered correctly to authenticate the user
Confidentiality of PINs and keys	PINs and keys may never be transferred in plaintext outside a secure environment. If PINs and keys are processed or stored in components, these components must be protected against unauthorized reading or modification. The system must prevent discovery of the PIN by means of an exhaustive search
Logging	All events relevant to security must be logged in the components concerned. The logs must be protected against manipulation
Key management	There must be mechanisms for the distributing, administering and changing keys. Only dynamic keys may be used with symmetric cryptographic algorithms, and keys must be separated according to their intended use
Hardware	All actions relevant to security must be protected against unauthorized access
Organizational measures in production and personalization	Only the evaluated program code with the described parameter values may be used in actual operation
Process security	The correctness of the processes and the data flows in the evaluated program is ensured by examination of the source code
Other applications	Additional applications must not have any effects that endanger security
Encryption methods	The cryptographic algorithms that are used must correspond to the technical state of the art and may not be based on maintaining the secrecy of the method
Unambiguous representation	Each security component must be clearly identifiable in the overall system
Personnel requirement	Only suitable persons should have access to the components and mechanisms relevant to security

of seven requirement areas related to security. The contents of this standard are very practically oriented and also address the details of technical implementation, such as criteria for the quality of random number generators.

#### 15.5.4 Summary

In a real smart card project, evaluation is usually approached as follows: the card issuer and the producer of the operating system jointly generate the operational requirements and specify

the threats to be taken into account. They then agree on the necessary evaluation level. Next, the operating system is produced in accordance with this evaluation level using a suitable method, and the necessary operating system documentation is generated. In the final step, the completed operating system with all of its components can be evaluated with respect to the previously agreed evaluation level by a suitable independent entity called the evaluation authority agreed level.

A Common Criteria evaluation has certain benefits for the card issuer and the application provider because they can be assured that with regard to many security aspects, the design and implementation of the essential mechanisms are clearly defined and functionally adequate. However, this assurance comes at the price of certain drawbacks in the dynamic smart card market. The development time is prolonged considerably by the necessary evaluation, even at the lowest evaluation levels. The additional documentation also increases the development cost, which is ultimately reflected in the selling price of the operating system.

Even with an evaluated product, it must always be borne in mind that evaluation only guarantees that all processes and mechanisms always function as described in the relevant documents. Evaluation does not mean that the evaluation authority has fully tested the product, but only that it has reviewed the documentation it received and examined the source code and object code according to the relevant evaluation level. It is equally important for the target hardware to ensure a level of security equivalent to that of the evaluation level. There is no benefit at all in having error-free, secure software if the hardware provides a back door that can be used to bypass the software at minimal expense. Evaluation increases the probability that the software behaves as intended, but it does not provide a guarantee of freedom from errors or defense against all imaginable forms of attack.

# 16

## Smart Card Security

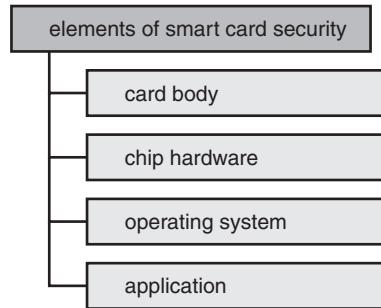
The main advantage of smart cards relative to other data storage media, such as magnetic-stripe cards and memory sticks, is that they provide secure storage of confidential data. An essential requirement for this is the availability of chip hardware designed and optimized for this purpose, along with suitable cryptographic algorithms for protecting confidential data. However, security depends on more than just special microcontroller hardware and algorithms implemented in the operating system software. The security of the smart card application and the design principles used in the system development process are also of fundamental importance. This chapter provides a synopsis of the essential principles, methods and strategies for producing secure smart cards and secure smart card applications.

The essential characteristic of a smart card is that it provides a secure environment for data and programs. If the amount of effort needed to read data from a smart card were not so large, it would essentially amount to nothing more than a memory stick with a different interface.

Of course, it is practically impossible to configure a complete system, or even a smart card, to provide perfect security that can defend against everything and everybody. If sufficient effort and expense is devoted to an attack, every system can be breached or manipulated. However, every potential attacker performs a sort of cost/benefit analysis – either consciously or unconsciously – and the reward of breaking a system must be worth the time, money and effort necessary to achieve this objective. If the result in terms of money or peer esteem is not worth the effort, nobody will invest much effort in breaking a system or a smart card.

The security of a smart card is ensured by four components (see Figure 16.1). The first component is the card body in which the microcontroller is embedded. Many of the security features of the card body are not only machine readable, but can also be visually checked by humans. The techniques used for these features are not specific to smart cards, but are also used with other types of cards. The remaining components – the chip hardware, the operating system, and the application – protect the data and programs in the smart card microcontroller.

The security of a smart card is assured only when all of these components are present and their defense mechanisms are in good order. If the card is used exclusively in an environment where it is not subject to human verification, the card body component is not necessary. The three components that are independent of the card body, however, are indispensable for the physical and logical security of a smart card with respect to attacks. If any of these components



**Figure 16.1** Classification of smart card security components

fails, or if any of them does not meet the applicable requirements, the smart card is no longer secure, since these components are coupled to each other in a logical AND relationship.

The useful life of a smart card is generally three years. The challenge to manufacturers of smart card microcontrollers and producers of smart card operating systems is to stay at least this far ahead of potential attackers. This allows the consequences of attacks to be minimized or avoided by employing suitable countermeasures. However, it is not always possible to maintain such a lead, which is why it is always important in the design and development of application architectures to ensure that a successful attack on a single smart card cannot compromise the entire system.

This is especially important in light of the fact that there are large smart card applications in which the cards are not replaced regularly. They include the smart card application with the highest number of cards in use: the SIM cards used in the GSM mobile telecommunication system. It is entirely possible for SIM cards to be used for 10 to 15 years. If this life expectancy is compared with the history of known forms of attack listed in Table 16.2 on page 676, it is easy to see that reliable protection against attacks over such a long period is difficult to achieve. However, the GSM system has the advantage of being a pure online system, which enables a flexible response to known forms of manipulation.

If an attacker pursues the general objective of producing new cards, two different approaches are available. The attacker can concentrate on producing clones, which means producing exact copies of the attacked cards. However, clones are easy to recognize in online systems. The other approach is to create valid new cards that are not copies of existing cards. Newly created cards produced by an attacker can only be detected in an online system by using a list of all usable cards with their unique identifiers. Such a list is called a whitelist.

## 16.1 CLASSIFICATION OF ATTACKS AND ATTACKERS

The primary problem faced by all information technology systems subject to attack is the avalanche effect that often occurs after a successful attack. If a printer manages to produce a large number of counterfeit banknotes in sufficient quality, this is a matter of concern for the affected national bank, but in practice it does not cause inflation of the national currency. In the first place, a counterfeiter would never be able to produce enough banknotes for this, and in the second place, it is very risky to put a large number of counterfeit notes into circulation.

The situation with electronic money is somewhat different. As it consists of nothing more than immaterial data, in practice it is impossible to distinguish between an original and a copy. In addition, when a new counterfeiting method becomes known an avalanche effect can occur due to copycat counterfeiters. This can be seen quite clearly with counterfeit prepaid telephone cards, which have been produced in considerable quantities. Some network operators can only defend themselves against this form of attack by restricting the calling destinations of card phones.

If a design error or weakness in a major smart card system becomes known, it can be assumed that this information will be distributed over the entire world via the Internet within a few days or weeks. Very quickly, suitable software and any necessary hardware will be offered on the Internet, usually with complete documentation, making it easy for others to reproduce the original attack. This software is also usually provided in the form of source code, allowing it to be enhanced by others, which also generally happens. This leads to a very rapid evolution of the hardware and software, which are quickly optimized to suit their intended purpose.

In the following description, we attempt to present a systematic classification of possible forms of attack and attackers. The emphasis naturally lies on the information technology aspects of smart cards, rather than the security features of the card body that can be verified by humans. This classification allows potential forms of attack to be evaluated so that suitable precautions can be taken. As is well known, it is considerably easier to defend against known forms of attack than unknown forms.

Our classification of the forms of attack is based on the ISO 13491-1 standard, which describes the concepts, requirements, and evaluation methods for cryptographically secure devices in the banking sector.

### 16.1.1 Classification of attacks

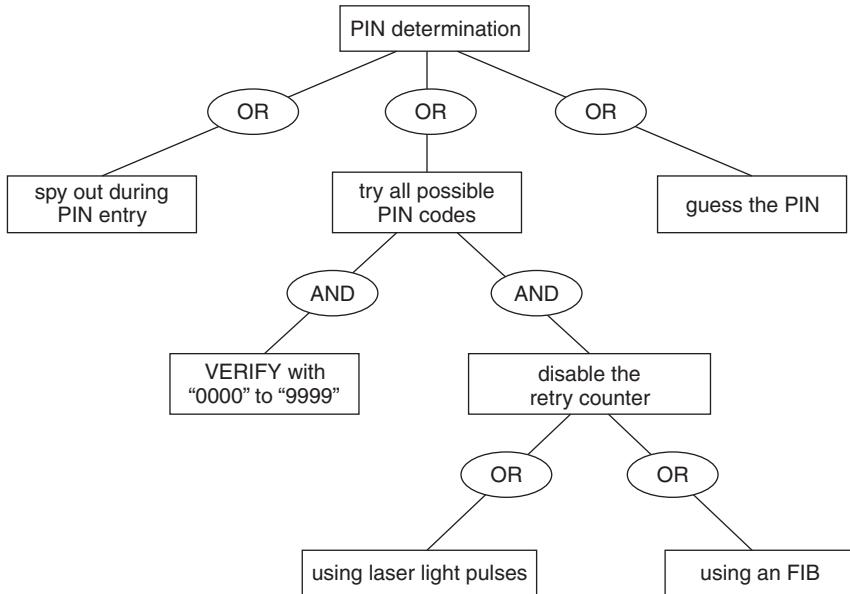
There are various approaches to the classification of attacks on smart cards. For instance, in a security evaluation all possible forms of attack are grouped by card life cycle phase and described formally [IC Protection 97, Isselhorst 97]. This yields long lists that identify every conceivable form of attack in each phase. The actual evaluation consists of examining each item in the list to see whether the system or smart card can defend against it.

An alternative approach is to perform an attack tree analysis (see Figure 16.2) in the same way as a fault tree analysis [Schneier 99]. This is also called a threat tree analysis, and it is very useful for detailed evaluations and describing dependencies.

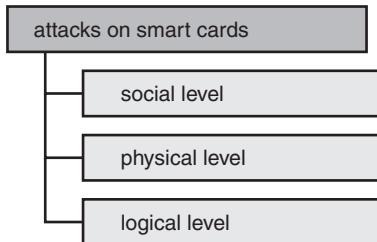
The root of an attack tree describes a specific goal, while the leaves describe the individual steps of the attack in combination with AND and OR branches. The AND branches describe attack steps that are collectively necessary for achieving the objective (logical AND relationship). By contrast, the OR branches describe alternative attack steps that can be used to achieve the objective, which means that they have a logical OR relationship.

For simplification, an attack tree can be divided into subtrees with reusable subgoals. With this approach, a rather large number of attack trees are necessary to analyze all potential forms of attack on smart cards.

In this book, we use a different classification in order to present the subject in as realistic a manner as possible and illustrate the ping-pong game of attack and defense. In addition, our intention here is to present a general overview of forms of attack and defense that is not specific to any particular system.



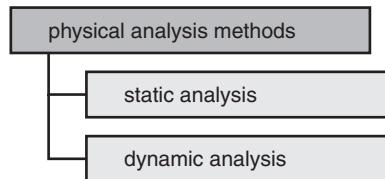
**Figure 16.2** A small extract from a smart card attack tree with the goal of determining the PIN code



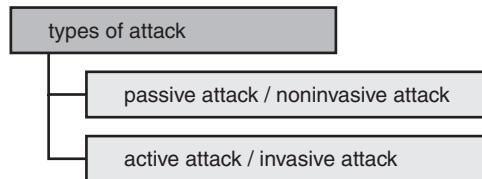
**Figure 16.3** Classification of attacks on smart cards

In principle, attacks on smart cards can be divided into three types as shown in Figure 16.3: attacks at the social level, attacks at the physical level, and attacks at the logical level. Naturally, mixed forms of attack also occur in practice. For example, an attack at the physical level may prepare the way for a subsequent attack at the logical level, which for example is the case with differential fault analysis.

Attacks at the social level are primarily aimed at people who handle smart cards. They may be chip designers employed by semiconductor manufacturers, software designers, or further on in the card life cycle, cardholders. These attacks can only partially be countered by technical means. They must primarily be countered by organizational means. Surreptitiously acquiring a PIN by watching it being keyed in can readily be prevented by visual screens on either side of the keypad. Attacks at the social level aimed at smart card programmers are rendered pointless by making the algorithms public, as well as by having their program code evaluated by other parties. In this case, security depends only on the secret keys, and the knowledge possessed by software developers is of no use to an attacker.



**Figure 16.4** Classification of physical methods for analyzing smart card microcontrollers



**Figure 16.5** Classification of forms of attacks on smart cards

Attacks on smart cards at the physical level usually require a certain amount of technical equipment, since it is necessary to obtain physical access to the smart card microcontroller hardware in some way in order to analyze it (see Figure 16.4). Such attacks may be static, which means that no power is applied to the microcontroller, or dynamic, which means that the microcontroller is operating (see Figure 16.5). Static physical attacks impose no time restrictions on the attacker, who can proceed at his own pace. With a dynamic attack, by contrast, the attacker must have access to sufficiently fast measuring and recording equipment.

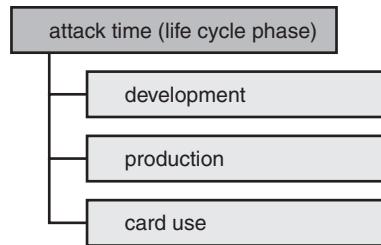
Most currently known successful forms of attack on smart cards take place at the logical level. These attacks are based on pure contemplation or computation. This category includes traditional cryptanalysis as well as attacks that exploit known weaknesses of smart card operating systems and Trojan horses in the executable code of smart card applications.

As with the cryptanalysis of cryptographic protocols, these attacks can be divided into passive and active types. In a passive attack, the attacker analyzes the ciphertext or cryptographic protocol without modifying it, and may for example make measurements on the semiconductor device. This is called a noninvasive attack. By contrast, an active attack (also called an invasive attack) involves manipulation of the data transfer or the microcontroller.

The smart card life cycle phases defined in the ISO 10202-1 standard can be used to classify the timing of potential attacks.<sup>1</sup> However, this leads to verbose results, so for the sake of readability we have undertaken a simplification and classified the attacks into three phases: development, production, and card usage (see Figure 16.6).

Attacks during the development phase encompass system design, chip development, operating system development, and application generation. Production refers in general to all processes used to produce hardware. This covers the entire range from wafer fabrication by semiconductor manufacturers to card personalization and distribution of the cards to the users. Card usage is the phase in which the smart cards are in the field, which means when they are being used by cardholders.

<sup>1</sup> See also Chapter 14, ‘Smart Card Production’, on page 567



**Figure 16.6** Classification of the timing of potential attacks

### 16.1.2 Consequences of attacks and classification of attackers

In order to realistically estimate the strengths and weaknesses of attackers with respect to the security of smart cards, it is important first to have at least a rough idea of the possible types of attackers. This information can then be used to help devise defense strategies and defense mechanisms.

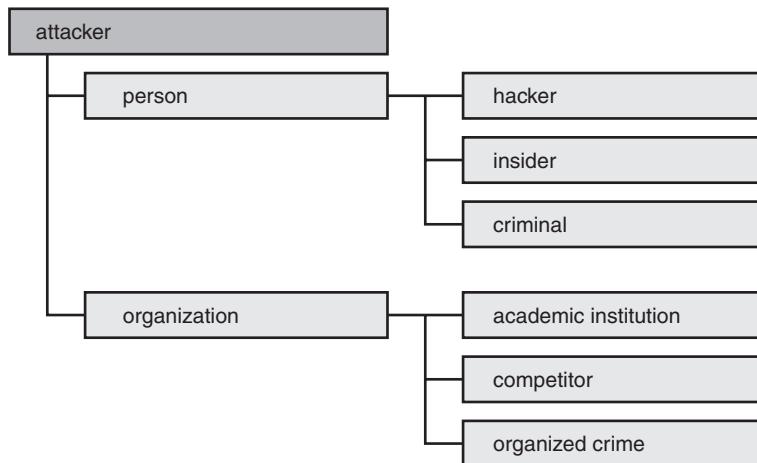
Typical attackers usually have one of two basic motivations. The first motivation is simple greed, while the second motivation is the desire for fame and status in a particular peer group. These two motivations have different consequences for the system operator. An attacker motivated by the desire for financial gain may, at a certain personal risk, take the approach of becoming a card issuer,<sup>2</sup> or the attacker may attempt to extort the system operator. Both approaches can be combated by the usual judicial means. If details of the attack become public, the reputation of the smart card system will be damaged. The worst damage to the reputation of a system operator occurs when a large number of cardholders lose money as a result of an attack.

Similar damage to the reputation of a smart card system can also result from attacks prompted by a compulsive desire to perform scientific research, rather than criminal tendencies. An attacker of this sort is only satisfied if the results of his or her efforts can be published in a suitable manner. The attacker is also under strong pressure to publish these results as quickly as possible, due to the well-known fact that only the first person to publish garners fame in this profession. The end result is that the system operator, with little or no warning, is confronted with the publication of a detailed description of an attack on the system. Following this, the published attack is refined step by step by other interested parties and explained in terms that can also be understood by outsiders. The peak of indignity comes when programs that carry out the attack in a fully automated manner are published.

In the spring of 1998, several GSM network operators found themselves confronted with a similar sequence of events. However, in this case the attack on the COMP 128 cryptographic algorithm, which was used for the A3 and A8 functions, did not have major detrimental effects on normal network operation. There is a particularly significant aspect of this form of attack with regard to the attacker: the attacker is regarded as the successful discoverer of a security leak, and thus as one of the good guys, and almost never faces the threat of legal action as a result.

The key conclusion that can be drawn from these scenarios is that it ultimately does not particularly matter to a system operator whether an attack comes from a good guy or a bad guy.

<sup>2</sup> This is the approach usually taken with automatically reloadable phone cards



**Figure 16.7** Classification of possible types of attackers

With a truly dangerous attack, the damage in terms of financial loss and damaged reputation is usually quite significant. In the worst case, it is necessary to shut down the system, block all existing cards, and issue new cards that are immune to the attack. In the case of a large system with several million cards, issuing new cards can take more than half a year.

The classification chart in Figure 16.7 shows the possible types of attackers based on the previously described aspects and practical experience. All of the listed types of attackers can be equally dangerous to a smart card system, but they have different capabilities and options. A typical hacker, for instance, has a moderate amount of system expertise, good creative ideas, and usually a similar group of friends. Hackers usually do not have access to a well-equipped laboratory, and their financial resources are limited. However, a competent hacker who employs a suitable approach can easily obtain access to a large amount of computing power, for example by organizing an Internet campaign.

All insiders who have very good system knowledge form a special class of attackers. They may have access to hardware and software components, and they may even be aware of weaknesses. As long as only individuals are involved, they are equivalent to hackers in terms of their resources and options. However, insiders are neither anonymous nor especially numerous, so attacks by insiders can usually be traced back to their source.

The third class of individuals who can be regarded as potential attackers consists of criminals. Although they usually do not have extensive technical knowledge, they exhibit considerable energy when it comes to obtaining personal advantages (primarily financial) as a result of their activities.

A potential source of attacks that cannot be ignored in practice consists of academic institutions, such as universities and technical colleges with their students and professors. They do not necessarily have special knowledge of specific smart card microcontrollers or applications, but they do have a large amount of generally useful knowledge. In addition, they have access to a large pool of qualified and inexpensive labor in the form of undergraduate and graduate students, as well as adequate technical equipment in their laboratories. Many of these institutions also have plentiful computing power and highly motivated people with an experimental bent.

**Table 16.1** Factors affecting the effort and cost necessary for an attack on the hardware or software of security components, based on the prerequisites for an attack

Degree of attractiveness	low	moderate	high
Necessary knowledge and skills	much	moderate	low
Necessary secrets	many	moderate	few
Necessary amount of time	large	moderate	small
Acquisition of the necessary equipment (purchase or access)	difficult	moderate	easy
Access to the components under attack	difficult	moderate	easy
Value of the result (money or reputation)	low	moderate	high

A special class of attackers is formed by competitors. They usually have considerable technical knowledge, and some of them have a very sophisticated analytical infrastructure.

Organized criminal organizations naturally represent an entirely different level of attacks on smart card systems. They have enough money to acquire – either by normal purchase or illicitly – all of the knowledge and tools necessary for successful attacks.

### 16.1.3 Classification of the attractiveness of attacks

To allow effective precautionary measures to be taken, the attractiveness of an attack should be evaluated for each relevant weakness. This can be done in an objective mathematical manner by using value analysis to compute a prioritized list of probable targets of attack. Although the scheme described here is simplified, it allows the attractiveness of various forms of attack, and thus the probable lines of attack, to be estimated relatively well. Naturally, attackers usually choose forms of attack that require the least effort and expense. Their actions are influenced, either consciously or unconsciously, by the six factors listed in Table 16.1.

Forms of attack that require relatively little specific knowledge or skills are more attractive to individuals or organizations. Attacks whose success does not require learning any secrets are also more attractive. Incidentally, maintaining several secrets does not contradict Kerckhoff's principle, which states that the security of a system should depend on the key alone rather than the cryptographic algorithm, since this principle does not imply that everything except the key should be revealed for the sake of security. Maintaining many secrets creates an enormous obstacle to mounting a successful attack.

Especially with regard to systematic key searches, the amount of time required plays an important role. The classic example is breaking a cryptographic algorithm by using a brute-force attack that would require 10 000 years on average. No serious attack could be mounted on this basis.

The attractiveness of an attack is also very strongly dependent on the equipment needed for the attack. This need not necessarily involve purchasing the equipment, since it may be sufficient to rent the equipment or acquire access to it by some other means. For example, equipment that can generate and precisely position focused ion beams costs several hundred thousand euros, but this equipment can be rented by the day at research institutes, and some students can use this equipment for free in their research work.

The availability of the components to be attacked also strongly influences the attractiveness of a particular form of attack. For example, a card-based electronic purse system can be

attacked in the user's home by analyzing the user's own card and its card-specific keys, or it can be attacked by analyzing a security module with its system-wide master keys. The difficulty with the latter approach is that access to the security module is protected by several security measures.

Incidentally, this is why smart cards for pay television are so strongly exposed to attack. Working at home, an attacker can devote as much time as desired to studying the communication processes and behavior of the smart card and trying to duplicate them with a computer or DIY circuitry, without being observed or hindered by other persons. However, if the attacker attempted something similar with a smart card terminal in a supermarket, the cashier would immediately put a stop to this. A good overview of the security of electronic money with and without smart cards can be found in [BIS 96].

The final factor, which is of decisive importance, is naturally the value of the result of the attacker's efforts. These efforts must be rewarded, either in monetary form or in the form of enhanced public prestige or standing with a peer group. From this, it can be concluded that only hackers and academic groups form a potential source of attacks on field trials of electronic purses, since there are far too few locations where the cards can be used and the businesses are mostly too small (bakeries, kiosks and the like) for any significant amount of money to be gained from an attack.

## 16.2 A HISTORY OF ATTACKS

From the beginning of smart card technology in the late 1980s to now, there has been a new approach to attacks almost every year (see Table 16.2). In the early to mid-1990s, attacks often targeted the smart card hardware. From the mid-1990s onward, there were many methods that utilized mathematical approaches in combination with analysis at the logical level. Since the turn of the millennium, there have been more and attacks using methods that previously proved successful with PCs connected to the Internet.

There is one form of attack that has again and again been used successfully and that pursues a very simple approach: an exhaustive search of the key space (brute-force attack). This approach will continue to be effective in the future because there are many systems in use for which it is undesirable or impossible to change to a stronger cryptographic algorithm due to compatibility reasons. The situation is further aggravated by the fact that the computing power available with computers linked by the Internet will increase enormously in the coming years.

## 16.3 ATTACKS AND DEFENSE MEASURES DURING DEVELOPMENT

A large number of security measures are taken right from the start of development of the microcontroller hardware and the software of smart card operating systems. Like quality, security must be built in from the very beginning of a development project; it cannot be designed into a product afterward.

With regard to attacks in the development phase, it can generally be said that access to the facilities concerned is very difficult and the required level of expertise is very high. The

**Table 16.2** Summary of typical attacks affecting systems with smart cards, in order of the date when they first became known. The listed attacks and associated primary countermeasures are described in more detail in the text

Known since	Attack	Brief description
Before 1990	Tapping data communication	Data transfer between the terminal and the card can be tapped by attaching wires to the module. The countermeasure was the introduction of secure messaging
~ 1990	Removing the passivation layer	Removing the passivation layer on top of the microcontroller is a prerequisite for physical access to the components on the microcontroller die. The countermeasure was the introduction of passivation detectors in microcontrollers
~ 1990	Manipulation of data transfers	Data transfers between the terminal and the card can be manipulated as desired by electrically insulating the module contacts and attaching suitable wires to the module. The countermeasure was the introduction of secure messaging
~ 1991	Erasing the EEPROM using UV light	Actions such as resetting counters to their initial values can be performed by using UV light to erase the EEPROM. The countermeasure was the introduction of light sensors in microcontrollers
~ 1991	Equivalent circuits for memory cards	Equivalent circuits for memory cards can be used to emulate the operation of a memory card and its secret authentication feature. The countermeasure was the introduction of challenge-response authentication for memory cards
~ 1992	Interrupting the supply voltage	Writing the retry counter during PIN verification can be prevented by interrupting the supply voltage. The countermeasure was to increment the retry counter before performing the PIN test
~ 1993	Stopping the clock	Conclusions regarding the RAM contents can be drawn by stopping the clock and using an electron-beam tester to analyze the RAM. The countermeasure was the introduction of underfrequency detectors in microcontrollers
~ 1993	Manipulating the microcontroller with a laser cutter	The components on the microcontroller die can be manipulated using a laser cutter. The countermeasure was the introduction of shields on microcontrollers
1995	Timing attack	Due to ignorance, the implementations of many cryptographic algorithms exhibit a dependence between the key value and the processing time. This can be used to help determine the secret key. The countermeasure was the implementation of noise-free cryptographic algorithms

**Table 16.2** (*Continuation*) Summary of typical attacks affecting systems with smart cards

Known since	Attack	Brief description
~ 1995	Bus tapping with microprobes	The buses on the microcontroller die can be tapped using microprobes. The countermeasure was scrambling the buses on the microcontroller die
1996	Differential fault analysis (DFA)	Secret keys for cryptographic algorithms can be determined by selectively introducing scattered computation errors in the processor. The countermeasures were adding glitch detectors on microcontroller dies and using suitable preventive measures in cryptographic algorithms
~ 1996	FIB manipulation of the microcontroller	The components on the microcontroller die can be manipulated using a focused ion beam (FIB). The countermeasure was the introduction of shields on microcontrollers
1997	Exhaustive key search with DES	Using powerful computers or networks of computers, DES keys can be computed within a few hours with a brute-force attack. The countermeasure was the use of triple DES
1997	Statistical distribution of PIN codes	PIN code generation for the German Eurocheque card system did not yield a uniform statistical distribution, with the result that some PIN values were significantly more common than others. The countermeasure was an improved PIN generation algorithm
1998	Power analysis (SPA/DPA)	The data being processed can be determined from the current consumption of the processor. The countermeasures were the introduction of random delays in the processor, using processors with constant current consumption, and a large number of precautionary measures in the microcontroller software
1998	Brute-force attack on the key space of COMP128	Due to a design weakness in the COMP128 authentication algorithm used by some network operators, the secret keys could be determined by a brute-force attack. The countermeasure was using a different authentication algorithm and limiting the number of authentications
1998	Disturbing the processor	By disturbing the processor (such as by using intense flashes of light), it is possible to interfere with its operation at critical points during program execution. The countermeasures were using suitable detectors in microcontrollers along with a large number of precautionary measures in the software

(Continued)

**Table 16.2** (*Continuation*) Summary of typical attacks affecting systems with smart cards

Known since	Attack	Brief description
1999	Brute-force attack on the key space of Yes Card	Due to the small key size of the 321-bit public key used with the French Yes Card (bank card), which dated from 1988, the private key could be calculated. To demonstrate the success of this attack, a functional smart card (not a clone) was produced and its data was signed using the private key that had been discovered. This led to the arrest of the discoverer of this weakness. Counterfeit cards continued to appear from 2001 onward, until the broken key was replaced in 2002
~ 2000	Measuring electromagnetic radiation (SEMA/DEMA)	An adequate data basis for subsequent SPA or DPA analysis can be obtained by measuring the electromagnetic radiation above the chip surface [Agrawal 02]
2001 onward	Software attacks using downloadable program code	The extension of many smart card operating systems to support executable program code made it possible to download manipulated code. The most common countermeasure consists of verification of the code by the smart card before it is executed, and downloading is protected by strong cryptographic means. This prevents unauthorized code downloading by external parties
~ 2002	Processor disturbance using spikes	Processor operation at certain points during program execution can be influenced by using short voltage spikes on the supply line to interfere with its operation
~ 2003	Processor disturbance using high-energy radiation	The operation of the processor can be disturbed by exposing it to high-energy radiation ( $\alpha$ , $\beta$ and $\gamma$ radiation, X-rays, and neutron beams) [Janke 03]. As a countermeasure, correct program execution is ensured by checksums calculated in parallel with program execution
2005	Brute-force attack on the key space of wireless vehicle keys	Due to an inadequate key space, the wireless anti-start systems of certain vehicle models can be bypassed by systematically trying all possible cryptographic keys [Bono 05, Bogdanov 07]
2005	Buffer overflow with RFID (code injection)	Entering executable code in terminals by taking advantage of buffer overflow in wireless tags (RFID tags)
2005	Relay attack with RFID devices	An implementation of the idea of using a wireless link to a distant location to allow payments to be made with contactless smart cards without the knowledge of the cardholders
2005	RFID zapper	A simple circuit can be used to generate an electromagnetic pulse (EMP) that destroys contactless cards (denial of service attack)

**Table 16.2** (*Continuation*) Summary of typical attacks affecting systems with smart cards

Known since	Attack	Brief description
2005	Brute-force attack on the key space of Dutch RFID passports	Due to an unfavorable choice of data in the machine-readable zone (MRZ) of Dutch passports, the Basic Access Control key for contactless readout of the RFID data could be determined by systematically trying all possible keys [Jacobs 05]
2007/2008	Hardware analysis of a secret cryptographic algorithm	The operating principles of the secret cryptographic algorithm Cryptro 1 and the associated random number generation process could be determined by systematic layer removal with Mifare Classic memory chips [Krissler 08]. This analysis yielded a weaker cryptographic algorithm and predictable random numbers, with the result that the cards could be reloaded using replay attacks and could be cloned

attractiveness of an attack is thus correspondingly reduced. Nevertheless, the potential hazard of a successful attack in this phase is significant because there are very extensive possibilities for manipulating the hardware and software.

### 16.3.1 Smart card microcontroller development

The hardware development of a smart card microcontroller takes many months. It is performed by a small group of people working in supervised, controlled-access rooms in the facilities of a semiconductor manufacturer. The computer systems used for semiconductor design are usually part of an independent network that is isolated from the rest of the world. This makes it impossible to alter the chip design from outside, and it prevents outsiders from obtaining information about the internal design of the chip.

Very extensive insider knowledge is needed to undertake manipulations to a chip design that would weaken its security, so this form of attack is probably very unlikely. In addition, nowadays the designs and protection mechanisms of almost all smart card chips are evaluated by independent bodies, so an insider attack would not go undetected.

However, it certainly could be advantageous to an attacker to know the exact design criteria and the arrangement of the functional components on the chip, since the attacker would then know aspects of the chip design such as the protective mechanisms and sensors present on the chip and the scrambling schemes of the buses and memories. This knowledge could later be useful to an attacker in the course of physical chip analysis.

#### Protection: design criteria

There are several basic criteria that apply to the design of the functions of a smart card microcontroller. One of these is that the mechanisms for protection against static and dynamic attacks must be effective. Sensors and other protective components are of little use if they can

be circumvented too easily or if they are ineffective under certain conditions. An example is a sensor on the microcontroller chip that is so large that it can easily be destroyed by a needle, rendering it useless as a protection measure.

A design criterion that differs from the criteria for standard chips and is very important is that absolutely no undocumented mechanisms or functions may be present on the chip ('that's not a bug, that's a feature'). Such undocumented features are usually not fully tested because only a few people know about them, so they often exhibit various errors and weaknesses. As they are not documented, they can be unintentionally overlooked during hardware evaluation and possibly be used later for attacks. The use of such undocumented features is thus strictly prohibited, even though they can often be helpful for developers.

### **Protection: unique chip number**

As part of semiconductor hardware development, all of the hardware security components of the microcontroller must be designed and implemented. In addition to sensors and shields, one of the security components is write-once, read-multiple (WORM) memory, which is also called one-time programmable (OTP) memory. When the semiconductor devices are manufactured, a unique chip number is written to this memory. This makes each chip different, so the chips can be traced individually and the resulting smart cards can later be identified uniquely in the system. In addition, chip numbers can be used for key derivation, and they make it possible to generate blacklists that can be used to take suspect cards out of circulation.

Although these numbers cannot be altered in the original chips, it should be borne in mind that they do not provide any protection against the production of cloned chips using freely programmable microcontrollers. Consequently, security mechanisms cannot be based solely on the presence of a specific chip number in the WORM memory of a particular chip. This unique number can only be used as the basis for true cryptographic security mechanisms. For example, the chip number can be used for the derivation of secret keys, which are in turn used for challenge-response authentication.

### **16.3.2 Smart card operating system development**

Software for smart cards is developed according to modern software development principles. Regardless of which life cycle model is used (waterfall, spiral or whatever), certain general requirements must be observed.

A separate, fully isolated network that does not allow any external access is always necessary for the development computers. The development tools, such as compilers and simulators, are software packages whose proper operation must be verified in dedicated tests. Sometimes two different compilers are used to ensure that the results are correct. Using software whose origins are not fully traceable is fundamentally prohibited, since such software would offer a possible means to manipulate the development tools and thereby modify the programs to be generated.

### **Protection: development principles**

As with hardware development, no undocumented features may be built into the software. For example, it would certainly be possible to convert the laborious blackbox tests commonly used

with smart cards into whitebox tests by incorporating commands that could be used to read out arbitrary regions of memory. However, if one of these commands were inadvertently left in the operating system, it could be used to read out secret keys in real smart cards. To eliminate any possibility of such an attack, generating dump commands is undesirable, even though they can save valuable development time. However, schedule pressure and the constantly increasing complexity of smart card operating systems have led to a relaxation of this principle. To ensure that none of these commands used in the development phase is ever present in real smart cards after they are issued, special tests to verify the absence of such commands are run during smart card production.

An additional principle is that programmers should never work alone on software development. This is anyhow prohibited by software quality assurance considerations, but the four-eyes principle must also be observed for reasons of security against attacks. This is an effective way to increase the difficulty of insider attacks, since at least two developers must agree to cooperate on any attack. In addition, internal source code reviews are conducted regularly to assure the quality of the code and to monitor the development process.

Once software development is completed, the entire source code and its functions are often reviewed by an independent body as part of a software evaluation.<sup>3</sup> The main reason for these time-consuming and costly reviews is to check for software errors, but they also have the effect of making it impossible for a developer to conceal malware (such as a Trojan horse) in the operating system. In practice, Trojan horses can only be found by reviewing the entire program code, since an experienced programmer can certainly find ways and means to conceal a Trojan horse so that it cannot be found by a blackbox test.

### Protection: shared secrets

If several people work on a task, the result will be significantly more resistant to attack due to the diverse opinions and experience of the people involved. The shared secrets principle is the opposite of this ‘everyone knows everything about everything’ approach. In the development of security components, full knowledge of the component should fundamentally never be vested in a single individual, since that person would then be subject to attack. Consequently, development knowledge is divided between several individuals, in the same way as in some military areas, so that there is never anyone who knows everything, although it is still possible for experts to discuss specific topics.

Similar considerations apply to the process of completing the smart card operating system, which is when tables, program code and configuration data are loaded into nonvolatile memory (EEPROM or flash). In addition to providing increased flexibility, this involves a security aspect. With this approach, the chip manufacturer who receives the final, assembled ROM code for producing the fabrication masks does not have full knowledge of the operating system. The portions of the operating system that are located in nonvolatile memory are not known to the chip manufacturer, who thus cannot learn all of the security mechanisms and functions of the operating system by analyzing the ROM code. Flash-based microcontrollers have an advantage here because the smart card operating system does not have to be provided to the semiconductor manufacturer for preparation of a ROM mask.

<sup>3</sup> See also Section 15.5, ‘Evaluation of Hardware and Software’, on page 659

## 16.4 ATTACKS AND DEFENSE MEASURES DURING PRODUCTION

Attacks during the production of chips or smart cards are typically insider attacks because the production environments are closed. Access is strictly controlled, and every entry is logged. Nevertheless, security measures cannot be dispensed with in the production phase, since some technically very interesting and effective attacks can be carried out in this phase.

### Protection: authentication during production

Already at the wafer fabrication stage, smart card microcontrollers are individualized by chip numbers and protected by transport codes. With current operating systems, the transport code is chip-specific, and authentication is mandatory for every access to the chip in the production process. Although this makes chip production more complicated and requires a security module in every production machine, it considerably enhances security.

A straightforward form of attack during production is to feed dummy chips or dummy smart cards into the system. They behave the same as genuine components, but incorporate special features, such as a memory dump command. The earliest opportunity to replace a genuine chip with a dummy chip is after the wafer has been separated into individual dice. This form of attack can be described briefly here by using a smart card for digital signatures as an example. In this scenario, the attacker replaces a genuine smart card with a dummy card during initialization. This card is then initialized with genuine data and afterward personalized.

As this smart card has the full functionality of a genuine smart card, the microcontroller will also generate the key for the asymmetric cryptographic algorithm. It obtains the data needed for this from the initialization and personalization data. After this, the attacker must manage to recover possession of this card, and then the special dump command can be used to read the secret signature key from the card. As the associated public key has been signed by the trust center and thus confirmed as genuine, the attacker knows everything necessary to produce as many duplicate cards as desired, all of which will be recognized as genuine.

This attack is unrealistic if only because organizational measures are taken to prevent chips and smart cards from being taken into or removed from the relevant production facilities. In addition, swapping of chips or cards is prevented by mandatory mutual authentication of the smart card and the security module of the production machine as part of each production operation.<sup>4</sup>

## 16.5 ATTACKS AND DEFENSE MEASURES DURING CARD USAGE

Access to the component under attack – the smart card – is generally much easier for an attacker after the smart card has been issued than in the previous phases of its life cycle. This is one of the reasons why the probability of attack is relatively high in the card usage phase.

The idea of self-destructive smart card microcontrollers appears again and again in many publications as a sort of panacea against all forms of attack. There are hardware security modules, such as those used in military applications, in which such mechanisms are sometimes

<sup>4</sup> See also Section 14.7, ‘Loading Static Data’, on page 609 and Section 14.8, ‘Loading Individual Data’, on page 618 for detailed descriptions of the usual cryptographic processes for the initialization and personalization of smart cards

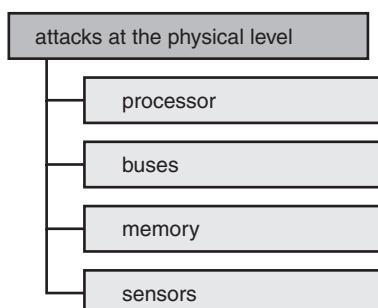
used, but this sort of defense mechanism is not possible in smart cards for a number of reasons. First of all, in the absence of external power a smart card has no way to recognize a potential attack, and secondly, no form of active defense mechanism is possible because the smart card does not have an internal source of operating power. Besides this, for purely legal reasons it is unlikely that any cardholder would find true self-destruction capability acceptable. Who would assume responsibility for any loss or damage that might occur under unfavorable circumstances simply because a smart card mistakenly destroyed itself? In addition, true self-destruction is not at all necessary, since in almost all cases it is sufficient to erase the secret keys stored in the card.

There is yet another aspect to the subject of erasing keys or blocking smart cards. It is very difficult for a smart card to even recognize that it is being attacked. There is simply no sensor available that can generate the message: ‘Attack! Erase everything!’ An excessively low operating voltage or excessively high clock frequency could be a sign of an attack, but these situations also occur in normal operation due to unfavorable ambient conditions. Dirty or corroded contacts have high contact resistance and thus cause the operating voltage to be lower than normal. An excessive clock frequency can be present in a smart card terminal intended to be used with cards that operate at high frequencies. As recognizing an actual attack is so difficult and usually not even possible, automatic mechanisms for blocking the card or erasing the keys are usually not used.

Here we describe and explain several forms of attack that can be regarded as nearly classic. In a certain sense, these descriptions present a picture of the state of the art. They are primarily intended to provide a sound overview for people who are unfamiliar with the subject of smart card security, in order to avoid the use of mechanisms that are already known to be vulnerable. These attacks can be foiled by the defense measures described below, which in turn can be countered by slightly modified attack scenarios. This leads to the well-known cat-and-mouse game of measures and countermeasures for attack and defense.

The scenarios presented here do not form an invitation to compromise the security of smart card systems, since without exception they are both known and published [Kömmerling 99]. They do not represent any serious threat to the security of contemporary smart card systems, since they have long since been dealt with by suitable protection measures. However, a few years ago it would have been possible to achieve a certain amount success with such scenarios.

The attacks are divided into those that are directed against the chip hardware (see Figure 16.8) and those in which an attempt is made to compromise the smart card system at the logical level. The physical attacks and analysis methods can also be divided into static



**Figure 16.8** Classification of the points of attack on a smart card microcontroller at the physical level

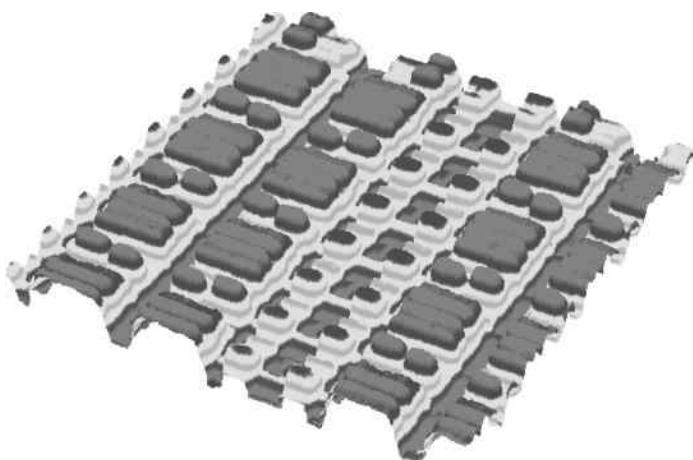
and dynamic types. With a static analysis, the chip is not operating, but it may be electrically powered. With a dynamic analysis, which is much more difficult to perform, the chip operates with its full range of functions during the analysis.

### 16.5.1 Attacks on the hardware

Manipulations at the semiconductor level require considerable technical resources. Depending on the attack scenario, the necessary equipment may include a microscope, a laser cutter, micromanipulators, focused ion beams, chemical etching equipment, and fast computers for analyzing, logging, and evaluating the electrical processes in the chip. This equipment and the knowledge of how to use it are available to only a few specialists and organizations, which strongly reduces the probability of an attack at the physical level. Nevertheless, a card producer or semiconductor manufacturer must always assume that a potential attacker could employ the devices and equipment necessary for such an attack, which means that suitable protection must be built into the hardware.

In order to carry out an attack at the physical level, a few preliminary steps are necessary. The first thing that has to be done is to remove the module from the card, which can easily be done using a sharp knife. After this, the epoxy resin encapsulation must be removed from the chip. Anderson and Kuhne [Anderson 96b] used fuming nitric acid for this purpose, with an infrared lamp as a heat source, followed by an acetone rinse to clean the chip. After this, the semiconductor chip is free and still fully operational. Many people think that the chip now lies unprotected before them and only has to be read out (see Figure 16.9), but this is by no means so. An attacker still has to work through a multitude of security measures before gaining access to the secrets.

The protective components in the hardware can be divided into passive and active components. The passive components are based directly on the technology of semiconductor



**Figure 16.9** Graphic representation of the surface profile of a smart card microcontroller, as measured using an atomic force microscope. The maximum surface relief in this illustration is only  $2.3\text{ }\mu\text{m}$  (Reproduced with permission from Giesecke & Devrient)



**Figure 16.10** Making contact with an exposed smart card microcontroller using needle probes under an optical microscope (Reproduced with permission from Giesecke & Devrient)

manufacturing. They include all processes and methods that can be used to protect the memory regions and the other functional components of the microcontroller against various types of analysis.

In addition to these passive measures based on semiconductor manufacturing processes, there is a broad spectrum of active components that can be used on silicon chips. Active protection means the integration of various types of sensors on the silicon die. These sensors are queried and evaluated by the smart card software as needed. This is naturally only possible when the chip is fully powered and operational. A chip without electrical power cannot measure any sensor signals, let alone evaluate them. Sometimes the boundary between useful protective components and technical gadgetry is particularly narrow where sensors are concerned.

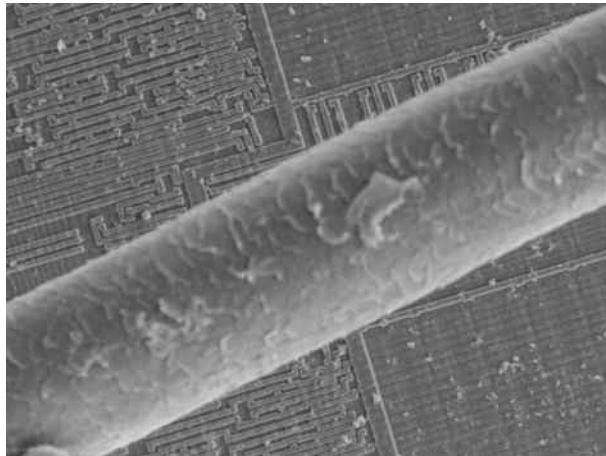
A light sensor that is supposed to prevent optical analysis of the memory will not respond if the chip is located on the object carrier of an optical microscope without power or a clock signal (see Figure 16.10). In addition, it is very easy to visually identify such a sensor on the chip surface and cover it with a drop of black ink, so its protective function can easily be neutralized, even when the chip is operating. However, this can be countered by distributing a large number of light sensors over the entire chip area.

Long-term functional reliability is also an important consideration. For example, a temperature sensor that causes the smart card software to erase the entire nonvolatile memory in response to brief but harmless overheating of the chip makes absolutely no contribution to increased functional reliability or security against attacks. Consequently, most smart card microcontrollers employ only a few sensors.

Here we describe the smart card microcontroller protective mechanisms that are the most important and most often used in practice.

### Protection: semiconductor technology

The dimensions of the structures on the chip (track widths, transistor sizes, etc.) approach the limit of what is currently technically feasible. The usual structure widths are in the range



**Figure 16.11** Photograph of a human hair in comparison with the semiconductor structures of a smart card microcontroller at 1000 $\times$  magnification (Reproduced with permission from Giesecke & Devrient)

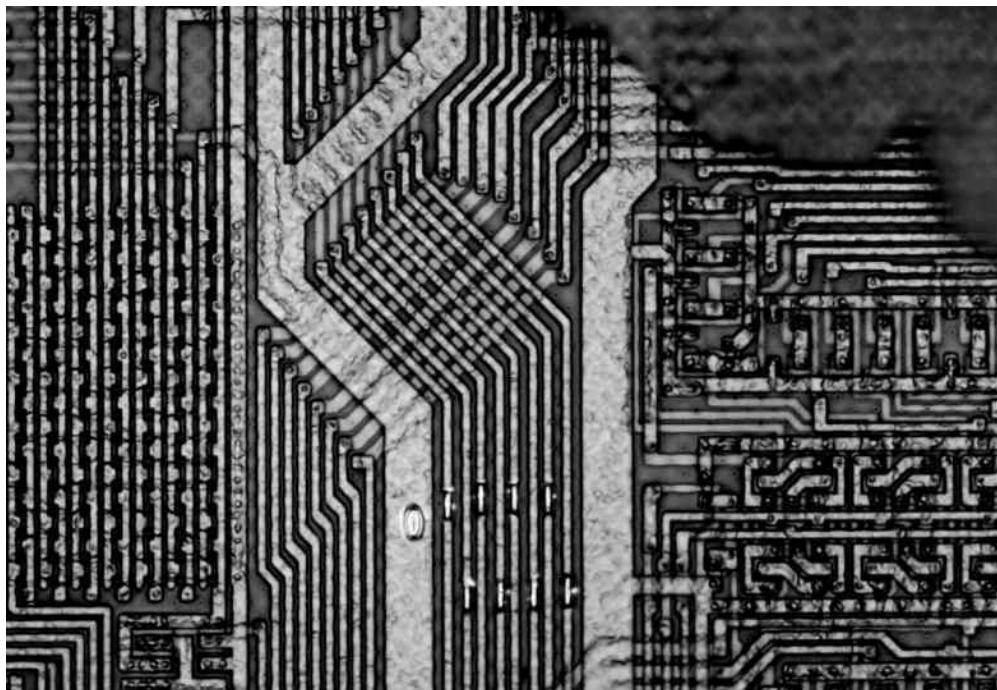
of 0.15  $\mu\text{m}$  to 90 nm (see Figure 16.11), which in itself is no longer technically remarkable. However, the transistor density on the silicon belongs to the highest level that can currently be achieved using standard lithographic fabrication methods. These very fine structures alone make it nearly impossible to extract any information from the chip using analytic methods, for which reason semiconductor technologies with structure sizes of around of 1  $\mu\text{m}$  are presently still regarded as secure. This is bound to be reduced in the future.

### Protection: chip design

Standard cells, which may contain components such as a processor core or a particular type of memory, are often used in designing semiconductor devices. The advantage of using standard cells is that it allows a semiconductor manufacturer to quickly produce a variety of different types of chips with a high level of quality. This technique, which has been developed for mass-produced components where security is not an issue, cannot be used with smart card microcontrollers. This is because the designs and functions of standard cells are known, and their use would provide a potential attacker with too much information and thus considerably simplify the attacker's task. The functional components of smart card microcontrollers are developed specifically for this application and are not used for any other purpose.

### Protection: dummy structures

Dummy structures on the chip are a defense measure that often gives rise to controversy among experts. Dummy structures are elements of the semiconductor device that do not have any actual function, but instead are intended to confuse and mislead an attacker. The associated security is based entirely on keeping the existence and locations of such structures secret. Dummy structures can also be monitored, so that any changes to them can be detected and cause the chip to switch off. The main disadvantage of dummy structures is the additional space that they occupy on the chip.



**Figure 16.12** Exposed buses on a microcontroller chip. The passivation layer has been drilled through in the middle with a laser cutter to allow the underlying layer to be contacted with needle probes. Needle probes can be used to measure the signals on the tracks, which is called probing. It is also possible to use needle probes to inject signals in the bus tracks or override the existing signals. This is called forcing (Reproduced with permission from Flylogic)

### Protection: chip buses

All internal buses of the chip (see Figure 16.12), which connect the processor to the three different types of memory (ROM, RAM, and nonvolatile memory), are not brought out from the chip, so it is not possible to make direct connections to these buses. It is thus not possible for an attacker to tap the address, data or control bus of the microcontroller or control the bus signals in order to read out the memory contents. It is entirely conceivable that if contact can be made with a bus, forcing signals can be used to alter the data transported by the bus. The buses are usually fabricated in the lower layers of the semiconductor device in order to make it difficult to make direct contact with them from the surface. In addition, the buses on the chip are scrambled in a static, chip-specific, or session-specific manner, so the functions of the individual bus lines cannot be recognized from the outside. There are even smart card microcontrollers whose bus scrambling is continuously modified during a session.

### Protection: memory design

With ROM-based smart card microcontrollers, most program routines are usually stored in ROM. The contents of ROMs commonly used in the industry can be read bit by bit using an optical microscope. It would not be particularly difficult to arrange these bits in bytes and then

arrange the bytes to obtain the complete ROM code. To prevent exactly this type of analysis, the ROM is located in one of the lower layers of the chip instead of the top layer, which is the most easily accessible. This impedes optical analysis.

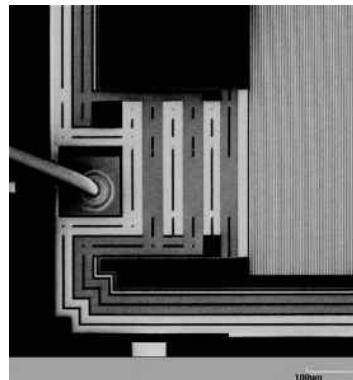
However, if the chip were glued face down on a carrier and the rear surface were ground off, it would be possible to read the contents of the ROM. For this reason, only ion-implanted ROM is used in smart card microcontrollers, since the contents of this type of ROM cannot be seen using either visible or ultraviolet light. This also largely protects against selective etching, which is a process that attempts to etch the semiconductor such that the contents of the ROM are made visible.

### Protection: shields

A potential form of attack consists of analyzing the electrical potentials on the surface of the chip while it is operating. With suitably high scanning resolution, this technique can be used to measure charges (voltages) in very small regions of the die. With this information, it is possible to draw conclusions about the contents of the RAM while the chip is operating. This analysis can be prevented very effectively by placing current-carrying metalization layers on top of the memory region or the entire chip (see Figures 16.13 and 16.14). If these metalization layers are removed by chemical etching, the chip will no longer operate properly because they serve as conductors for electrical power during operation. In many cases, several shields are arranged on top of each other and continuously monitored for integrity. Here a distinction can be made between supply shields and sensing shields. A supply shield distributes power to the



**Figure 16.13** An SLE66CX322P smart card microcontroller with and without its active shield. The chip with the active shield is shown on the left. Only the bonding pads can be recognized with an optical microscope; the rest of the chip is covered by the shield for protection against attacks. The functional units below the shield are shown on the right. The EEPROM is at the top, the ROM is at the bottom right, and the dark square on the left-hand side is the CPU. The bright structures in the picture are functional units such as RAM, a random number generator, and sensors, which are protected by separate shields (Infineon)



**Figure 16.14** Detail of a smart card microcontroller whose entire surface is protected by an active shield. Only the pads for the bonding wires are left exposed (Infineon)

components on the chip, while a sensing shield is used to detect physical attacks. The spacing of the serpentine tracks of the shields is very narrow (around 1 µm center to center).<sup>5</sup>

Meandering conductor structures can also be fabricated on top of the entire chip area or especially vulnerable areas, such as an underfrequency detector. These structures can easily be monitored by measuring their resistance or capacitance, or they can be incorporated in the circuitry of the chip so that it immediately stops working if they are damaged. Security can be further increased by altering the connections of these meander structures during a session. This provides protection against using a focused ion beam (FIB) tool to bridge the meanders.

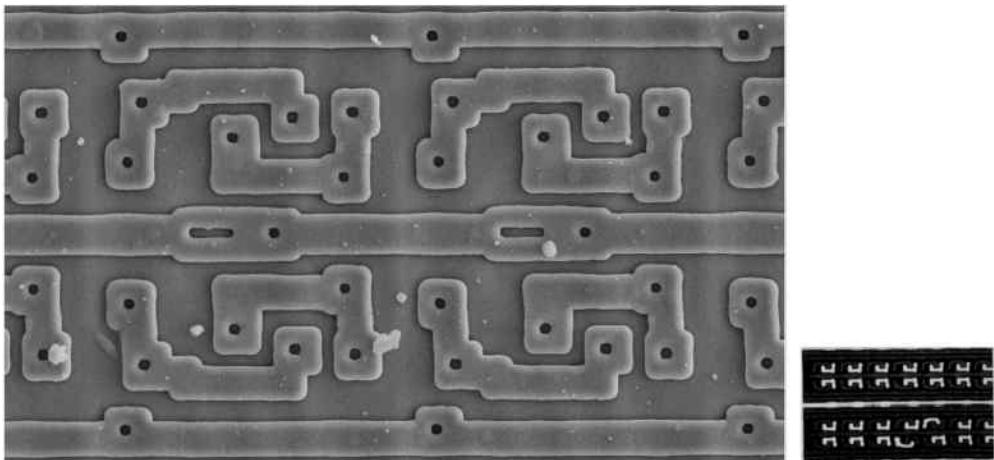
It is also conceivable to use opaque shields whose integrity is continuously monitored by phototransistors, which are easily implemented in semiconductor devices. Removal of the shield can be detected immediately, with the result that the chip stops working.

### Attack and defense: reading out the volatile memory

It is commonly known that RAM loses its data contents in the absence of power. However, this does not occur if the memory cells are cooled to below  $-60^{\circ}\text{C}$ . Also, the RAM contents are not necessarily lost when power is removed if the stored data remains unchanged for a long time. The reason for this effect is described in a paper by Peter Gutmann on the subject of securely erasing memory media [Gutmann 96]. For this reason, secret keys are not held in RAM any longer than is absolutely necessary, after which they are immediately erased or replaced by other data. This minimizes the risk that traces of secret keys may be left in the RAM cells, and it impedes attacks based on fixing the RAM contents by freezing or burning.

Reading out RAM cells is very difficult because it requires detecting the switching states of the transistors forming the memory cells. However, it is certainly possible to extract stored data from RAM cells using sophisticated electron microscopes and special contrast-enhancement methods (see Figure 16.15). A prerequisite for this is removing the passivation layer and the underlying metallization layers, which protect the RAM against exactly this form of attack. Removing the metallization layers unavoidably destroys the RAM cells, since these layers form an integral part of the RAM circuitry.

<sup>5</sup> A good overview of physical attacks can be found in an article by Marcus Janke and Peter Laackmann [Janke 02]



**Figure 16.15** Photomicrograph of several RAM cells without protective metallization layers, magnified by a factor of 3000. The photo on the right shows the electrical potentials of the same RAM cells as measured with an electron beam tester during operation. The distribution of zeros and ones in the RAM is clearly visible (Reproduced with permission from Giesecke & Devrient)

linearly increasing memory addresses										scrambled memory addresses									
00	01	02	03	04	05	06	07	08	09	06	01	19	03	04	05	40	07	10	09
10	11	12	13	14	15	16	17	18	19	15	11	12	13	14	18	16			
20	21	22								20	21	22	17	00	02				
30										30									
40										08									

**Figure 16.16** Comparison of a conventional semiconductor memory and a scrambled memory in a smart card microcontroller

### Protection: memory scrambling

Scrambling the memory on the microcontroller chip, which is similar in principle to the established practice of bus scrambling, is being used increasingly often. The security of this technique is based on the secrecy of the scrambling scheme for the memory cells (see Figure 16.16). Memory scrambling is easily implemented and does not require much additional space on the chip. If the scrambling scheme is not known, it is extremely difficult for an attacker to discover how the memory cells are actually addressed.

Nonvolatile memory can also be scrambled using software, although this increases program overhead and requires all write accesses to be performed as atomic operations, as otherwise the system would be very vulnerable to sudden power interruptions. However, software memory scrambling has the advantage that it can be made chip-specific and even dynamic, with the memory contents being rearranged during the course of a session.

### Protection: memory encryption

In addition to memory scrambling, batch-specific or chip-specific encryption of the memory and some of the processor registers is available with modern smart card microcontrollers. The corresponding data is decrypted when it is read and encrypted when it is written, all in real time. In addition to key encryption, with some types of chips the memory addresses can be included in the encryption/decryption process so that identical data has different values in different memory regions after encryption. Session keys can also be used, especially for RAM regions.

With this form of encryption, even if data can be read from memory as a result of a successful attack, the plaintext cannot be recovered without knowledge of the secret key. This makes things considerably more difficult for an attacker, who must either know where the relevant key is stored or systematically read all the data stored in the chip.

### Attack and defense: storing secrets in encrypted form

There is actually little point to storing secrets in a smart card in encrypted form, since one of the main advantages of a smart card is that it prevents unauthorized access to secret data. However, it is certainly possible to imagine situations in which the encryption of secret data would increase the security of a smart card, as described below. For this reason, in some cases all the keys stored in a smart card are encrypted using a card-specific key.

If an attack on the smart card operating system were successful, so that internal program processes could be altered, it is entirely conceivable that data held in memory could be revealed as a result. An almost classic example of this is manipulation of the data transmission routine so that it sends more than just the data held in the transmit buffer to the terminal.<sup>6</sup> A similar situation could occur due to an error in the operating system. In such situations, an attacker cannot obtain any advantage from the revealed data if the secrets in the card are stored in encrypted form, since the attacker does not know the secret key used for this purpose. This key is specially protected and stored in another location, or even distributed in the memory. With this protection mechanism, even data obtained from the memory by bypassing the operating system routines cannot be put to any use. All the secrets can be encrypted using a single card-specific key, which naturally never leaves the card, or different keys can be used for the different secrets in the card. The specific implementation depends mainly on how much effort one wants to invest in this protection measure.

However, with this method care must be taken to ensure that the length of the protected secrets is not too short and that an adequate encryption algorithm is used, as otherwise the method will be ineffective. This can be illustrated by the following example. A PIN code could be encrypted using a one-way function, with the result being stored in nonvolatile memory for use as the reference value in subsequent PIN comparisons. A card-specific key could be used for the one-way function, so that the reference data for two identical PINs would be different in two different cards.

At first glance, it would appear that it would now be impossible to derive the PIN code from the reference value if the reference value could be read from the nonvolatile memory. However, a clever attacker would not take this approach. If normal four-digit PIN codes are used, the number space of the PINs has a lower limit of 0000 and an upper limit of 9999. This means that the number of possible PINs is exactly 10 000. An attacker who can read the encrypted PIN code from the smart card memory and additionally knows the one-way function

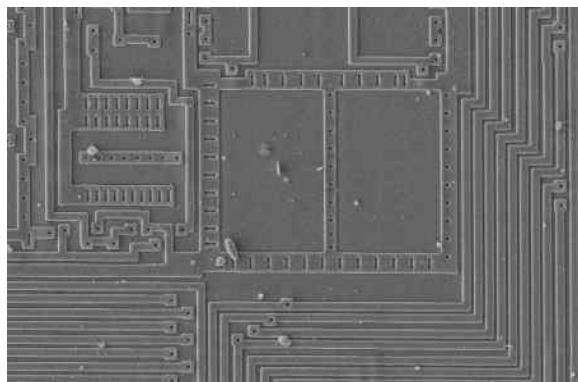
<sup>6</sup> See also Figure 16.36 on page 710

and associated card-specific key used for the encryption will proceed as follows. The attacker uses the available information to perform one-way encryption of all possible four-digit PIN codes (10 000 values) in succession. After an average of 5000 attempts, the attacker will have obtained a result that matches the reference value in the smart card, and with this the attacker knows the PIN code.

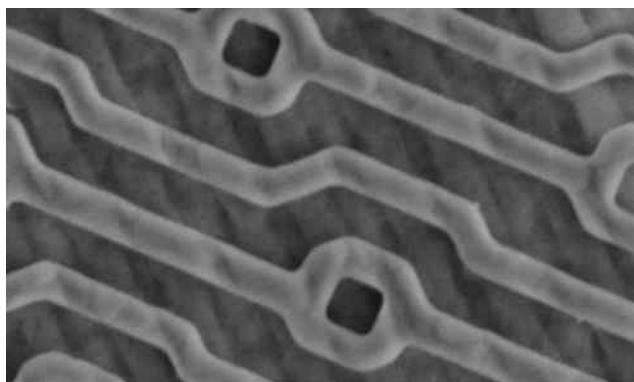
As can be seen, in this case storing the PIN code in encrypted form does not provide any significant benefit. It would be distinctly better to encrypt the entire file, including all the keys and the PIN code. The resulting number space would be much too large for searching using this form of attack, so the attacker would not be able to determine the PIN code in this way.

### Protection: passivation monitoring

At the conclusion of the fabrication process, a passivation layer is placed on top of the microcontroller on the silicon. The passivation layer (see Figures 16.17 and 16.18) prevents oxidation



**Figure 16.17** Photomicrograph of a passivation monitoring arrangement at 1000 $\times$  magnification. The passivation sensor essentially consists of the two rectangular semiconductor structures and operates on the principle of capacitance measurement (Reproduced with permission from Giesecke & Devrient)



**Figure 16.18** Photomicrograph of a modern passivation arrangement at 8000 $\times$  magnification. The track spacing is 4  $\mu\text{m}$ , and the passivation sensor operates on the principle of resistance measurement (Reproduced with permission from Giesecke & Devrient)

due to atmospheric oxygen and damage to the chip surface by other chemical processes. The passivation layer must always be removed before any sort of manipulation of the chip can be performed. It should be borne in mind that although the passivation layer can be removed using chemical processes, the exposed chip is then very vulnerable to oxidation, which can destroy it relatively quickly. A sensor circuit can use resistance or capacitance measurement to determine whether the passivation layer is still present. If it is missing or damaged, this can trigger an interrupt to the chip software or cause the complete hardware of the chip to be shut down, which reliably prevents any sort of dynamic analysis.

### **Protection: voltage monitoring**

Every smart card microcontroller has a voltage monitor, which ensures an orderly shutdown of the device if the upper or lower supply voltage limit is exceeded. This gives the software the assurance that the chip will not operate in marginal regions where it may not function properly. Without this voltage monitor, the program counter of the processor might become unstable if the chip is operating in a marginal region, which could result in uncontrolled program execution. This faulty behavior could be used as a starting point for determining secret keys by using differential fault analysis (DFA), which is described below.

For this reason, it is important for the voltage monitor also to be able to detect very brief voltage peaks or dropouts in order to protect against typical attacks based on the intentional inducement of scattered processor errors. As an example, the usual shutdown limits are 2.3 and 6.3 V for smart cards with a specified supply voltage range of 3–5 V. These values lie slightly outside the voltage range of 2.7–5.5 V specified by various standards, in order to allow for tolerances in sensor calibration during semiconductor fabrication.

Voltage monitoring in particular is very important for the security of the microcontroller. A conceivable form of attack would be first to use a focused ion beam (FIB) or similar tool to disable the relevant sensors and then to start the actual attack. For this reason, the components that are vital to the security of the microcontroller are often specially protected so that manipulation can be detected, causing the smart card to shut down automatically.

Another type of sensor that is partly based on voltage monitoring is the power-on detector. This power-on detection, which is present in all chips and is independent of the external reset signal, ensures that the chip is always put in a defined initial state when power is first applied. Like voltage monitoring, this is necessary to ensure correct operation.

### **Protection: frequency monitoring**

Smart cards usually operate with external clock signals, so their internal processing speed is determined externally. This means that, at least in theory, it is possible to run the microcontroller in single-step mode. This would provide outstanding opportunities for analysis, in particular measuring the current consumption (power analysis) or the electrical potential on the surface of the chip. To prevent such attacks, a functional component for detecting underfrequency and overfrequency conditions is integrated in the chip. This eliminates the possibility of reducing the clock rate to unallowable levels.

Most specifications state a minimum clock frequency of 1 MHz. For technical reasons, underfrequency detection has a large tolerance range, so the chip usually stops working at around 500 kHz. This ensures that the chip will always operate at the minimum specified clock frequency of 1 MHz. The specified upper frequency limit is usually 5 MHz, and typical overfrequency detectors disable the chip at approximately 7 MHz. Modern microcontroller hardware is often built such that the chip will not operate if the clock frequency is too high.

To provide reliable protection against the hazards of single-step mode, it is naturally necessary to protect the underfrequency detector with shields so that any attempt to tamper with the detector will be recognized.

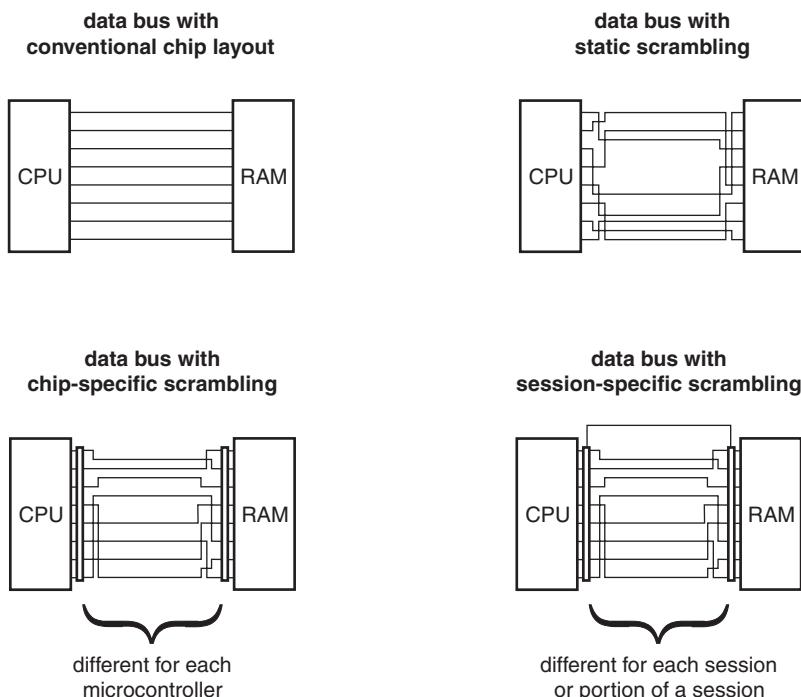
A small but interesting detail can be mentioned here in passing. Smart cards in the telecommunication sector allow the clock to be stopped in order to reduce power consumption in mobile telephones. Naturally, this must not lead to triggering of the underfrequency detector.

### Protection: temperature monitoring

Some types of chips incorporate a temperature sensor, but the benefits of this are debatable. A brief temperature excursion outside the specified operating range will not cause any permanent damage to the chip and does not constitute an attack. Shutting down the chip in such marginal situations could cause an increased failure rate in use without providing the operator of the smart card system with any additional security.

### Protection: bus scrambling

Many smart card controllers have scrambled internal memory buses. This means that the individual bus lines are not arranged next to each other in increasing or decreasing order, but are instead arranged randomly and swapped several times, or even arranged on top of each other in several layers (see Figure 16.19). This represents an additional hurdle for a



**Figure 16.19** Bus scrambling in a smart card microcontroller, here with an 8-bit data bus between the CPU and the RAM. The data bus lines shown here represent information flows rather than electrical leads. The encryption units are shown as separate components here for the sake of clarity, but they are actually interleaved with the rest of the components in such a manner that they cannot be recognized as separate components, thus making them immune to attack

potential attacker, who does not know which bus line is associated with which address bit or control function.

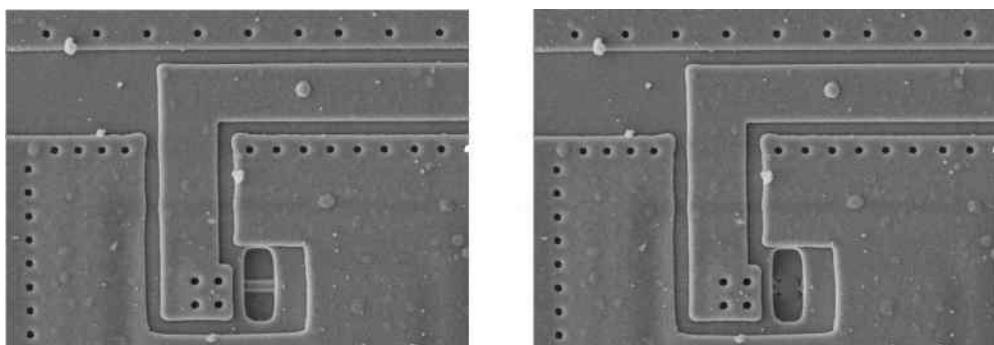
Bus scrambling was originally performed only in static form, with the same scrambling scheme on every chip. With static scrambling, it would probably not be especially difficult for an attacker to discover the scrambling scheme within a reasonable length of time and thus take it into account when tapping the bus. The security provided by this technique can be improved by using chip-specific scrambling. This is not achieved by using different exposure masks for the buses of individual chips, since this is currently not technically feasible or affordable. Instead, scrambling is performed by randomizer circuits ahead of the memory, which can be driven by the chip serial number or another unique feature. This does not impose significant additional fabrication effort, and it makes life considerably more difficult for anyone who wants to tap the bus. A combination of chip-specific and session-specific scrambling can also be obtained by using different initial values for the scrambler.

### Protection: irreversible switching from test mode to user mode

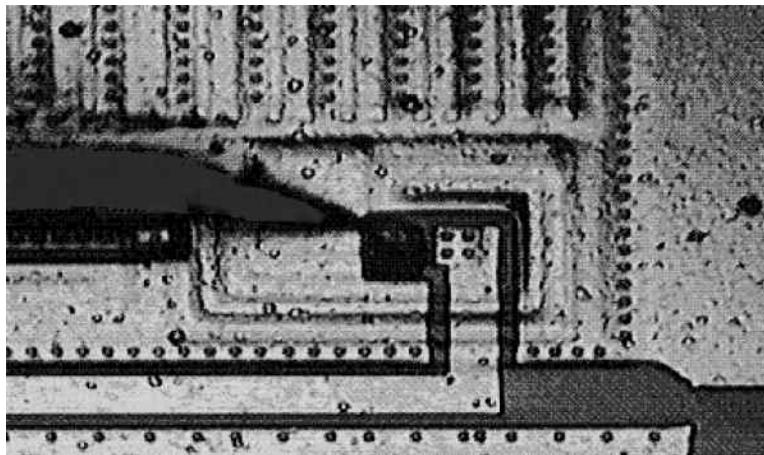
All microcontrollers have a test mode that is used to test the chips during fabrication and run internal test programs while the devices are still on the wafer or after they have been packaged in modules. The test mode allows types of access to the memory that are strictly forbidden later on when the chips are in normal use. However, for production reasons it is essential to be able to read data from the nonvolatile memory in this mode.

Switching from test mode to user mode must be irreversible. This can be achieved with a polysilicon fuse on the chip (see Figure 16.20). In practice, the fuse is blown by applying a voltage to a test point on the chip provided for this purpose. This results in hardware switching of the chip to user mode. This is normally irreversible. However, a fuse is by its nature a relatively large structure on the surface of the chip. It is conceivable that the fuse could be bridged mechanically after the overlying portion of the passivation layer is removed (see Figure 16.21). This would put the microcontroller back into test mode, and the memory could be read out using the extended access capabilities available in this mode. If the complete memory contents are known, it is easy to produce clones of the smart card.

To defend against this form of attack, most semiconductor manufacturers have adopted the practice of reserving a portion of the nonvolatile memory for the switchover mechanism, in addition to using a fuse. If a certain unalterable value is located in this memory region, the



**Figure 16.20** Photomicrograph of a polysilicon fuse at 2000 $\times$  magnification. The fuse in the picture on the left is still intact, while the fuse in the picture on the right has been blown (Reproduced with permission from Giesecke & Devrient)



**Figure 16.21** Photomicrograph of a polysilicon fuse and microprobe needle at 500× magnification. A blown fuse could be bridged using a microprobe needle (Reproduced with permission from Giesecke & Devrient)

chip has been irreversibly switched to the user mode. Even if the fuse is bridged over, the chip will not return to the test mode because the additional logical switch in the nonvolatile memory prevents this.

The security of the switchover from test mode to user mode can be increased even further by a very simple structural measure in the layout of the microcontroller on the wafer. If the test pads used to make contact with the chip for running the tests are sown off when the wafer is separated into dice, neither a fuse nor any EEPROM cells are needed to switch between the two modes, since the features needed for the test mode will no longer be present.

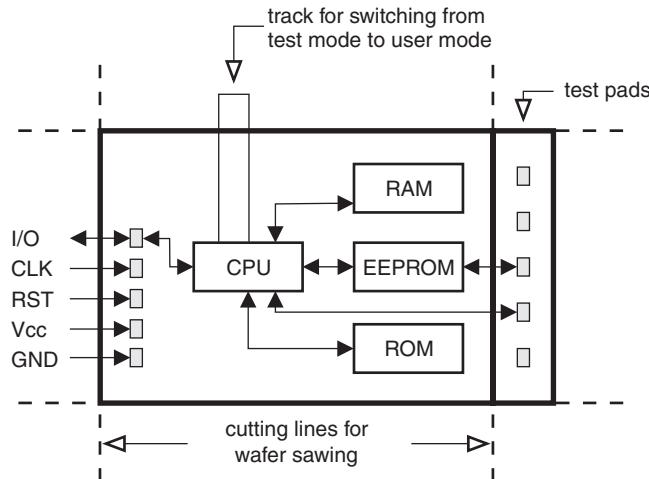
It is also possible to replace the fuse for switching from test mode to user mode by a track that is irreversibly broken when the dice are sown from the wafer (see Figure 16.22). With current technology, it is not possible to make a connection to a sown-through track on the edge of a chip.

#### Dynamic analysis and defense: tapping the microcontroller memory buses

Before the microcontroller buses between the CPU and the memories (ROM, RAM, EEPROM and flash) can be tapped, the chip must be exposed and the passivation layer on the surface of the chip must be removed. The passivation layer protects the chip against oxidation, and it also protects the chip against attack because its integrity is monitored by sensors. According to Anderson and Kuhn [Anderson 96b], it can be removed by etching with hydrofluoric acid. A laser cutter<sup>7</sup> is also a suitable tool for selectively drilling holes in the passivation layer at the necessary locations, as illustrated in Figure 16.12 on page 687.

After the passivation layer is removed from the entire surface of the chip or selected areas, it is at least theoretically possible to make contact with the memory address, data and control buses by using microprobe needles. If it is possible to make electrical connections to all lines of these three buses, it is very easy to address the individual memory cells and read any desired

<sup>7</sup> A laser cutter is a device used for drilling and cutting with a high-power laser beam at a resolution of a fraction of a micrometer



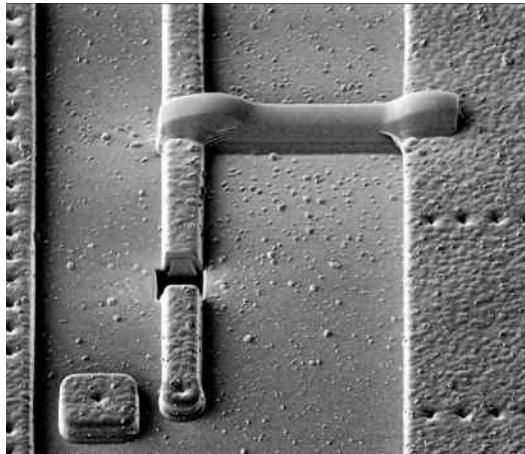
**Figure 16.22** Depiction of one of several options for irreversible separation of the test pads used for CPU and memory testing of a smart card microcontroller and a possible arrangement for automatic switching from test mode to user mode

regions of the ROM and EEPROM or flash memory. The chip does not have to be powered for this, and any desired type of connection device can be used. The consequences of a successful attack using this method would be grave, since in principle it allows all of the secret data in the nonvolatile memory to be read.

This method could be extended by making connections to the buses and then operating the chip in the normal manner. In this way, it would be possible to tap all of the data traffic between the CPU and the memories and record it if a sufficiently fast logic analyzer is available.

As already mentioned, it is very difficult to make electrical connections to individual tracks on the chip. With an eight-bit microcontroller, the number of connections needed for this attack is sixteen for the address bus, eight for the data bus, and one to four for the control bus. In total, at least 25 simultaneous connections would have to be established between an external analysis computer and the tracks on the chip. Even with modern micromanipulator technology, this is presently not possible due to the very small dimensions of the semiconductor structures. It would be possible to use a focused ion beam (FIB) tool, which is commonly used in the semiconductor industry, to implant a sort of contact for each bus line (see Figure 16.23). These areas then could be used as contact points for microprobe needles. However, the effort required for this is enormous. Even if an attacker managed to do this, it would still be necessary to determine the bus scrambling scheme in order to read the data, since the bus lines are not arranged on the chip in an orderly fashion next to each other, but instead in a manner that is not easily recognizable.

If in a few years dramatically improved technology makes it possible to make connections to the buses of current microcontrollers, this will probably not have any effect on security because by that time the semiconductor structures will have become significantly finer than they now are. In addition, in the long term micromechanical technology will probably lag behind semiconductor technology, which is based on optical methods. This means that even in the future, this form of attack will probably not pose a significant threat to the security of smart cards.



**Figure 16.23** An example of using a focused ion beam (FIB) on a semiconductor chip. The track on the surface of the chip running from the top to the bottom of the picture has been broken using an FIB and then connected to a parallel track by a new metallization structure deposited by the FIB, which can be seen in the upper part of the picture (Reproduced with permission from Fraunhofer Institute IC, CTG)

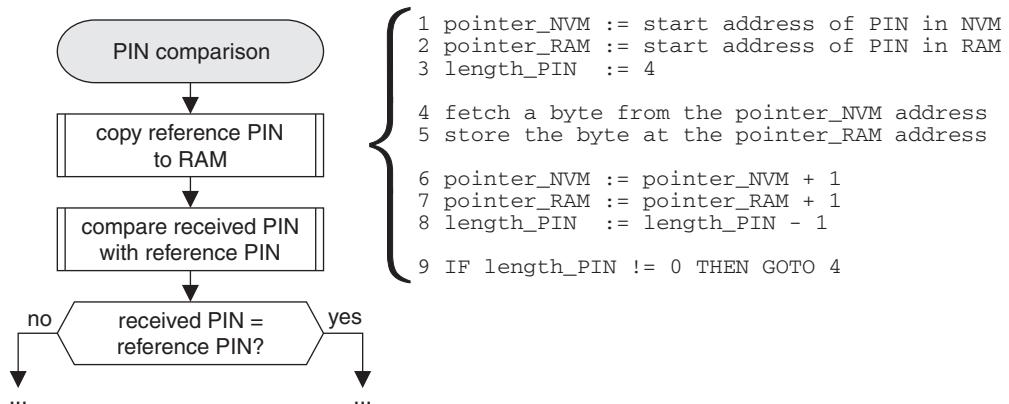
### Dynamic analysis and defense: measuring CPU current consumption

In the first edition of this book in 1995, the following statement appeared at this point: ‘The processor design is also crucial with regard to security. A smart card processor must have nearly the same current consumption for all machine instructions. Otherwise, the current consumption can be used to deduce the instruction being processed. A certain amount of secret information can be derived from these deductions.’ The fact that is possible to deduce the instructions being executed by a processor, and even the data being processed, by analyzing the current consumption of the processor while it is operating was thus already known for several years when Paul Kocher, Joshua Jaffe and Benjamin Jun published a paper on simple power analysis, (SPA) and differential power analysis (DPA) in June 1998 [Kocher 98a].<sup>8</sup>

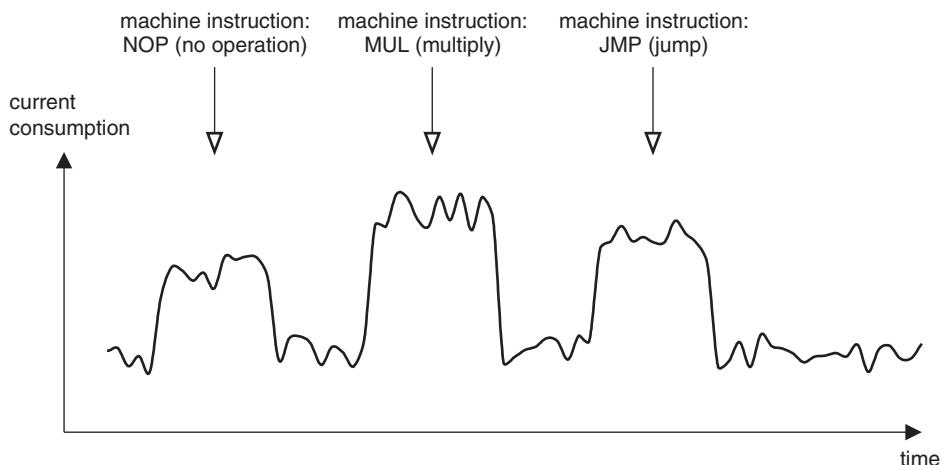
The operating principle of simple power analysis (SPA) is relatively straightforward. The current consumption of the microcontroller is determined by using an analog-to-digital converter to make high-resolution measurements of the voltage drop across a resistor connected in series with the power supply (see Figure 16.28). With a high-performance processor such as a Pentium CPU, it would not be possible to deduce the instructions being executed, due to the complexity of the internal processes. However, the relatively simple architectures of 8051 and ARM CPUs used in smart card microcontrollers result in measurable and interpretable variations in current consumption that depend on the instructions and data being processed. To illustrate this, imagine that a particular program sequence with a particular set of data always produces the same curve of processor current versus time. If the same program is then run using different data, the curve of current versus time will be different. This difference can be used to determine the data being processed by the program.

A simple attack scenario using SPA could take the following form. To start with, the attacker needs a smart card whose PIN code is known and can be changed to any desired value. Using this smart card together with suitable measuring equipment (see Figure 16.29), the

<sup>8</sup> Detailed overviews of this topic can be found in [Kocher 98b] and [Messerges 99]

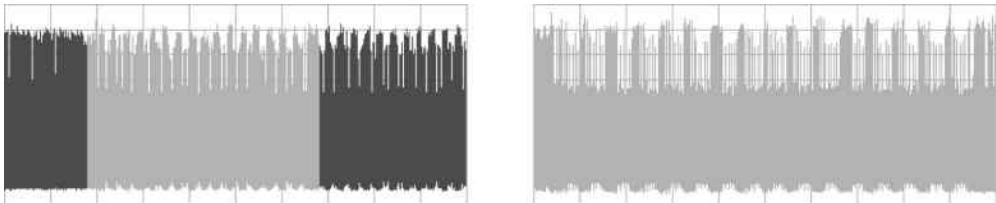


**Figure 16.24** Simplified operating principle of determining an unknown four-byte PIN code by comparing measurements of current consumption during the copy routine with a known and an unknown smart card. Depending on the programming of the comparison routine, the measurements can also be made at a different time. In this example, the current is measured while the reference PIN is being copied to RAM because the current consumption of the processor while executing this routine depends only on the data being processed (the PIN)

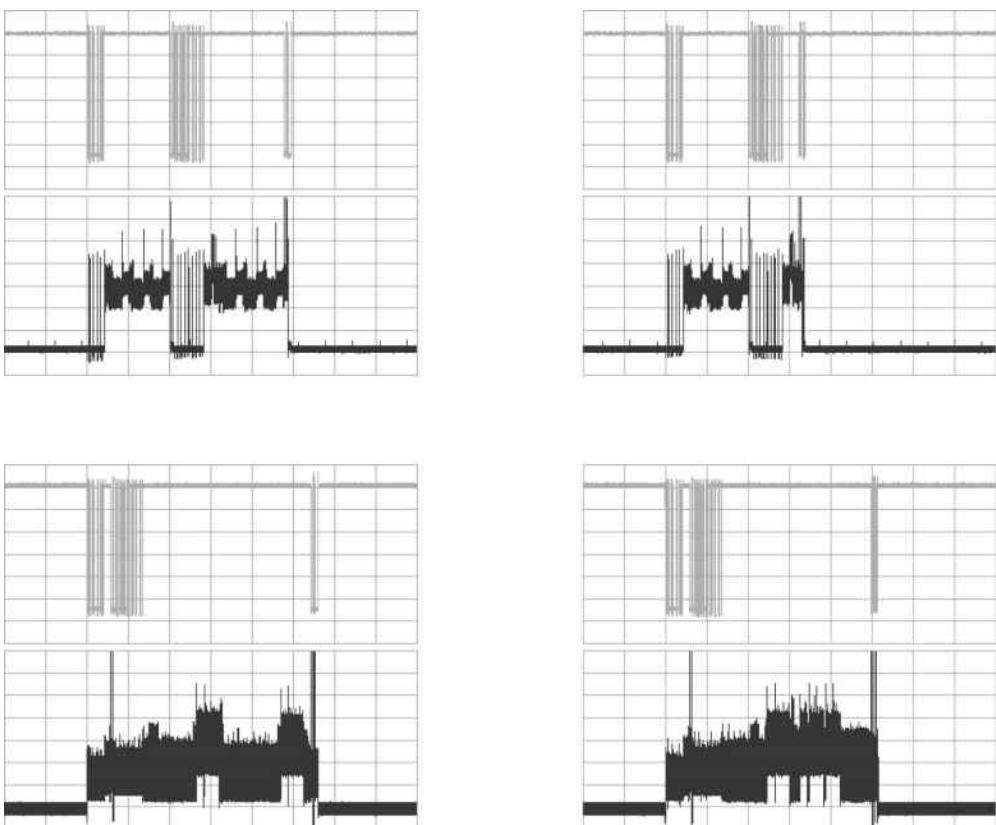


**Figure 16.25** Simplified depiction of the variation in the current consumption of a smart card microcontroller while processing various machine instructions. Besides being dependent on the machine instruction being processed, the current consumption may depend on the data being processed

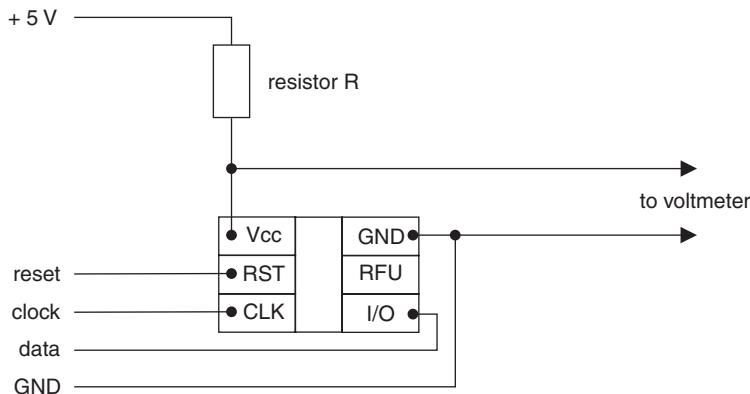
attacker records reference current patterns for all possible PIN codes. These patterns depend on the microcontroller, the operating system, and the individual PIN code. The measurement period includes the time interval when the reference PIN is copied from nonvolatile memory (EEPROM or flash) to RAM. This is followed by the comparison with the externally supplied PIN code, as illustrated in Figures 16.24 and 16.25. After this, similar comparative measurements are made using an arbitrarily chosen PIN with a second smart card whose PIN code is not known. The resulting current pattern is compared with the reference patterns, which allows the unknown PIN code to be determined, as illustrated in Figures 16.26 and 16.27.



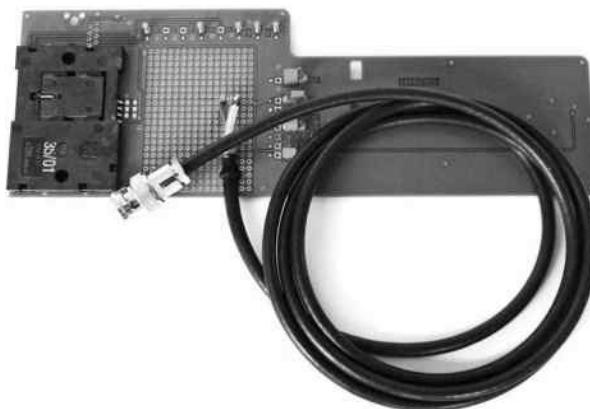
**Figure 16.26** Current consumption curves during triple-DES computation by a smart card microcontroller without SPA/DPA countermeasures. The curve on the right is an enlarged section of the curve on the left (Reproduced with permission from Giesecke & Devrient)



**Figure 16.27** Plots of current versus time during PIN comparison by a smart card microcontroller. The upper curve of each set shows the serial communication process with the  $T = 0$  protocol, while the lower curve shows the current consumption of the microcontroller. The set of curves at the top left shows the current consumption during PIN comparison with the correct PIN, while the set at the upper left shows the same process with an incorrect PIN. The software used with the upper sets of curves is not hardened against this type of analysis. By contrast, the software of the microcontroller used for the lower sets of curves has suitable defense measures. The curves at the lower left show the current consumption during PIN comparison with the correct PIN, while the curves at the lower right show the current consumption with an incorrect PIN (Reproduced with permission from Giesecke & Devrient)



**Figure 16.28** Wiring diagram of a simple arrangement for using a series resistor to measure the current consumption of a smart card microcontroller



**Figure 16.29** An adapter (commonly called a paddle) used to connect a smart card to a terminal with the card available for measurement purposes. The smart card contact unit is visible on the left, a prototyping area for electronic circuitry is located in the middle, and the eight contacts for the terminal are on the right. The paddle also has a sense resistor in series with the power lead for making current measurements for SPA/DPA analysis. The signal across the sense resistor is fed to an instrumentation amplifier via the attached coaxial cable

A variety of measures can be used to defend against this attack. A relatively simple measure is to insert a random delay, such as by using random wait states, before the start of the entire PIN routine. This makes it considerably more difficult to synchronize the current measurements with the copying process, since it no longer occurs at a fixed time. Another measure is to XOR the PIN code with a card-specific random number and to store the result in nonvolatile memory as the reference value. This makes the PIN code card-specific, regardless of its actual value, so a comparison of current consumption during the copying process does not provide any useful information. Naturally, the PIN code entered by the user must be XORed with the same random number before the comparison operation.

Differential power analysis (DPA) can reveal even finer differences in the current consumption of a microcontroller than simple power analysis. With DPA, the current consumption is first measured while the microcontroller is processing known data and then again while it is processing unknown data. These measurements are usually repeated many times so that noise can be eliminated by averaging. The difference between the two measurements is then calculated and used to deduce the unknown data.

A refinement of DPA called high-order differential power analysis (HO-DPA) is mentioned in a paper by Kocher *et al.* This involves measuring not only the current consumption of the microcontroller, but also other quantities that depend on the program being executed by the processor, such as the electromagnetic radiation of the chip. The measurement data collected using both known and unknown data is used in the same way as with DPA to calculate differences, which can then be used to compute the unknown data.

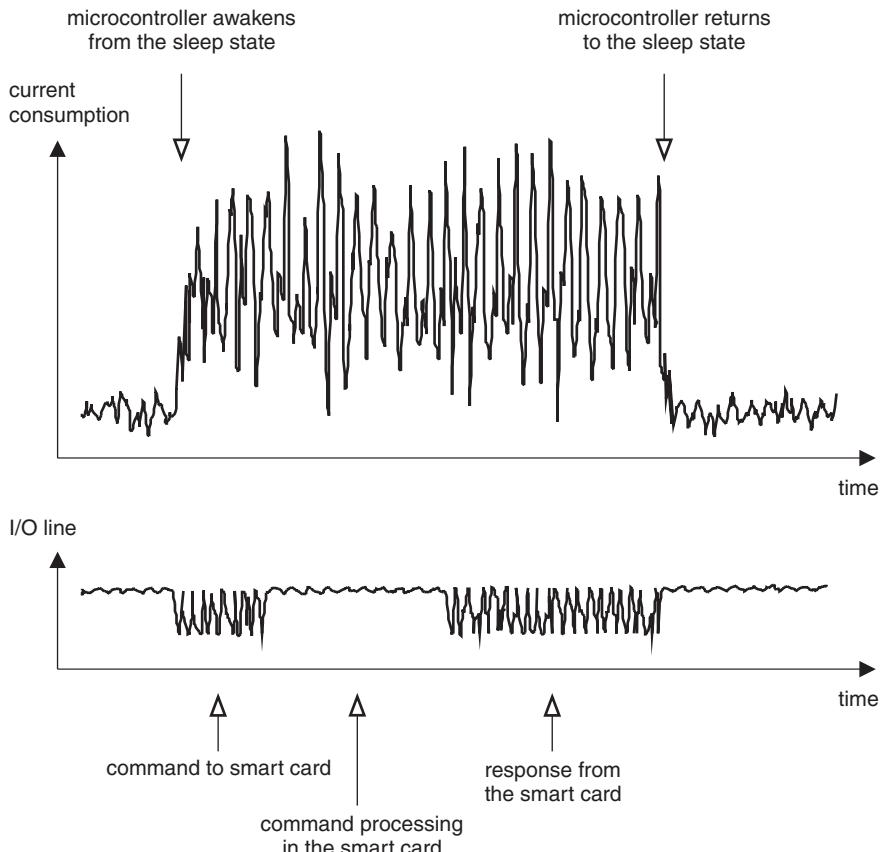
These three types of power analysis for smart card microcontrollers are very serious forms of attack on hardware and software that have not been protected by suitable countermeasures. This is because the current consumption of some microcontrollers is distinctly dependent on the executed machine instructions and the data processed by the instructions, as illustrated in Figure 16.30. In addition, the equipment resources necessary for a successful attack are reasonably limited. However, a variety of effective countermeasures based on suitably improved hardware and modified software are now available.

The simplest hardware solution is to incorporate a fast-acting voltage regulator in the chip to ensure that the current consumption remains constant, regardless of the instructions or data being processed. Artificial noise current generators on the chip are also an effective solution. A technically more complicated solution is to alter the semiconductor design of the processor so that it draws a constant current. However, all these approaches result in more or less increased microcontroller power consumption, which is undesirable in certain application areas such as telecommunication. An alternative, simpler defense measure is to activate certain components of the microcontroller that are not needed for the actual process while performing SPA/DPA-critical processes. A CRC checksum generator or numerical coprocessor could be used for this purpose, using random data as input in order to generate artificial noise in the current consumption.

Using randomly generated delays (random wait states) with the processor and in the operating system software considerably increases the difficulty of synchronizing the current measurements without incurring the penalty of increased power consumption. A similar approach is taken with smart card microcontrollers that have on-chip clock generators, by randomly varying the clock frequency within certain limits.

Due to the data traffic on the bus lines, the current consumption of a microcontroller is also data-dependent. This dependency can be reduced or in some cases even eliminated by using a dual-rail bus implementation. With this approach, the bus has twice as many lines as are actually necessary – for example, an eight-bit data bus has sixteen lines instead of eight. Each pair of bus lines is driven with opposite polarity, with the result that for every bus line that is driven high, another line is driven low. This provides effective hardware-level protection against SPA/DPA analysis because the net current consumption of the bus is independent of the transported data. Incidentally, high-security cryptoprocessors are also often implemented using this sort of complementary logic to hinder side-channel attacks such as SPA and DPA.

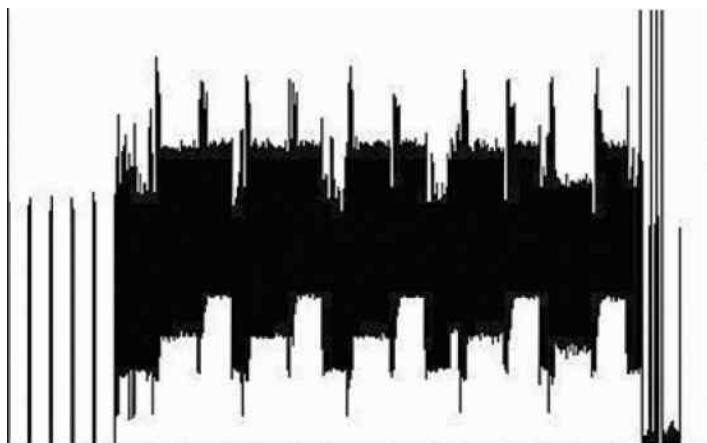
On the software side, an immense variety of countermeasures are now available. Here we briefly describe a few representative examples. The simplest approach is to use only machine instructions that have very similar current consumptions. In this case, machine instructions



**Figure 16.30** Simplified depiction of the current consumption of a smart card microcontroller in the quiescent state and the variation in its current consumption during operation. From the current drawn by the microcontroller, it is possible to recognize when it is awakened from the sleep state by the first falling edge on the I/O line, after which its current consumption varies continuously, depending on the machine instructions being executed

whose current consumption is significantly different from the average level are not allowed to be used in the assembly language code. Another approach is to have several different, randomly selected processes for performing the same computations in cryptographic algorithms. This makes it considerably more difficult for an observer to identify a relationship between known and unknown machine instructions or processed data. A similar sort of defense consists of using chip-specific tables for the S boxes of the triple-DES algorithm.

To impede the acquisition of the data necessary for successful power analysis, all keys should be protected by irreversibly decremented retry counters. In addition, it is necessary to block free use of all commands such as INTERNAL AUTHENTICATE that can be used to pass any desired data through a cryptographic algorithm in the smart card. If it is essential to use commands of this sort for some reason, the smart card must verify the authenticity of the terminal before executing them. Restricting command usage also impedes the collection of reference data for subsequent power analysis.



**Figure 16.31** Current consumption during execution of an UPDATE BINARY command, with the current plotted on the  $Y$  axis versus time on the  $X$  axis. The distinctly visible blocks in the current curve result primarily from write operations to nonvolatile memory. This current versus time curve is characteristic for a particular microcontroller type in combination with a particular operating system. The shape of the curve is usually so unique that the software being used can be deduced from the curve

It should be noted that, as a matter of principle, secret data should never be processed bitwise because bitwise processing considerably simplifies SPA/DPA analysis. This applies in particular to all algorithms for computing checksums of memory regions, such as CRC checksums. If keys must be loaded into the registers of a cryptoprocessor, in some implementations they are intermixed with random numbers that are also loaded into suitable registers as dummy values, which renders the resulting measurements useless. Of course, the true keys must be located in the proper registers at the end of the loading process.

SPA/DPA analysis is not limited to ferreting out secret data stored in smart cards. It can also be used for purposes, such as providing unequivocal proof that specific program code is used in a smart card. This is done by first performing an SPA analysis of the function concerned in the smart card under examination and then comparing the measured power consumption curve with the curve measured with a reference card, as illustrated in Figure 16.31. Under favorable conditions, this technique can, for example, be used to show that a competitor's product uses program routines obtained from elsewhere, even if the source code is not known. The technical basis for this is that a particular compiler usually generates the same machine code from the same source code. The differences arising from the subsequent linking process, due to the almost inevitable differences in memory localization, are relatively small.

Testing smart card software for resistance to SPA/DPA attacks has now become a sophisticated activity with all the earmarks of a specific field.<sup>9</sup> It has become common practice in software development to make periodic measurements and modify the software as necessary, depending on the measured results, in order to foil SPA/DPA attacks. At the start of software development, these measurements are made with the software in EEPROM, and the analysis can be extended and refined when the first sample microcontrollers with the software in ROM

<sup>9</sup> Power Analysis Attacks by Stefan Mangard *et al.* [Mangard 07] provides a good introduction to this subject



**Figure 16.32** A typical coil for measuring the electromagnetic radiation of a smart card microcontroller. An optical microscope used for precise positioning of the coil is visible at the top of the picture (Infineon)

are obtained from the semiconductor manufacturer. Experience has shown that this aspect is clearly significant with regard to SPA/DPA measurements.

SPA and DPA are naturally suitable for more than just mounting attacks on cryptographic algorithms. Both methods are also very suitable for analyzing all processor activities. With suitable experience and equipment, it is even possible to determine the data being copied from memory if the card is not resistant to this form of attack.

#### **Analysis and defense: measuring the electromagnetic radiation of the microcontroller**

Deductions regarding the internal processes of the smart card microcontroller can be made from its electromagnetic radiation by using a method similar to differential power analysis. It is readily possible to measure low-amplitude, localized magnetic fields by using small coils connected to low-noise amplifiers. The measurements can be analyzed in the same way as with SPA or DPA, and detailed knowledge of the internal structures of the semiconductor device is not necessary. This method is called simple electromagnetic analysis (SEMA) or differential electromagnetic analysis (DEMA).

To determine the optimum position of the coil, repetitive processing of a cryptographic algorithm is initiated in the microcontroller and the coil is held a short distance above the surface of the chip and moved in successive small increments until the strongest signal is received (see Figure 16.32). After this, the SPA or DPA method can be used to attempt to determine the secret key. The advantage of this approach relative to power analysis (SPA or DPA) is that localized measurements are possible, which can perhaps be used to reduce the influence of a noise source on the chip.

A drawback of this method is its distinctly higher noise level compared with direct power analysis via the supply lines. In addition, certain areas of the chip, such as the buses of semiconductor devices, can be effectively protected against this sort of attack by stacking several tracks on top of each other, with the result that, although, the electromagnetic field can be measured

using sensitive detectors, it is not possible to determine which of the tracks is the source of the signal. In addition, the metallic shields commonly placed over the active structures of the chip considerably reduce the radiated energy level because they form a sort of Faraday cage.

### **Manipulation and defense: altering the memory content of the smart card microcontroller**

Directly reading the memory content of a microcontroller is an attack scenario whose danger is immediately apparent. A similar scenario that is almost as strong a form of attack is intentionally altering the memory contents of a smart card microcontroller. This does not mean introducing random errors in the computation process of a cryptographic algorithm, which forms the basis of differential fault analysis (DFA), but instead selectively changing the values of certain bits or bytes in the ROM or nonvolatile memory. Attacks of this sort are collectively called fault induction attacks.

Nonselective changes in all types of memory can be produced by means such as exposing the module to X-rays or shining ultraviolet light on the exposed chip. EEPROM and flash memory cells can be discharged by exposing them to ultraviolet light, which causes their contents to assume the value of the lowest-energy state. This process is exactly the same as erasing a conventional EPROM using an ultraviolet lamp. It cannot reasonably be used for an attack, since the attacker has no control over which EEPROM or flash memory cells are altered.

However, ultraviolet light can be focused, or a narrow laser beam can be used instead. With this approach, it is certainly possible to alter the contents of individual memory cells. A laser has the advantage that if it is sufficiently powerful, it can also be used to alter the contents of ROM cells. A focused ion beam can also be used in a similar manner to alter the contents of memory cells.

It is certainly possible to use this approach to mount theoretically effective attacks. For example, the random number generator could be manipulated such that it no longer produces random numbers, but instead always supplies the same value. If this is possible, authentication of the terminal by the smart card can be broken by a replay attack using a previously recorded value.

Other forms of attack based on the intentional modification of specific memory locations are also conceivable. For example, all S boxes of the DES algorithm could be intentionally set to a uniform value of 0 or 1. This would mean that the DES algorithm would no longer act as an encryption algorithm, but only as a linear transformation [Anderson 96a].

If the exact location of the key in the EEPROM or flash memory is known and it is possible to modify individual bits of the stored key (using focused ultraviolet light, for example), it is naturally possible to utilize these conditions to mount an effective attack. The attack consists of setting an arbitrary bit of the key to 0 and then executing a command that uses the DES algorithm with the modified key. If the return code indicates a parity error in the key, the bit that was modified was originally set to 1, while if no parity error is reported, the bit was already set to 0. This procedure is repeated for the remaining 55 bits of the key, with the result that the secret key is known [Zieschang 98].

Many other types of attack along the same lines are conceivable, such as selectively modifying program processes or changing pointer contents. Although these attacks may appear very simple and attractive on paper, it would be very difficult to carry them out in practice. The necessary conditions for a successful attack are not exactly easy to achieve, so this form of attack remains an interesting but theoretical concept.

In order to alter bits selectively, an attacker must have detailed knowledge of the physical addresses of the data and program code in the memory, as well as the scrambling and/or encryption schemes of the memory in question. In addition, all data and routines that are relevant to security are protected by checksums that are always recomputed before the data or routine is used. This means that the attacker would also have to selectively modify the checksum to match the modified data. It should also be borne in mind that all shields over the memory concerned must be bypassed before any manipulation can take place. All of these considerations strongly reduce the attractiveness of this form of attack, although it admittedly sounds very attractive in theory.

### **Manipulation: differential fault analysis (DFA)**

As is well known, the operation of electronic devices can be severely impaired by electromagnetic interference. For instance, a mobile telephone can cause the processors of many types of small computer-controlled appliances to crash. The cause lies in the memory cells, whose contents can be altered by high-frequency alternating fields.

In 1996, Dan Boneh, Richard DeMillo and Richard Lipton published a study [Boneh 96] describing a theoretical method for determining the secret keys of asymmetric cryptographic algorithms by introducing scattered hardware errors. As the three discoverers of this method worked at the Bell Communications Research (Bellcore) Laboratories at the time, this type of attack is often called a Bellcore attack.

Only two months later, Eli Biham and Adi Shamir published an extension of the Bellcore attack called differential fault analysis (DFA) [Biham 96], which also included symmetric cryptographic algorithms such as DES. This meant that, at least in theory, many smart card applications were exposed to a new and serious form of attack.

The basic principle of both of these forms of attack is relatively simple. In the first step, an arbitrary plaintext is encrypted using the key to be broken, and the resulting ciphertext is saved. Following this, the operation of the card is disturbed by external interference while it is processing the cryptographic algorithm, for example by exposing it to ionizing radiation or high-frequency fields, in order to alter a single bit of the key in a random location while the computation is being performed. This yields a ciphertext that is incorrectly encrypted due to the altered bit. This process is repeated many times, and all the results are saved for analysis. The remainder of the procedure for determining the value of the secret key is purely mathematical, and it is fully described in the cited papers.

The strength of this attack is primarily due to the fact that it is not even necessary to know the location of the altered bit in the secret key. Biham and Shamir state in their publication that with a single corrupted key bit, 200 ciphertext blocks are sufficient to compute the value of the secret DES key. If triple DES (with a 168-bit key) is used in place of simple DES, the number of required ciphertexts does not increase significantly. Even if more than one bit is altered, this attack remains effective; the only consequence is that more incorrectly encrypted ciphertexts are needed.

In practice, this type of attack is not as simple as it sounds. If at all possible, only one bit should be altered, or at least only very few bits. If the entire microcontroller is simply bathed in microwave radiation, usually so many bits will be altered that the processor will simply crash. Consequently, an attempt is made to induce isolated errors in the processor by injecting specially prepared glitches in the power or clock lines. If the filters on the associated input leads cannot neutralize these glitches, they can produce the desired processing errors.

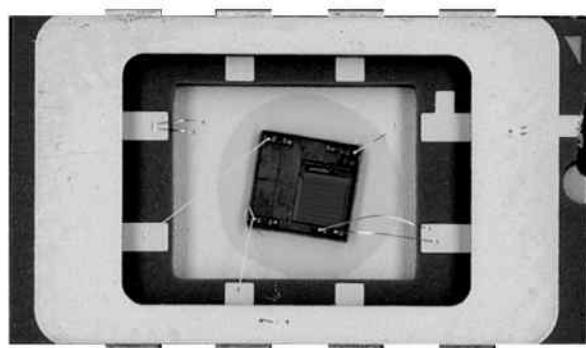
However, a smart card is not totally helpless in the face of a Bellcore attack or DFA if suitable precautions are taken. The easiest defense is simply to compute the cryptographic algorithm twice and compare the two results. If they match, no attempt has been made to alter any bits from outside the card. This defense assumes that intentionally introduced random errors can never alter the same bit twice in a row. This is a realistic assumption, since if it ever became possible to alter specific bits in a smart card processor selectively, attacks that are much simpler and faster than DFA would be possible.

The main disadvantage of double computation is the additional elapsed time, which can cause problems. This applies primarily to attacks on time-intensive asymmetric cryptographic algorithms, such as RSA and DSS. Another effective defense against differential fault analysis can be achieved by always encrypting different plaintexts. The simplest solution is to prefix the plaintext to be encrypted with a random number. This means that the cryptographic algorithm always encrypts different data, which prevents DFA from being used.

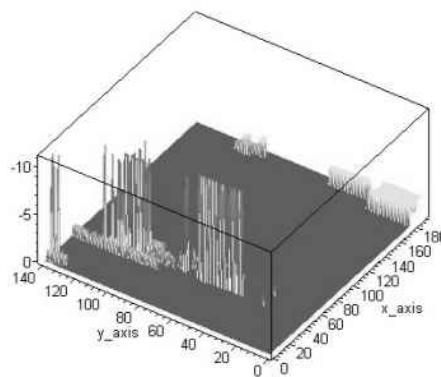
In summary, the Bellcore attack and differential fault analysis are unquestionably dangerous forms of attack that can succeed with smart cards that do not incorporate adequate protection measures. However, all smart card operating systems and applications were modified to protect them against these forms of attack shortly after they became known, so neither the Bellcore attack nor DFA currently represents a serious threat.

### Attack and defense: light attack on the processor

In a form of attack on the secret keys of cryptographic algorithms similar to differential fault analysis, an attempt is made to affect the execution of program code by interfering with the operation of the processor. This form of attack, which has been known to manufacturers of smart cards and smart card microcontrollers since around 1998 as ‘light attack’, was described in mid-2002 by Sergei Skorobogatov and Ross Anderson [Skorobogatov 02] as ‘optical fault induction attack’. Their paper describes an arrangement in which a standard commercial flash unit is attached to the camera adapter flange of a conventional optical microscope. A highly restricted region of the RAM of a standard microcontroller (PIC16F84) is then exposed to light from the flash unit. With microcontrollers that are not hardened against this form of attack, this arrangement can be used to selectively set certain bits in the RAM to the logic 0 or 1 state.



**Figure 16.33** An exposed smart card microcontroller in a DIL package, suitable for further study (Reproduced with permission from Flylogic)

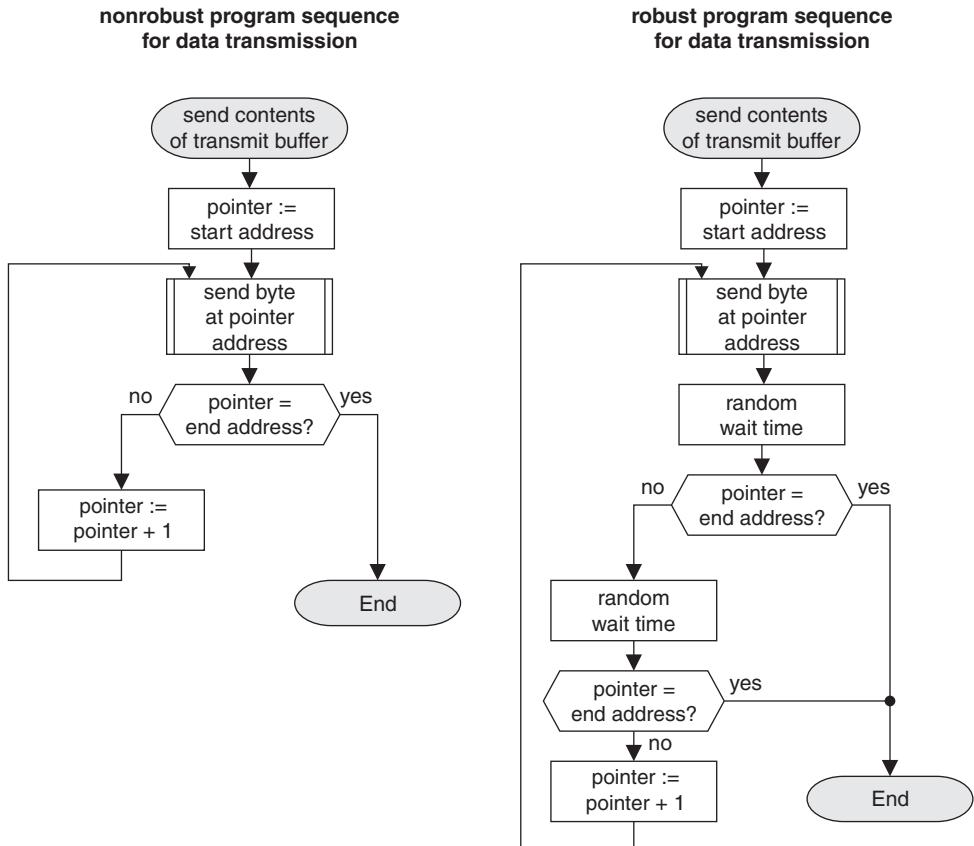


**Figure 16.34** Sensitivity of a microcontroller to infrared light: the height of the column above the chip surface indicates the light sensitivity (Reproduced with permission from Giesecke & Devrient)



**Figure 16.35** A workstation for testing the sensitivity of smart card microcontrollers to light flashes. A laser connected to an optical microscope can be seen in the middle, with the control computer of the microcontroller on the left (Infineon)

Nowadays lasers are used instead of flash tubes to produce the flashes of light. These flashes often have a duration of only a few nanoseconds, and they are sometimes highly focused so that only small areas of the chip are affected. Light in the visible as well as the infrared spectrum can be used for this form of attack (see Figure 16.34). Depending on the attack scenario, a series of flashes with precisely controlled timing may also be used. The angle of incidence of the light may be perpendicular to the chip surface, oblique, or even from the side. In practice, laser cutters are typically used for this form of attack, with the power level reduced to the point that it does not destroy the chip structure.

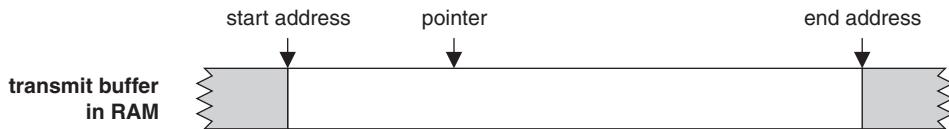


**Figure 16.36** Examples of nonrobust and robust routines for transferring the contents of the transmit buffer shown in Figure 16.37 on the facing page. The process on the left can be attacked successfully with a single processor disturbance. The process on the right is hardened by an additional query and a dual wait loop with random duration

Incidentally, attacks based on interfering with the processor are distinctly easier if the source code of the program being executed is known. In this case, the attacker knows the location and timing of critical decision points in the program flow. The attacker might also know the software measures employed to defend against this form of attack. If the attacker knows the source code, it is considerably easier to synchronize the timing of the laser pulses, as otherwise the critical timing must be determined by laborious trial and error. In this regard, confidential software is preferable to open-source software.

In addition to light flashes, spikes or glitches on the supply lines or high-frequency signals can be used to disturb the processor.<sup>10</sup> If the disturbance is triggered at the right time in the program flow, specific decision processes in the program can be affected. A simple example is shown in Figure 16.36. The task of the routine shown in the figure is to transfer the contents of a transmit buffer, whose boundaries are specified by a start address and an end address. If the

<sup>10</sup> See [Lamla 00] and [Bar-El 04]



**Figure 16.37** Typical structure of a smart card operating system transmit buffer

attacker manages to interfere with the instruction that tests for the end address of the transmit buffer, data located after the end of the buffer will also be sent to the terminal. If this memory region happens to be the working memory of a cryptographic algorithm and holds its secret keys, these keys could be read illicitly in this manner.

The defense against this attack involves several levels. At the hardware level, it is important for the smart card microcontroller to have suitable sensors so it can detect all attempts to disturb the processor. These sensors can be voltage sensors to detect glitches or spikes and a large number of suitable light sensors on the chip. To make it impossible to defeat a few light sensors by covering them with black ink, it is a good idea to have a fairly large number of sensors distributed over the surface of the chip. This by itself is sufficient to preclude many forms of attack. An opaque chip encapsulation material provides only limited protection, since it can be removed relatively easily using chemical methods.

The second level of protection must be implemented in the software. In the example shown in Figure 16.36 on the facing page, the program code can be made significantly more robust by using a ' $\geq$ ' query instead of an '=' query. Another possible countermeasure is to execute the query twice with a random interval between the two queries. This requires the attacker to use two flashes of light to manipulate the query, with the additional difficulty that the timing of the second flash is not precisely predictable.

The current state of the art for countermeasures is card-specific timing for critical queries combined with multiple queries. Another common practice is regularly to call dummy routines scattered through the program flow under the control of a random timer. However, care must be taken to ensure that these dummy routines do not consume too much processing time, as otherwise the processing speed of commands will be impaired.

In addition, all confidential data in RAM should always be deleted immediately after use or be temporarily encrypted. To further mitigate the consequences of this form of attack, it is also a good idea to encrypt all secret data (such as PIN codes and keys) located in memory. If an attacker manages to read out portions of the memory by manipulating queries, the resulting data will be of no use to the attacker if it is encrypted.

Light attacks can also be used to affect the contents of certain processor registers. To prevent this form of manipulation when parameters are passed to functions, the registers can be protected by checksums that are tested by the calling function. This test can be repeated several times to circumvent a second flash intended to negate the first test. Multilevel security measures of this sort should be included in an operating system only after careful deliberation, as they come at the price of memory space, processing time, and robustness.

If a memory management unit (MMU) is available, it can also be configured to monitor compliance with specific boundaries when sending data. In addition, modern processors can detect illegal machine instructions and invalid addresses and respond appropriately. As can be clearly seen from this defense scenario, a form of attack that certainly can be regarded as serious can be blocked by a suitable combination of protection measures in hardware and software.

### 16.5.2 Attacks on the operating system

Protective mechanisms in the hardware form the basis for protective mechanisms in the operating system software. All potential weaknesses must be dealt with, since the three components of the protective mechanisms – hardware, operating system and application – are mutually dependent. Like a chain, their strength is determined by the weakest link. If a specific mechanism in a smart card fails, the entire security of the card collapses. The operating system in particular forms the basis for the applications whose data and processes must be protected.

Here we devote particular attention to protection against typical attacks, rather than general smart card security functions. However, these general functions also make a significant contribution to operational reliability and protection against attacks. In this regard, we refer you to the discussion in Chapter 13, ‘Smart Card Operating Systems’, on page 441.

#### **Analysis: determining the command set of a smart card**

Although the command classes and commands supported by a smart card are usually not public knowledge, is very easy to determine what they are. This is more useful for determining the basic command set of a smart card than for mounting an attack on the card, but it is conceivable that this information could be used as the basis for an attack.

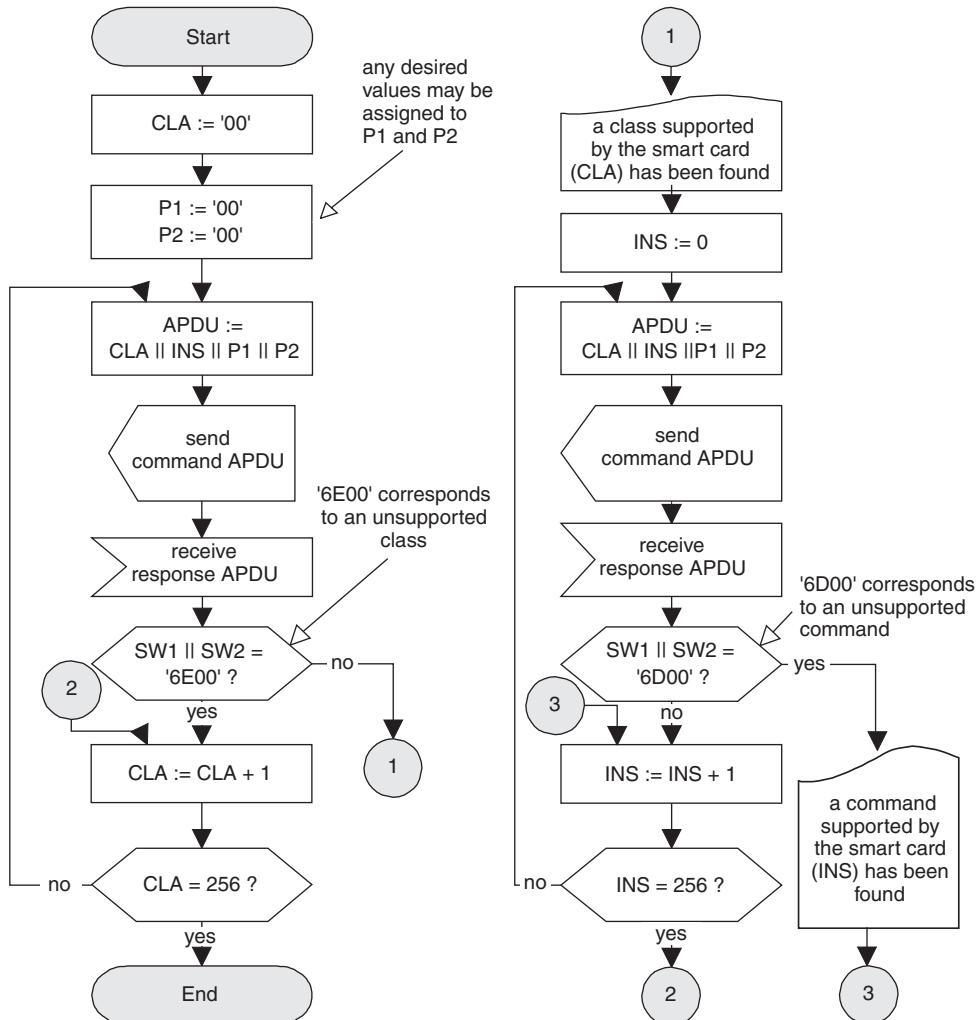
The procedure for determining the command set is briefly illustrated in Figure 16.38 on the next page. The first step is to generate a command APDU and send it to the smart card using a freely programmable terminal. A different class byte is used each time to explore the range from ‘00’ to ‘FF’. If a return code other than ‘invalid class’ is received, a valid class byte has been found. There are usually two or three valid command classes, which can then be used to try all possible instruction bytes in the next round. This consists of sending command APDUs with different instruction bytes to the smart card and noting the ones that yield a return code other than ‘unknown instruction’. If suitable software is available in the terminal, it takes only a few minutes to determine which commands are supported by a particular smart card. A similar procedure can be used to determine at least some of the command parameters supported by the identified commands.

This algorithm can be accelerated significantly by using only the class byte codes specified by the ISO/IEC 7816-4 standard and making the instruction byte an indexed variable. This strongly reduces the number space of the class byte by taking secure messaging and logical channels into account. The instruction byte can also be limited to even numbers because the odd-numbered codes contain only Vpp control information, which is no longer used.

The effectiveness of this simple search algorithm for instruction classes, commands and parameters arises from the fact that nearly all command interpreters in smart card operating systems evaluate received commands by starting with the class byte and working through the following bytes. Processing stops as soon as the first invalid byte is found, and a suitable return code is generated and sent back to the terminal.

This procedure only works if the smart card does not have a general state machine that controls the command sequence. Even if it does, the procedure can be used to determine the allowed command sequence by using an iterative approach.

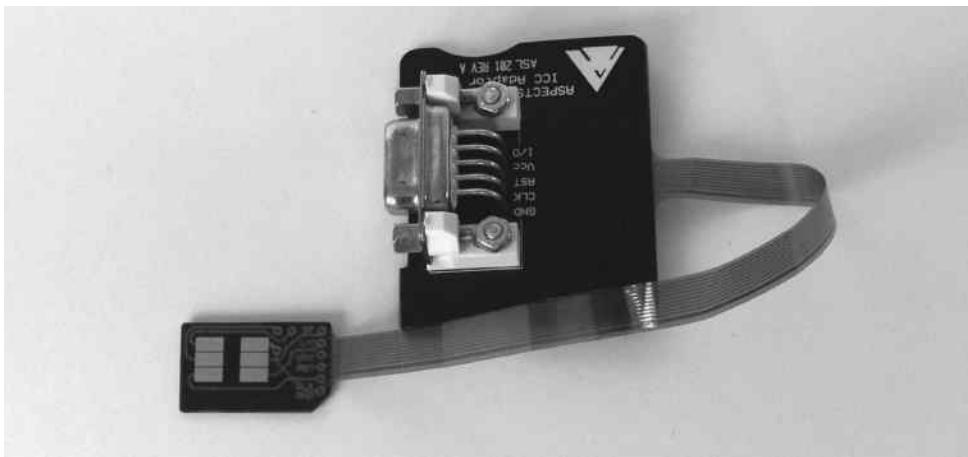
Although the utility of this for an attacker may not appear especially large, since the command set is usually not secret, it at least allows all possible commands to be determined quickly and easily. Incidentally, it is also very useful for determining whether the operating system producer implemented any undocumented commands in the smart card.



**Figure 16.38** Basic exhaustive search procedure for determining the commands supported by a smart card operating system. A complete search is only possible if the command execution sequence is not controlled by a state machine. The operating principle consists of systematically testing all class byte (CLS) and instruction byte (INS) codes in turn while ignoring any other codes, such as secure messaging, logical channels, Vpp control, and so on

### Attack: intercepting data transmission

A slightly modified smart card can be used to intercept data during a session and to manipulate the data as necessary. The modifications consist of gluing an insulated dummy contact on top of the I/O contact so that the original I/O interface is no longer connected to the I/O contact. The new contact and the original I/O contact are then connected to a fast computer. With a suitable program, the computer can delete or insert any desired data in the transfers between the terminal and the smart card. If the computer is sufficiently fast, neither the terminal nor the card will detect any difference between normal and manipulated communication.



**Figure 16.39** An adapter for plug-in modules used to intercept data transfers between the mobile equipment and the SIM card

It is clear that the course of a session can be radically affected using this method. Whether an attacker can derive any advantage from this depends primarily on the application in the smart card. A generally accepted design principle is that it must not be possible to impair security by intercepting data or by deleting or inserting data during the communication process. If this principle is not observed, an attacker can certainly obtain an advantage by using this method. There are known cases of fraud using simulated memory cards.

For protection against this form of attack, some terminals have shutters that cut through any wires attached to the smart card. Secure messaging<sup>11</sup> can also be used very effectively to enable reliable detection of any manipulation of the data during transmission.

Many terminals can be used only under supervision, which makes it difficult for an attacker to use a manipulated card with leads connected to an accompanying computer. In summary, although this form of attack can be regarded as theoretically very interesting and quite promising, in practice it yields only modest results.

### Attack and defense: power interruption

A form of attack that was successful with many smart cards up to a few years ago is to interrupt power to the card at a specific time during command execution. This form of attack is based on the fact that with conventional programming, all write operations to EEPROM or flash memory pages are performed sequentially. If the programmer arranged the sequence of write operations unwisely, an attacker can derive an advantage by interrupting power at the right time.

This can be briefly illustrated by using a highly simplified example (see Figure 16.40). If the balance of an electronic purse is increased before the log file is updated when a purse load command is processed, an attacker would have a good chance of loading the smart card at no cost. This only requires interrupting the power to the card at the right time or suddenly pulling the card out of the terminal with millisecond accuracy. In this case, the purse balance would be increased to the new value, but the transaction would not be logged and no response to the

<sup>11</sup> See also Section 8.4, 'Secure Data Transmission', on page 225

operation	purse balance	purse balance file (in binary notation)
1. Current purse balance	100 euros	0110 0100
2. Debit 10 euros	90 euros	
3. Erase EEPROM	255 euros	1111 1111
4. Write the new purse balance	90 euros	0101 1010

**Figure 16.40** Example process for updating the balance of an electronic purse. Here it is assumed that the erased state of the EEPROM is logic 1. For technical reasons, this means that the entire EEPROM page must be erased (set to 1) if any bit in the page must be changed from 0 to 1. If in this example the power to the smart card is interrupted exactly after the EEPROM has been erased (after step 3), the purse balance will be set to its maximum value and the attacker will have effectively created money. This can be reliably prevented by using atomic operations

command would be sent. In this past, this form of attack was certainly feasible with simple electronic purse systems.

To determine the exact time to interrupt processing, the attacker only has to use an electronic counter to count the clock pulses after transmission of the command and then use successive trial and error to determine the right time. It hardly needs saying that the entire process can be automated with a computer.

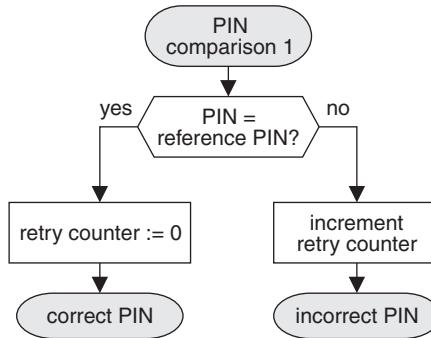
Although this form of attack appears both attractive and simple, in practice there are several effective countermeasures. The simplest approach consists of a reasonable arrangement of the EEPROM or flash write instructions. The EN 1546 standard for multisector electronic purses is well worth examining in this regard, since all the electronic purses described in this standard are explicitly protected against this form of attack.

However, even a perfectly arranged sequence of write operations cannot by itself provide absolute protection. This can be illustrated using another example. When the electronic purse in the previous example is being loaded, it may be necessary to erase the EEPROM or flash memory before the write operation. If the erased state of the memory corresponds to the maximum value of the purse balance, which incidentally is the usual case, the purse can be artificially loaded to its maximum value by simply interrupting power to the card at the right time. The right time is when the erase operation has just been completed and the write operation has not yet been started.

Operating system designers know an effective countermeasure to this form of attack, which is to use atomic operations as described in Section 13.10 on page 480. Atomic operations are indivisible, which means they cannot be split. In other words, they execute either completely or not at all, which provides adequate protection against the form of attack described above. Even the optimally ordered EEPROM/flash write operations described in the EN 1546 standard require atomic operations in several places to prevent this form of attack.

### Attack and defense: power analysis during PIN comparison

A technically very attractive form of attack on comparison features, such as PINs, can be carried out using a combination of physical measurement of a parameter and variation of



**Figure 16.41** A PIN comparison routine that compares a PIN code sent to the card with the PIN code stored in the card. This function is not protected against attacks

logical values. It applies to all mechanisms in which data is sent to the smart card and compared with a value stored in the card, with a retry counter being incremented according to the result of the comparison.

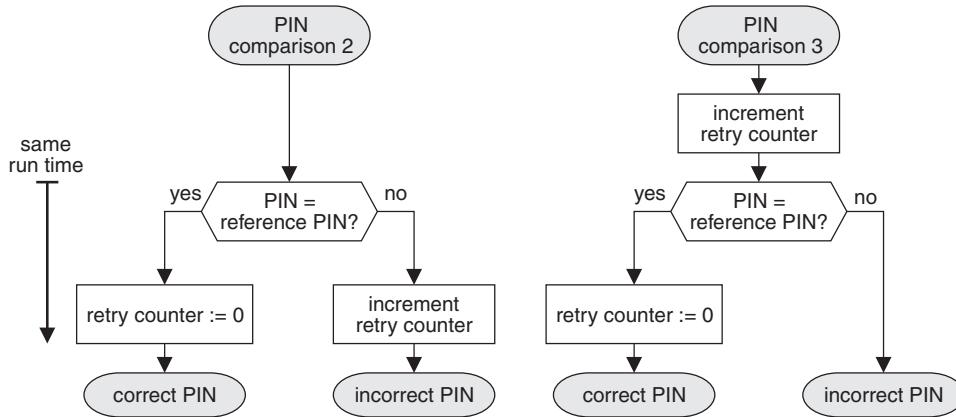
The attack operates on the principle of measuring the current drawn by the card, for example by measuring the voltage drop across a resistor in the Vcc lead.<sup>12</sup> If a suitable command with the comparison data is sent to the card, it can be seen from the current measurement whether the retry counter has been incremented, even before the return code is received. If the return code is sent before the retry counter is updated when the result of the comparison is positive, as illustrated in Figure 16.41, this method can be used to determine the value of the reference data. This is done by systematically sending all possible comparison values to the smart card and interrupting power to the card before the retry counter has been incremented if the result is negative. A positive result can be clearly recognized from the corresponding return code, which is sent before the retry counter is written.

There are two basic ways to defend against this form of attack, as illustrated in Figures 16.42 and 16.43. The simplest defense consists of always incrementing the retry counter before making the comparison and then decrementing it afterward as appropriate. With this approach, the attacker cannot obtain an advantage, regardless of when power to the card is interrupted, since the retry counter will have already been incremented. The second option is more complicated, but it provides the same protection. In this approach, the retry counter is incremented after a negative comparison, or written to an unused EEPROM cell after a positive comparison. Both of these write operations occur at the same time in the process, so the attacker cannot deduce the result of the comparison. The attacker must wait for the return code to learn the result of the comparison, at which point it is too late to prevent a write operation on the retry counter by interrupting the power.

### Attack and defense: timing analysis of PIN comparison

Programmers always devote a lot attention to writing code that runs as quickly as possible. This is usually important, but the effects of runtime minimization can also be utilized for very promising forms of attack. If a PIN is sent to a smart card for comparison, the associated

<sup>12</sup> See also Figure 16.28 on page 701



**Figure 16.42** Compared with the routine shown in Figure 16.41 on the preceding page, these two PIN comparison functions are hardened against certain forms of attack. The PIN comparison result with function 2 cannot be recognized externally by measuring the current consumption of the EEPROM or flash memory write routine because it has the same processing time for positive and negative comparisons. PIN comparison function 3 is insensitive to power interruption because the retry counter in nonvolatile memory is incremented before the comparison and is subsequently reset if the result is positive

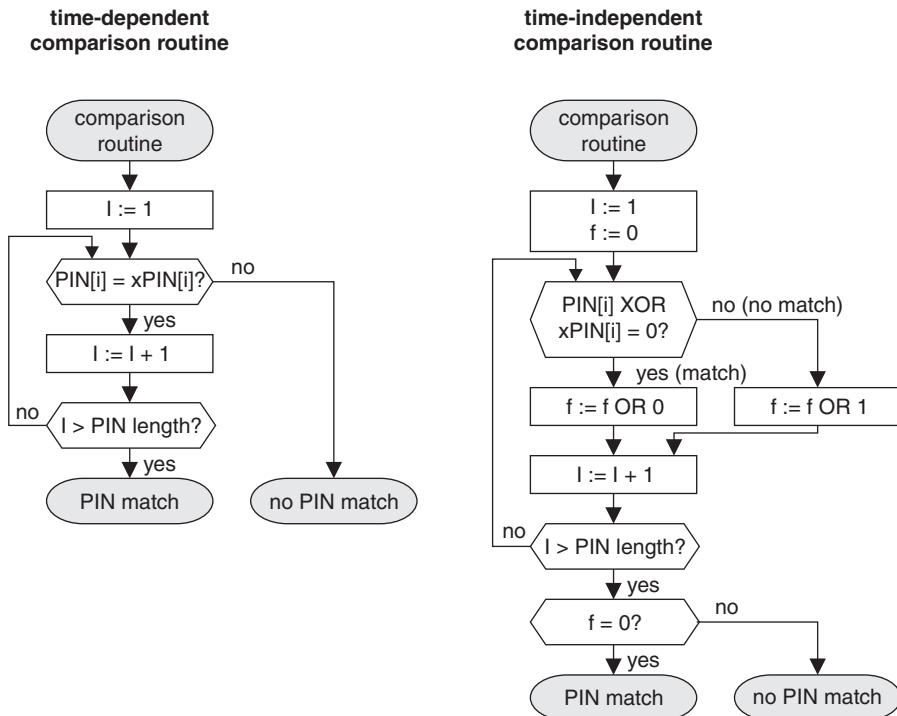
comparison routine normally compares the PIN it receives with the stored PIN value byte by byte. A programmer who is not security conscious will program this routine such that the first difference between the two compared values causes the routine immediately to terminate and to return to the calling program. This leads to small runtime variations in the comparison routine, which can be measured using suitable equipment (such as a storage oscilloscope). This can be used by an attacker to determine the secret PIN code in a relatively straightforward manner.

Up to a few years ago, this was still an effective form of attack on smart cards. However, it is now a known form of attack, and comparison routines are constructed such that all digits of a PIN are always compared. Consequently, there is no time difference between positive and negative comparison results.

### Protection: noise-free cryptographic algorithms

The security of smart card applications is based on the secret keys of the cryptographic algorithms. Certain types of access to the smart card or operations with the smart card require prior authentication of the terminal, and a secret key is used for this purpose. Naturally, authentication of the terminal by the card represents an attractive target for an attacker. By contrast, authentication of the card by the terminal is not attractive because this process can be manipulated as desired using a (dummy) terminal.

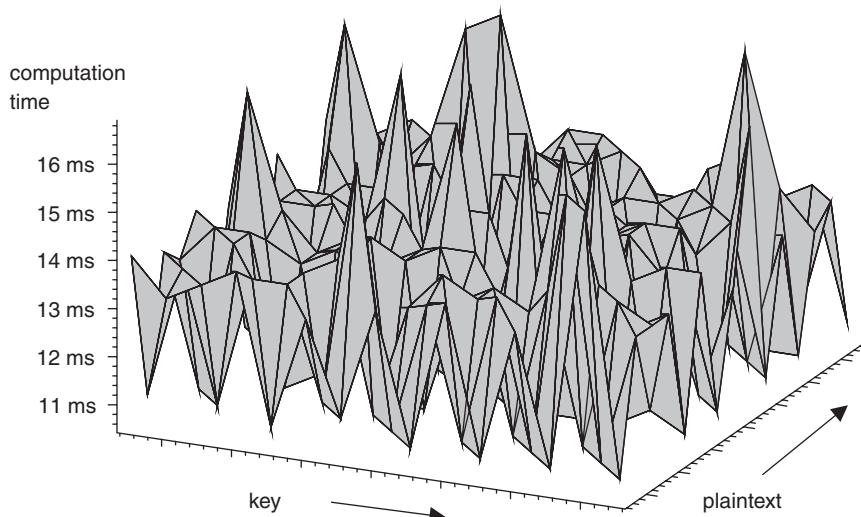
The smart card authenticates the terminal by sending it a random number, which the terminal then encrypts and returns to the card. The smart card then performs the same encryption and compares the result with the value received from the terminal. If the two values match, the terminal has been authenticated, and it receives a corresponding return code. If the authentication fails, the card sends a different return code. The starting point for this form of attack consists of analyzing the processing time between the command and the response from the smart card.



**Figure 16.43** Two different example routines for comparing secrets such as PIN codes or passwords. The comparison routine on the left terminates the comparison when it finds the first mismatch. This is the usual approach with runtime-optimized processes, but it can be utilized as a starting point for an attack. The routine on the left always compares all elements of the secret and is designed to have balanced execution paths, so it always takes the same time to perform a comparison, regardless of the result. In addition, the routine on the right uses an XOR operation for the comparison instead of a direct comparison instruction ( $=$ ), since the number of clock cycles necessary to execute a direct comparison instruction depends on the result with most processors. The following variables are used here:  $i$  = counter for the number of positions to be compared;  $f$  = variable used as a marker for stepwise comparison;  $PIN$  = storage location of the secret in the smart card;  $xPIN$  = storage location of the secret sent for comparison

As late as the early 1990s, cryptographic algorithms having significant variation in execution time with different keys and plaintexts were still used in some applications (see Figure 16.44). An attacker can exploit the resulting reduction in the size of the key space by using a brute-force attack to search for the secret key. The duration of the search is strongly dependent on the timing noise of the algorithm. A high noise level results in a correspondingly large reduction in the size of the key space and accordingly makes the key search easier and faster. If the exact implementation of the algorithm on the target computer is known, this information can also be included as reference data for generating the timing tables. This form of attack was publicized under the name ‘timing attack’ in a publication by Paul Kocher in 1995 [Kocher 95], which primarily dealt with the time dependencies of the RSA and DSS algorithms.

In principle, a timing analysis is a very severe threat to the security of a smart card. However, this form of attack has been known for a relatively long time, and all modern smart



**Figure 16.44** Example of the effects of ciphertext and plaintext data with an encryption algorithm that is not noise-free. The figure shows part of the plaintext/ciphertext space. It was generated using a previous implementation of the DES algorithm with 100 000 iterations per measured value

cards use only noise-free cryptographic algorithms, which means algorithms whose encryption or decryption time is independent of the input data. This effectively blocks this form of attack. However, programmers face a conflict of interest here, since a noise-free algorithm usually requires more program code and is always slower than a noisy version. The reason for this is that a noise-free algorithm must be designed such that the length of the execution path is the same for all combinations of plaintext, ciphertext, and keys. This means that the longest execution path is the reference value, and all other paths must be adjusted to match this length.

To provide additional security, in some applications each authentication key has its own retry counter, so that only a limited number of unsuccessful authentications can be performed. Once the retry counter has reached its maximum value, the smart card blocks all further authentication attempts.

### Protection: hardware and software tests after a reset

When the operating system is initialized, the major hardware components must always be tested to verify that they are in good working order. For instance, a RAM test is indispensable because all access conditions are stored in RAM during runtime and failure of a single bit could cause a complete security collapse. It is likewise necessary to compute and test the checksums of the most important regions of nonvolatile memory. The CPU is at least implicitly tested by sending the ATR, since the bulk of all possible machine instructions must be executed faultlessly for this to be possible. Explicit testing of the CPU and any NPU that may be present can usually be limited to sample testing, since completely testing all functions for flawless operation would take too much time and code.

FIPS 14-2 also specifies that the random number generator should be checked immediately after device activation. This consists of generating a series of random numbers and subjecting them to statistical tests. If they meet the specified quality standards, it can be assumed that the

random number generator has not been manipulated and the operating system can use it as necessary. Otherwise the usual response is to terminate operation with an error message.

If the operating system discovers a hardware error or checksum error, there are two possible courses of action. The first option is for the software to immediately jump to an endless loop, which means that an ATR cannot be sent and subsequent commands cannot be received. The main disadvantage of this is that the cause of this behavior cannot be recognized from the outside. The problem could be a broken bonding wire, a fractured chip, or a memory checksum error, but this cannot be determined by the user. A better option is thus to have the smart card attempt to send a special ATR before disabling itself by entering an endless loop. The error ATR at least gives the outside world an indication of what has happened inside the smart card. However, it must always be borne in mind that even sending an error ATR requires a largely functional CPU, a few bytes of working RAM, and several hundred bytes of program code.

### **Protection: layer separation in the operating system**

Layer separation with clearly defined parameters for transitions between the individual layers is a sign of a stable and robust smart card operating system. The consequences of possible design or programming errors in the operating system are minimized by clean separation of the layers in the operating system. Of course, this does not mean that such errors will not occur, but the effects of the errors will not be as extensive as with an operating system programmed in very compact, condensed code. Layer separation makes it difficult for any error that occurs in one layer to propagate to other layers.

### **Protection: supervised data transmission**

Another important element of security is supervision of data transmission in order to protect the memory against unauthorized access. All communication to and from the smart card takes place via an I/O interface supervised by the operating system. No other form of external access is possible. This constitutes an effective form of memory protection in the smart card because it ensures that the operating system always retains control over access to memory regions.

The transmission protocol, which is controlled by the transport manager, must intercept all possible incorrect inputs. There must be no possibility of influencing the data transmission process by manipulating transfer blocks in order to cause data to be illicitly sent from the memory to the terminal.

Memory overflows resulting in the injection of executable code, which is often a successful form of attack in the PC world, must be entirely impossible with smart cards. The mechanism of this attack is to send variable amounts of random data in response to requests from other computers in a network, with the intention of causing data to be written past the end of the receive buffer of the target computer. An executable program is appended to the random data and located in the message such that it will be executed by the computer after a buffer overflow occurs. This program enables the attacker to take control of the computer, after which the attacker can use it for any desired activities. For many years, this form of attack has repeatedly been astonishingly successful, but it must be strictly prohibited with smart cards, as otherwise the entire security system could be bypassed. Protection against this form of attack is achieved by strictly monitoring the receive buffer and rigorously terminating reception if the buffer would otherwise be overwritten. Incidentally, this would work equally well in PCs, but quality awareness is evidently not as well developed in that realm as in the realm of embedded systems.

### Protection: checksums of important memory contents

File structures, and in particular file headers (file descriptors), should be protected by checksums. This enables the operating system to at least detect any unintentional changes to data stored in memory. This requirement is especially important because the object-oriented access conditions for each file are stored in the file header.

All memory regions that are vitally important to the smart card operating system must be protected by checksums (EDCs). Whenever such a region is accessed or a program routine located in the region is called, the consistency of the memory contents must be verified before the access or code execution is allowed to proceed.

Care must be taken to ensure that checksum calculations are not performed bitwise, since bitwise processing provides a good starting point for power analysis (SPA or DSA), which in the worst case would allow the data being processed to be determined externally. This would make checksum testing detrimental instead of beneficial.

### Protection: encapsulation of applications

Some operating systems encapsulate the individual DFs containing the applications and their files, so that the applications are isolated from each other. This concept is based on software protection alone, with no support from the chip hardware. Consequently, the amount of protection is not as great as it could be. Nonetheless, even this software approach to application encapsulation can be very beneficial in case of an error, since it makes it impossible for the file manager to exceed the boundaries of a DF without explicit prior selection. As a result, the effects of a memory error on the files are at least limited to the DF concerned.

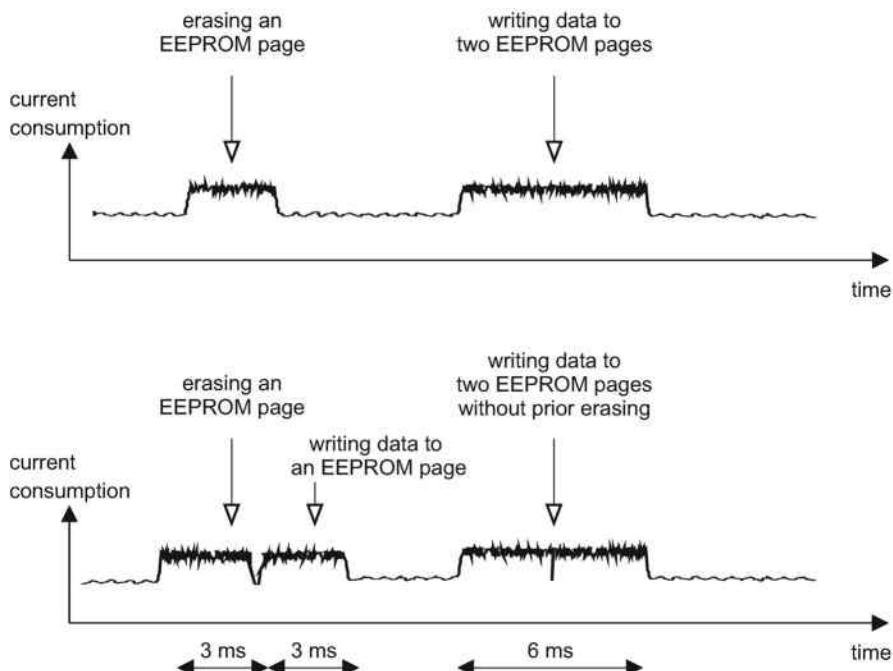
If hardware support for the operating system is available in the form of a memory management unit (MMU), individual applications can be fully encapsulated such that even manipulated software in an application cannot obtain unauthorized access to the memory regions of other applications.

A similar approach is taken with executable code by using a firewall or sandbox. This mechanism ensures that each program can only access data and code within the region allocated to the program. Any violation of the access boundaries leads to immediate termination of program execution. The access boundaries can be monitored in software, or better yet in hardware with an MMU.

### Protection: concealing operating system activities

When data must be written to the EEPROM or flash memory, the charge pump in the chip must be switched on. This increases the current consumption of the chip as illustrated in Figure 16.45, and with some types of microcontrollers this can be recognized with suitable test equipment. Consequently, the possibility that write operations to EEPROM or flash memory may be externally detectable must be taken into account in the design of the operating system. The software in the smart card must prevent an attacker from being able to derive any benefit from this.

It is very important that it should not be possible to make any useful deductions regarding the internal processes and decisions of the machine code by measuring the current drawn by the card. For example, it would be fatal if were possible to use such measurements to reliably judge the outcome of a PIN comparison before the completion of command processing and transmission of the return code, since this information could very easily be used to analyze the PIN code.



**Figure 16.45** Approximate depiction of the current consumption of a smart card microcontroller when the charge pump is switched on. A write operation to an EEPROM page with prior page erasure is shown on the left. An erase operation is not necessary in the transaction on the right, so here the write operation can take place immediately. The duration of an erase or write operation is 3 ms. The brief pauses when switching from erasing to writing or between writing two different pages are clearly visible

### Protection: object-oriented access conditions

Early smart card applications were always based on a centrally administered access control mechanism. A disadvantage of a centralized access control mechanism is that software errors or memory errors can affect the overall security of the smart card. Modern object-oriented file management systems with the access conditions held in the individual files have the advantage that a memory error affects only one file, with the security of all other files remaining intact. This is actually a fundamental property of all distributed systems. They are somewhat more difficult to program, but the self-contained nature of the individual components makes these systems significantly more resistant to attacks and resilient in case of errors.

### Protection: smart card deactivation

The operating system must allow the smart card to be fully deactivated. This is very important for the final phase of the smart card life cycle. With statistical methods, it is possible to perform very exact analyses of the software in the chip by collecting discarded but still fully functional smart cards. To prevent this, the operating system must have methods available that allow the operating system and all of its routines to be fully deactivated, so that it is impossible to analyze the electrical or runtime behavior of the cards.

### Attack and defense: random number generator

The random numbers generated by the smart card are used in authentication to individualize a session, which means to make each session unique and different from all preceding and subsequent sessions. The objective of this is to make it impossible to successfully replay data that has been obtained by eavesdropping on a previous session. Another form of attack would be to have the smart card generate so many random numbers that their sequence becomes predictable. Yet another possibility is to keep requesting random numbers from the smart card until the EEPROM or flash memory used by the random number generator no longer works properly, with the result that the same number is generated over and over again.

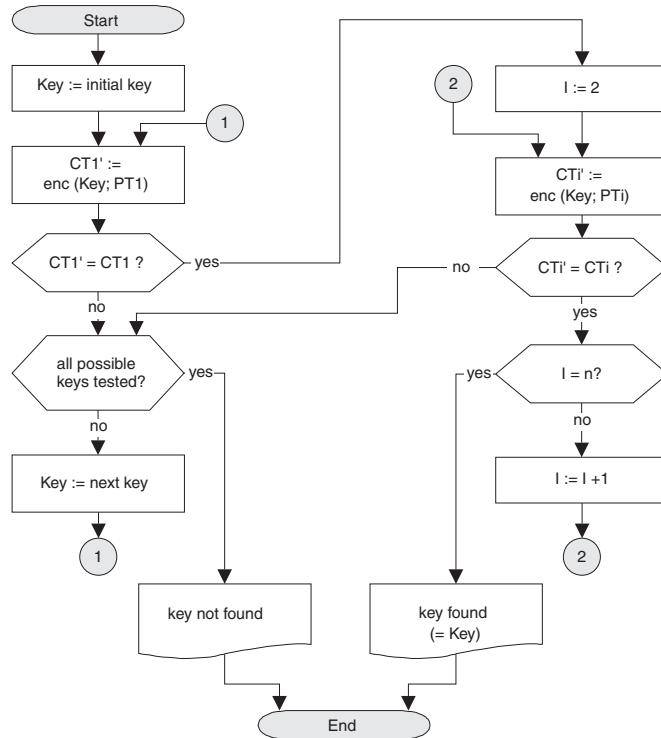
Any of these attacks could, if successful, bypass authentication of the terminal by the smart card. Without exception, they can only succeed with the first generations of smart cards. They will all fail with modern operating systems. The cycle length of current random number generators is so large that the same random number never appears twice in the lifetime of an individual smart card. Trying to generate so many random numbers that problems start to occur with the EEPROM or flash memory is also no longer feasible. If this happens, random number generation is simply blocked and any further authentication attempts are prevented. Modern random number generator designs in smart cards also allow the operating system to check the quality of their random numbers during start-up. If the required quality is not achieved, the random number generator or even the entire operating system is deactivated. This is required by standards such as FIPS 140-2.

A high-quality random number generator must fulfill some additional requirements, such as producing unpredictable random numbers and having a long cycle length (the number of values generated before the generator repeats itself). In addition, all smart cards used in a particular application must generate different random numbers. This may sound extremely obvious, but problems have repeatedly occurred in the past in this regard. This is achieved by storing different initial values for the pseudorandom number generator when the smart cards are initialized or personalized. The initial value is often called a seed number in allusion to a seed that develops into a plant. The design and evaluation criteria for pseudorandom number generators and methods for measuring the quality of random numbers are described extensively in Section 7.3, ‘Random Numbers’, on page 159.

### Attack and defense: viruses and Trojan horses

Until recently, computer viruses were entirely unknown in the smart card realm because there was no provision for downloading program code in the card usage phase. Modern smart card operating systems, however, allow program code to be downloaded to smart cards after they have been issued to cardholders and then executed. This means that in principle, the conditions necessary for computer viruses in smart cards have been created. By definition, a computer virus is a program that can reproduce itself and thus spread to other computers. Otherwise it is called a Trojan horse. Both types of program have in common that under certain conditions they can perform unauthorized actions in the host computer. With a smart card, this could involve reading and outputting the values of secret keys.

Unlike the situation with normal PCs, it is not a straightforward task to load a program into the memory of a smart card and then execute it. There are security mechanisms in the card that prevent programs from being run without authorization. For example, some applications may require prior authentication of the terminal. In addition, the code to be downloaded usually has be protected with at least a MAC or a digital signature. Some smart card operating systems have software or hardware mechanisms that isolate the memory regions used by individual



**Figure 16.46** A basic procedure for performing an exhaustive key search with a given cryptographic algorithm and several plaintext–ciphertext pairs, according to a proposal by James Massey [Massey 97]. The following abbreviations are used: CT: ciphertext; PT: plaintext; CT<sub>i</sub>, PT<sub>i</sub>: the *i*th plaintext–ciphertext pair; *n*: number of plaintext–ciphertext pairs

applications, so the applications in the smart card cannot influence each other. Due to these strong security measures, within the foreseeable future it is unlikely that any computer viruses or Trojan horses that are unintentionally downloaded when the card is already in use will be able to impair the functions or security of any applications in the card.

### Attack and defense: exhaustive key search

One form of attack at the cryptographic level is an exhaustive key search (see Figure 16.46). For this, the attacker needs at least one plaintext–ciphertext pair, and preferably several pairs, and of course the appropriate cryptographic algorithm. The attacker then encrypts the available plaintext using each possible key in turn until the available ciphertext is obtained. This key can then be tested with all other available plaintext–ciphertext pairs. If correct encryption can be performed in each case, the key that has been found is most likely the correct key. This method is essentially suitable for all encryption algorithms, although it is not always the best way to determine a secret key.

As early as 1993, Michael Wiener published plans for a special computer with an estimated cost of one million dollars that could test all DSS keys for a given plaintext–ciphertext pair within seven hours [Wiener 93]. This means that on average a 56-bit DES key could be determined in 3.5 hours. The DES Challenge III issued by RSA Inc. [RSA] was solved in

1999 in 22 hours and 15 minutes using computers networked via the Internet and a special computer called Deep Crack. The problem was to decrypt a message that had been encrypted using a 56-bit key. It was solved using a brute-force attack in which 22 % of the 56-bit key space was searched. The average computation rate was approximately 199 billion decryptions per second. Similar attacks on the RSA algorithm have also been carried out. For instance, the RSA Challenge, which was also issued by RSA Inc. and required the factorization of a 640-bit encryption, was solved in November 2005.

In practice, several different approaches are used to counter such attacks. The simplest and best known measure is to make the key space of the cryptographic algorithm so large that it is not possible to perform a systematic search within an acceptable length of time, even with very high computing power. This is why the DES algorithm has now been generally replaced by triple DES, since the key space of DES has become too small relative to currently available computing power.

Another defense can be achieved very easily by designing the application protocol to prevent the occurrence of plaintext–ciphertext pairs. Particularly with smart card applications, data encryption is not even necessary if the data is protected with a MAC. As several different text blocks can map into the same MAC, a brute-force attack on a MAC requires considerably more effort than a brute-force attack using a plaintext–ciphertext pair.

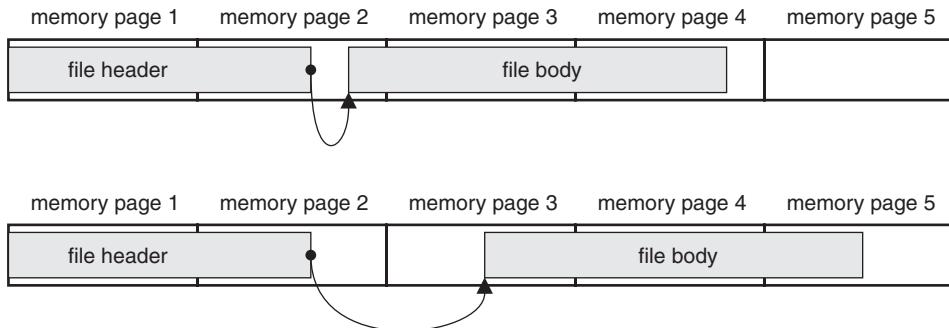
If a random number is prefixed to the plaintext in the smart card (which is called salting the data) before the data to be transferred is encrypted or a MAC is computed from the data, the result will be different each time the process is performed due to the difference in the input data. This also makes an exhaustive search more difficult, since in many cases the random number does not have to be disclosed. For example, it could be a secret shared by the security module and the smart card. Incidentally, prefixing a random number to the data to be encrypted in the smart card also provides very good protection against attacks using differential fault analysis (DFA) or power analysis (SPA/DPA), even if the random number is public.

Things can also be made more difficult for the attacker by using dynamic keys (session keys), which are different for each encryption operation. Even if an attacker manages to determine the value of the key by some happy accident, this will not be of any use to the attacker because a different key will be used in the next transaction.

### Attack and defense: memory reuse

Smart cards that are used intensively for several applications require extensive management of the nonvolatile memory, since files are generated, modified, and subsequently deleted in the various application scenarios. This forms the basis for a form of attack that has been used successfully for many decades in Unix systems and the PC world. For this purpose, the attacker creates a file with the largest possible size and then reads its contents immediately. The attacker hopes that the new file will occupy physical storage space previously used by another application and that the operating system will not have erased this region. In this case, the data previously stored in this region can be read, and in some cases valuable information can be derived from this data.

The countermeasure here is relatively simple. The operating system only has to physically erase all storage space released by applications, which means setting it to a particular value or writing it with random numbers. To be absolutely safe, this operation can also be performed immediately after new storage space is allocated. This ensures that unused storage space is always fully erased, even if an unexpected or intentionally induced shutdown occurs after it is released.



**Figure 16.47** Layout of a file header and file body in the nonvolatile memory of a smart card. With the upper arrangement, the file header can be manipulated by inducing an intentional power interruption during a write operation on the file body. This is not possible if the file elements are located on separate memory pages as shown in the lower arrangement

This basically simple defense measure is often overlooked, and sometimes it falls victim to measures taken into boost performance. However, it is essential in a secure operating system.

As an alternative to the previously described defense method, all files can be encrypted individually by the operating system. With this approach, all read and write operations can still be performed transparently from the application layer perspective, but it is not necessary to erase the contents of files that are no longer needed; only the associated key needs to be erased. With a sufficiently strong encryption algorithm, such as AES, it is impossible to reconstruct the contents of the storage area concerned after the key has been erased.

#### Attack and defense: file layout on memory pages

EEPROM and flash memory are always organized in pages. With an unfavorable arrangement of the file header and file body, this can form the basis for a severe form of attack that exploits the fact that, for memory optimization reasons, some file management systems place the file header and the file body on the same page, as illustrated in Figure 16.47.

If a write operation to the file body is terminated abruptly by interrupting the supply of power, the data in the file body as well as the data in the header on the same memory page will be in an undefined state. In the worst case, this can result in the access conditions stored in the file header being set to ‘always’ or cause corruption of the pointer structure so the header is associated with a different file body. The specific effects depend on the structure of the file header, the overlaps of the memory pages, and the data contents of the memory pages after the interrupted write operation. The results of this form of attack are effectively unpredictable, but under unfavorable conditions they can certainly have an adverse effect on security.

Atomic operations are a general form of defense against this attack, but it is undesirable to make all operations atomic because this would considerably increase the processing time for all write operations. An alternative way to prevent this form of attack is to protect the file header with a checksum. This enables the operating system to recognize inconsistencies in the header before the access occurs and to take appropriate action, which usually consists of deactivating the file. Another measure is to organize the arrangement of the file headers and

file bodies in nonvolatile memory so that the header and body of a particular file are never on the same page. This simple measure costs somewhat more memory space, but it reliably prevents any form of interaction between these two parts of a file.

### 16.5.3 Attacks on applications

The protective mechanisms of an application are based on suitable mechanisms in the hardware and the operating system. The application is dependent on having these two lower levels fully meet their obligations with regard to protection, since it cannot correct for any errors in the hardware or the operating system. For example, if it is possible to read the contents of the memory by using an analysis method, even the most complicated and secure encryption processes are of no use because the keys can be taken directly from memory by an attacker. Each application must nevertheless be constructed such that a successful attack on an individual card does not cause the entire system to be compromised.

#### Protection: simple mechanisms

To provide effective protection against attacks, all mechanisms of an application should be designed to be as uncomplicated as possible, and they should always conform to the generally applicable principle of ‘keep it as simple as possible’. In addition to simplifying implementation, this makes it easier to test the mechanisms and verify that the protective measures are effective. It is extremely dangerous to assume that protection against all possible forms of attack can be provided by simply making something sufficiently complicated. As a rule, exactly the opposite is true. A common consequence of using complicated processes and mechanisms is that things are forgotten or overlooked, which makes the attacker’s task that much easier.

The protective mechanisms provided by the operating system should always be used by the application. They are tested, are reliable, and provide protection at relatively low software levels. This does not mean that applications do not need to have their own protective mechanisms, but the mechanisms already present in the operating system should always be used.

#### Protection: conservative access privileges

In addition to the ‘keep it simple’ rule, there is a second generally applicable rule. This rule says that access privileges for the files and commands of a smart card should be granted as conservatively as possible. Access should generally be prohibited and only allowed when it is absolutely necessary.

The advantages of this approach are that it reduces the chance that access to important data and commands will be granted unintentionally and it increases the amount of work required to obtain each piece of necessary information. This can considerably reduce the attractiveness of an attack, since it increases the required effort and expense.

#### Protection: state machines for command sequences

Attacking a smart card application is considerably more difficult if it is not possible to execute every command at any desired time or an unlimited number of times. This can be implemented by using a state machine to govern the allowed command sequence. For example, if mutual

authentication of the terminal and the smart card is specified as the first required action, an attacker will have to overcome this protective barrier before executing any further commands.

### **Protection: double access security**

The attacker's job is made considerably more difficult if the smart card files are protected not only by access conditions stored in the objects, but also by using a state machine to determine the allowed commands and associated parameters. With this approach, an attacker cannot discover the specific features of the system by simply trying each command or combination of commands in turn. If command sequences are monitored by a state machine, only the commands defined in the application can be executed in the smart card. Execution of all other commands will be blocked by the state machine. This considerably reduces the scope of opportunity available to an attacker by means of command manipulation.

### **Protection: various test levels**

It has been standard practice for many decades to support various levels of testing for banknotes. This consists of security features that can be checked independently by different groups of people or different types of machines. For instance, many of the visual features, such as security threads and watermarks, can be checked by anyone on the street. For testing at the next level, an ultraviolet lamp is needed to recognize the fluorescent pigments in the paper. The features at the next higher level are used by automated equipment to verify that the notes are genuine. A typical example is the infrared properties of the banknote. Yet other level of independent features is available for the tests performed by the central banks.

This concept can easily be extended to smart cards, with the logical consequence that not every entity can check all the features. For example, a retail terminal for an electronic purse system may hold only some of the keys used for signature verification, rather than all of them. This will not weaken the cryptographic security of the system, and it has the advantage that an attacker cannot compromise the entire system by learning the master key of a retail terminal. Only the system operator will know all the keys necessary for a complete transaction data set, so the system operator will always be able to recognize an attack from the forged signatures and will be able to initiate suitable countermeasures.

### **Protection: security features**

Features incorporated in the microcontroller can provide additional operational security for smart cards. These features consist of supplementary functional units that can be tested by the terminal along with the software in the chip. Both analog and digital components are used for this purpose. The security of these features is based on concealment, and is different for each application, which makes the chips application-specific.

### **Protection: secure messaging**

Transferring data in an insecure environment entails certain risks. Using relatively simple technical manipulation of the interface between the terminal and the card, it is possible to insert or delete almost any desired data in the normal data stream during a session. If this happens while data relevant to security is being transferred, an attacker could derive a benefit from such manipulations. Secure messaging methods can be used to prevent this relatively simple and easily executed form of attack. However, full encryption of all of transferred data should generally be avoided, with encryption being reserved for transferring secret keys and similar data. One reason for not encrypting all of the data relates to data privacy legislation. Almost all

data that is written to the memory of a smart card is public. If this data is encrypted, nobody can check what is actually written to or read from the card. To forestall any suspicions regarding the encrypted data, which in principle would be justified, encryption of the transferred data should be avoided as much as possible.

### Protection: error recovery functions

If a session is terminated prematurely for an undefined reason or there are fundamental questions regarding an earlier session, it is a major benefit to have application-specific log files in the smart card. These files are maintained by the operating system, which updates them regularly during the session to reflect the current status of the application and any signatures or other data that may be received from the terminal. The logged data is held in a cyclic file in which the oldest record is always overwritten each time a new entry is made, causing the content of the oldest record to be lost. For example, if a log file holds twenty records, information regarding the most recent twenty sessions can be stored for subsequent analysis of the session history. This information can be used to resolve many questions and unambiguously clarify disputed transactions and processes.

Another reason for maintaining detailed log files in the smart card is that they facilitate certain error recovery functions. With a log file, it is possible to automatically restore the previous state of the card (roll back the card) if a session is terminated in an undefined manner. This would otherwise require analyzing the exact process and sequence of events, which might even require human intervention.

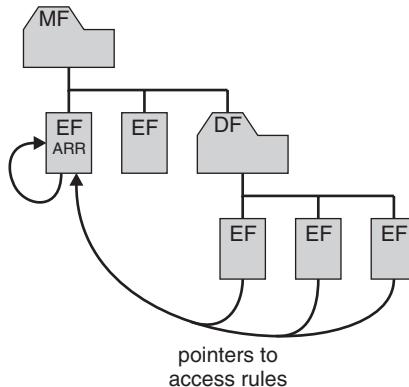
### Protection: authentication

Unilateral authentication, which is well known due to its use with magnetic-stripe cards, ultimately amounts to nothing more than verification by the terminal that the card is genuine. Due to its passive nature, a magnetic-stripe card cannot verify that the terminal is genuine. The introduction of smart cards has fundamentally changed this situation. Now the card can also check whether it has been inserted into a genuine terminal or is connected to a genuine background system. This has far-reaching consequences in terms of security, since it enables the card to take active measures against unauthorized access attempts.

Numerous possibilities arise from the ability of the card and the terminal to perform mutual authentication, although they are usually not exploited to anywhere near their full potential. A smart card should at minimum refuse to allow any form of attempted access if the terminal cannot be properly authenticated. This would make it impossible to carry out any sort of analysis of the smart card operating system in private, even if only to find out what commands are present.

### Protection: online behavior

Terminals with integrated security modules can operate fully independently in smart card applications. Of course, periodic uploads and downloads to and from a background system are still necessary, but they usually occur only infrequently. With relatively large applications having a large number of cards in circulation, it must at least be possible for the terminal to quickly make a connection to the background system if necessary, in order to provide direct end-to-end communication between a smart card and the background system. The importance of this increases with the size of the system and the extent of the benefit an attacker can obtain by means of fraud. This is because a direct communication link to the smart card allows the



**Figure 16.48** All file access privileges are stored in a central location in  $EF_{ARR}$ . Consequently, it must have special protection and must never be exposed to manipulation

background system to access the current database and block the card if necessary. In addition, the keys held in the background system are significantly more secure than the keys held in the many terminals in the field, even if the terminals have security modules. The background system can also generate very good statistical assessments of various card data it receives via sporadic end-to-end links to the smart cards.

All of these reasons are naturally particularly relevant to electronic purse systems based on smart cards. The compulsion to go online can be triggered in the smart card by time limits or randomly. An equally effective approach is to use a counter in the card to demand an online connection with mutual authentication after a certain number of offline transactions have occurred or when the value of an offline transaction exceeds a certain amount. At the end of the session, the background system can reset this counter or modify the parameters that control the online behavior of the card.

### Attack and defense: manipulation of $EF_{ARR}$

The  $EF_{ARR}$  (access rule reference) file plays a key role in systems that use rule-based access conditions<sup>13</sup> because it contains a set of records that hold all of the access conditions. As shown in Figure 16.48, the files concerned have pointers that reference the corresponding records in  $EF_{ARR}$  according to the allowed type of access.

Consequently, potential attackers focus on  $EF_{ARR}$  because the consequences for the associated files would be enormous if the data in this file could be altered. Accordingly, it must never be possible to delete  $EF_{ARR}$ , as otherwise the access conditions for an entire set of files would be undefined. However, a robust smart card operating system would notice the absence of  $EF_{ARR}$  and would respond by blocking all accesses, with the result that this form of attack would be ineffective. Nevertheless, particular care must always be taken in application design to ensure that  $EF_{ARR}$  is well protected. This applies to the write privileges for  $EF_{ARR}$  and the deletion and creating privileges for this file. If it could be deleted and created anew, a new version of  $EF_{ARR}$  that allows read access to all associated files could be generated. Naturally, this must be reliably prevented.

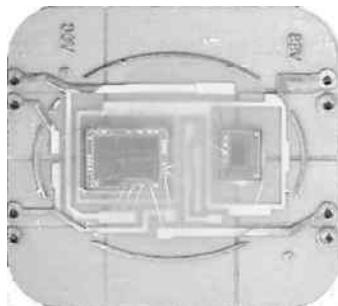
<sup>13</sup> See also Section 13.9, ‘ISO/IEC 7816-9 Resource Access’, on page 474

### 16.5.4 Attacks on the system

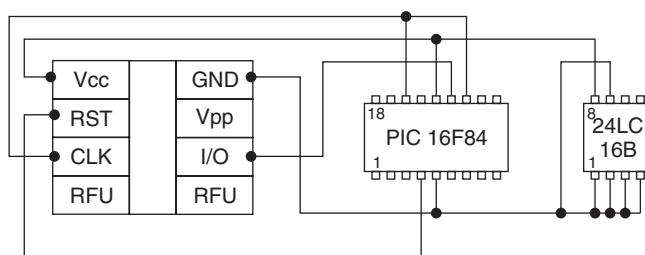
#### Attack and defense: dummy smart cards

Probably the simplest imaginable form of attack is to use a smart card that has been programmed by the user and includes supplementary analysis and logging functions. Until a few years ago, this was practically impossible because only a few companies were able to procure smart cards and the associated microcontrollers.

Nowadays, though, smart cards and configuration programs are freely available from a variety of companies. This naturally enlarges the options available to an attacker. Even without this, with a certain amount of effort and skill it is possible to assemble a working smart card by combining a plastic card with a standard microcontroller in an SMD package (see Figures 16.49 and 16.50). Such a card can at least be made to emulate the electrical interface of a real smart card and behave the same way for data transfers. Cards of this sort can now be obtained from a wide variety of dealers via the Internet. New possibilities are also provided by Java



**Figure 16.49** Rear view of an opened smart card module. The chip on the left is a standard PIC microcontroller, which is connected to an EEPROM memory chip on the right by bonding wires and tracks. This type of chip module is typically used in cloned smart cards and other forms of attack on smart card systems



**Figure 16.50** A typical equivalent circuit for a smart card microcontroller, built from standard discrete components (a PIC 16F84 microcontroller and a 24LC16B EEPROM memory chip). These components fit in a typical smart card module, so it is not possible to detect any difference from a genuine smart card microcontroller without investigating the module. This circuit and variations on it can be found on relevant Internet sites. Figure 16.49 shows the circuit construction in an opened chip module

technology for smart cards, which makes it easy for users to generate their own programs and load them into dummy cards.

With such a dummy card, it is possible to log at least part of the communication traffic with a terminal and subsequently analyze this information. After several attempts, it would probably be possible to perform part of the communication process in exactly the same way as a genuine smart card.

It is doubtful that any advantage could be obtained from this, since all professionally designed applications employ cryptographic protection for important activities. As long as the secret key is not known, an attack will not proceed any further than the first authentication attempt. Such an attack can only be successful if the secret key is known or the entire application operates without any cryptographic protection. If such an application exists, it is highly doubtful that any benefit that could be obtained from this form of attack would be large enough to justify the necessary effort and expense.

### **Protection: blacklists**

It is impossible to fully eliminate the possibility of counterfeit smart cards in a system, no matter how well the cards may be protected against attacks. To protect users as well as the system, a smart card system must also incorporate effective mechanisms to block stolen cards throughout the system. The methods used for this purpose are strongly dependent on the application concerned and the system design, but they are all based on a few fundamental techniques.

To prevent the use of fraudulent or lost smart cards, it is necessary to maintain lists that identify valid cards or invalid cards by means of some unique feature. This feature is usually a number, such as the card number. In terms of impeccable system design, which stipulates that everything that is not explicitly permitted is implicitly prohibited, a list of valid cards is the best solution. However, in a large system such a ‘whitelist’ would be awkwardly large and would require very frequent updating. This can be easily illustrated by noting that in a system with 10 million smart cards and an 8-byte card number, a whitelist would hold 80 MB of data.

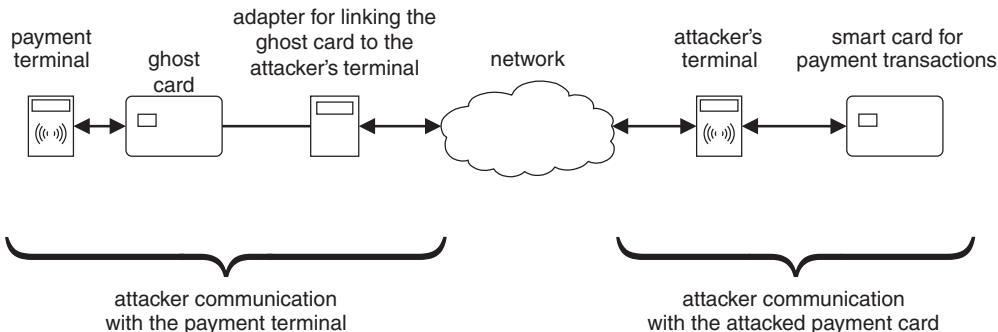
This is why lists of all blocked cards, which are called blacklists, are used in practice. In the example just mentioned, the size of the blacklist is only 800 KB if the number of blocked cards is 1 % of the total. However, if it is necessary to block significantly more than 1 % of the cards in the system due to attacks or lost cards, the size of the list will quickly become impractical even with this approach.

To further reduce the number and volume of data transfers between the entity that maintains the list and the entities that check cards against the list, ‘redlists’ are sometimes used. A redlist is a list of cards that are demonstrably fraudulent and should be confiscated immediately or at least blocked for all further transactions. Even in large systems, the number of entries in such a list lies in the two- or three-figure range.

Smart cards can be checked against these lists in real time in systems that operate online. In systems that operate partially or fully offline, updated blacklists and redlists must be sent to the terminals as often as possible. This should occur at least daily, as otherwise a protection mechanism based on blacklists will be ineffective.

### **Attack and defense: destruction of contactless smart cards by electromagnetic pulses**

A device for the ‘permanent deactivation’ of RFID tags and contactless smart cards was presented at the 22nd Chaos Computer Congress (22C3) in December 2005. In the simplest form, devices of this sort, which are now known as RFID zappers, consist of a remodeled



**Figure 16.51** Basic operating mechanism of a relay attack on a contactless payment card

photographic flash unit with a coil instead of a flash tube. When the unit is triggered, it generates an electromagnetic pulse (EMP) by discharging the flash capacitor through the coil. This pulse is strong enough to destroy wireless RFID tags or contactless smart cards by inducing a high voltage in their antennas. This is essentially a form of denial of service (DoS) attack, which in this case is irreversible.

As there are no technical obstacles to building an RFID zapper with significantly larger range, this form of attack can certainly pose a threat to a system. Particularly in local public transport systems with their typical widespread use of contactless cards, this form of attack could cause considerable problems for travelers.

The only form of defense against this attack is to incorporate improved protection circuitry at the semiconductor component inputs of the chip hardware. This can be implemented at reasonable effort and expense. A metal enclosure around the contactless card would also provide a simple form of protection, but this ‘Faraday cage’ would have to be removed each time the card was used, which would doubtless have a negative effect on user acceptance.

### Attack and defense: relay attack

A form of attack called a relay attack is possible with contactless smart cards.<sup>14</sup> In its simplest form, this attack consists of extending the communication range of a contactless smart card to several times the usual distance, as illustrated in Figure 16.51. In practice, this form of attack may be carried out by using a terminal located several meters away from a contactless smart card to establish a connection with the card without the knowledge of the cardholder and then use the card to make a payment.

This form of attack can succeed because current contactless smart cards do not have any confirmation button and only have to be held near a terminal to make a payment. An attacker equipped with a special terminal that can communicate over an extended range of several meters can exploit this property by standing in the vicinity of the target card. With this approach, the attacker uses a dummy smart card, called a ghost card, to effectively pay with a genuine smart card located a certain distance away.

This form of attack can be extended by carrying out the communication with the genuine card via a separate channel instead of simply enlarging the communication range of the

<sup>14</sup> Good overviews are provided in documents by Gerhard Hancke [Hancke 05], Thomas S. Heydt-Benjamin *et al.* [Heydt-Benjamin 06], and Ziv Kfir *et al.* [Kfir 05].

terminal. In practice, such an attack could be performed as follows. The first of a pair of attackers establishes contact with a contactless smart card located in one of the lockers of a swimming pool or fitness studio. After detecting a suitable contactless smart card, the first attacker uses a mobile telephone to connect it to the second attacker, who uses a ghost card to go shopping at almost any desired location, regardless of the distance from the genuine card. The communication link between the payment terminal and the genuine smart card is extended by the mobile telephone network, and the process is the same as with a genuine transaction.

It is not possible to use cryptographic methods to defend against this form of attack. Other approaches must be used instead. The ideal solution would be a button on the contactless card that must be pressed to confirm each transaction. Alternatively, the contactless smart card could be held in a screened wallet. Both approaches provide reliable protection against surreptitious communication with the contactless smart card. A conceivable supplementary defense mechanism would be to have the payment terminal analyze the timing of each transaction in order to detect the time delay resulting from an extended communication link. However, the reliability of this measure is strongly dependent on the quality of the extension channel, which is constantly improving with ongoing technical progress.

# 17

## Smart Card Terminals

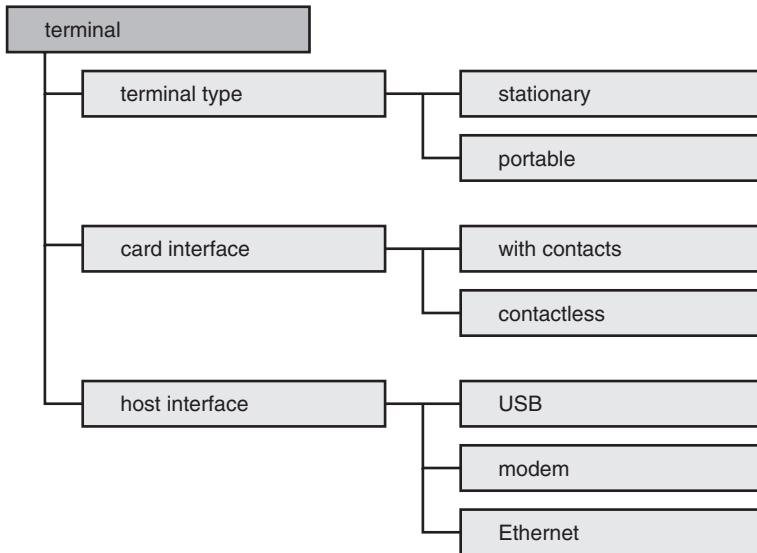
In most cases the only link between a smart card and the outside world is the serial interface. There is no other way in which data can be exchanged, so an additional device that provides electrical connections to the card is necessary. In this book, such a device is always called a terminal. However, other designations are also used, such as interface device (IFD), chip accepting device (CAD), chip card reader (CCR), smart card reader, and smart card adapter. All such devices provide the same basic functions, which are to supply power to the card and establish a data link. The terms ‘card reader’ and ‘smart card reader’ should not be understood to mean that these devices can only read data from the card. Write operations are naturally also possible. These terms were presumably copied from magnetic-card readers, most of which can only read data.

Any terminal consisting of more than just a contact unit, a voltage converter and a clock generator always has its own processor (with an 8-, 16- or 32-bit architecture) and associated memory. In simple terminals, the processor can be part of a microcontroller, but it is often a component of a single-board computer. Terminals are usually programmed only by terminal manufacturers using the C, C++, or Java programming language.

The difficulties associated with terminal programming by external parties are the same as with smart cards that support downloadable program code, so the solutions will probably develop in the same direction. The Europay Open Terminal Architecture (OTA) with a Forth interpreter, which was published in 1996, was one of the first attempts at such a solution, and Java for terminals is the latest candidate. The EMV specification also explicitly includes a concept for downloadable program code.

Terminals do not have hard disk drives, so they must store their programs and data in battery-backed RAM, EEPROM, or flash memory. The amount of available memory is usually around a few megabytes.

In contrast to smart cards, which all have similar structures, there are many different terminal architectures, as indicated in Figure 17.1 and Table 17.1. Terminals can be divided into two basic types: portable and fixed. Portable terminals are battery-powered, while fixed terminals are preferably powered from the AC mains or the data interface. Terminals can also be classified by their user interfaces. Portable devices in particular often have a display and simple keypad so their main functions can be used on site. Figure 17.2 depicts the logical architecture of a typical terminal with full functionality.



**Figure 17.1** The principal characteristics of smart card terminals

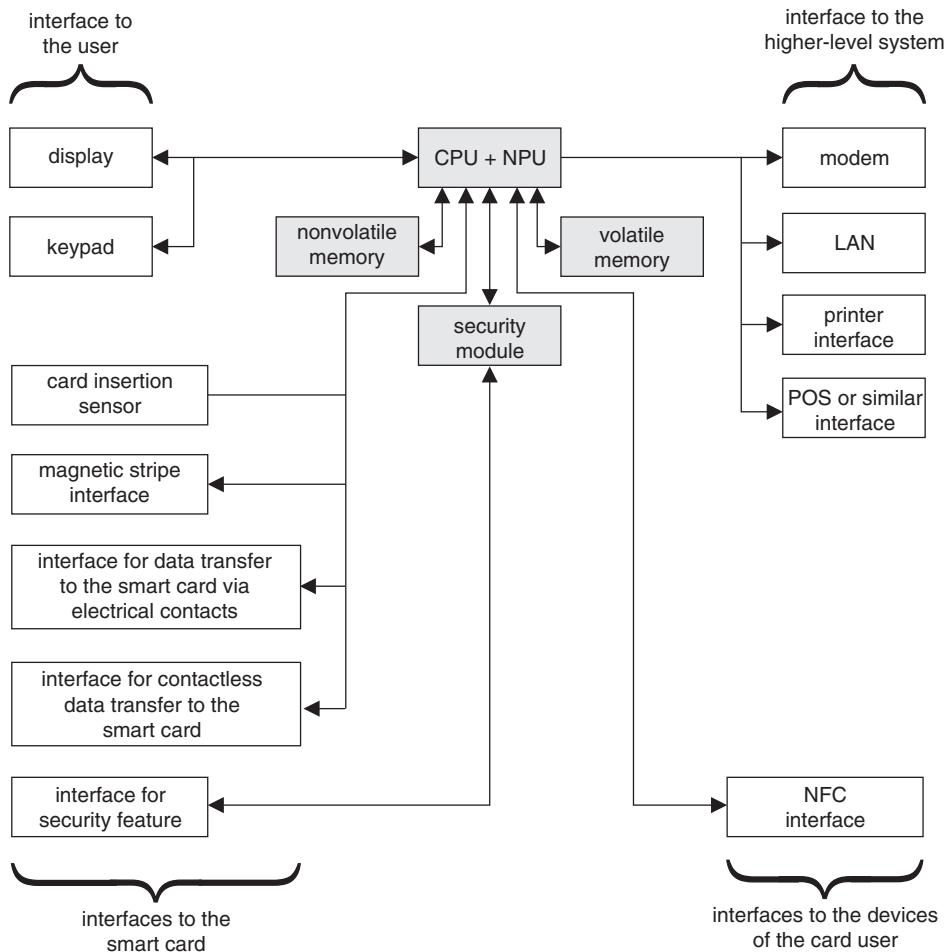
**Table 17.1** Terminal classes according to the German ZKA

Class	Functional elements
1	Contact unit and interface to other systems
2	Class 1 functional elements + keypad
3	Class 2 functional elements + display
4	Class 3 functional elements + security module

In addition to a display and a keypad, fixed terminals often have permanent links to higher-level computer systems. A terminal without a man-machine interface (no display or keypad) must have a direct connection to a computer in order to provide a link between the smart card and the user.

There is a practically oriented general classification of terminals in one of the German ZKA specifications, which divides terminals into four classes. A Class 1 terminal essentially consists of a contact unit without any supplementary functional elements, along with an interface to another system (such as USB). Class 2 includes the capabilities of Class 1 with the addition of a display. Class 3 has a keypad in addition to the elements of Class 2. Class 4 is the most elaborate, with all of the functional elements of Class 3 as well as a hardware security module (HSM) with RSA capability.

Classification into portable and fixed terminals leads to a further distinguishing feature, which is how the terminal is used. An online terminal has an uninterrupted link to a remote computer during operation, and the remote computer assumes part of the control function. A typical example is a terminal for physical access control, which is entirely controlled by a background system to which it is permanently connected.



**Figure 17.2** Typical architecture of a smart card terminal with a display, keypad, magnetic-stripe reader, and security module. Such terminals are often used at point-of-sale locations to allow payments to be made using a variety of cards (credit cards, debit cards, and electronic purses). A keypad that is specially protected against manipulation (a PIN pad) can be used if necessary. This diagram shows the basic energy and data flows and is not a schematic diagram

Offline terminals, by contrast, are fully independent of any higher-level system in operation. Although there are very many types of online terminals, pure offline devices are very rare. Offline terminals at least occasionally exchange data with a background system, if only to request a new copy of the blacklist or an updated version of the terminal software.

In typical applications inside a building, the physical link between the terminal and the remote computer is provided by an electrical cable or a fiber-optic cable. However, the link can also be provided by a telephone connection to the nearest computer center, as is the case with point-of-sale terminals for electronic payments. This may a dial-up link or a permanent link (leased line), depending on the application. Leased lines incur higher operating costs, so some of these terminals establish a connection to the higher-level system only as necessary.



**Figure 17.3** A smart card terminal in the form of a USB plug for use with cards in plug-in format

Smart card terminals in plug-in PC Card format (formerly called PCMCIA) do not readily fit into the above classification scheme. They can be used both online and offline, and with desktop as well as notebook computers. Such terminals are nothing more than simple and usually inexpensive hardware interfaces between the smart card and the computer. The only prerequisite for using a PC Card terminal is the availability of a PC Card slot, which must be either a type I slot (3.3 mm) or a type II slot (5 mm), depending on the manufacturer. Some PC Card smart card terminals also include expansion memory for the smart card and coprocessor components for the encryption and decryption of large data volumes. These terminals, which are only a few millimeters thick, are certainly the most versatile in use, and they can open up entirely new application areas for smart cards. With such terminals, smart cards can be used with standard PCs and standard software without any need for additional cables, power supplies, or external hardware. The range of potential applications is very wide. It includes access control for specific PC functions, software copy protection, and secure e-mail with digital signatures.

Naturally, the USB interface now found in every PC offers an alternative to conventional terminals (see Figure 17.3). This interface has practically eliminated all other serial and parallel interfaces. In addition to simplified configuration, the main advantage of USB is that it can supply enough power to operate even relatively complex devices.

Many years of R&D activity lie between the first two-chip smart cards and modern cards with very powerful microcontrollers. Terminals have experienced a similar technological evolution during this period. Due to lack of experience, the electrical and mechanical construction of the first terminals was often very primitive. This frequently resulted in damage to the microcontrollers in the smart cards and premature failures. Terminal manufacturers have now overcome these teething troubles, and a development stage has been reached in which the technical specifications of terminals, which are generally the same everywhere, are less important in purchasing decisions than the external appearance of the terminal.

In functional terms, a smart card terminal consists of two components: a contact unit for the card and a terminal computer. The card reader, into which the smart card is inserted and then connected electrically to the rest of the terminal, essentially has only a mechanical function. The terminal computer is needed to drive the contact unit, manage the user interface, and establish a link to a higher-level system. In the simplest case, it can be a single microcontroller, while in more sophisticated solutions it is a single-board computer.

Contactless communication is becoming an increasingly significant alternative to smart cards with contact interfaces. Naturally, the corresponding terminals must support this form of

communication. In contrast to smart cards, terminals are often designed to support more than one card interface protocol. There are terminals available that support several of the usual contactless transmission protocols, such as ISO/IEC 14443<sup>1</sup> Type A and Type B, ISO/IEC 15693,<sup>2</sup> FeliCa,<sup>3</sup> and Mifare.

## 17.1 MECHANICAL PROPERTIES

In mechanical terms, two things must happen when a smart card is inserted in a terminal. First, the card contacts must be electrically connected to the terminal computer. This is the task of the contact unit. Second, the terminal must detect the fact that a card has been inserted. This can be handled by a microswitch or an optical sensor (light barrier). One drawback of an optical sensor is that its reliability can be affected by dirt or transparent card materials. A mechanical switch is generally the most reliable solution.

A wide variety of contact units and contact forms are used in terminals. The TS 51.011 specification states certain limits on applied forces and contact shapes, and almost all terminals observe these limits. According to this specification, the tips of the contact elements in the terminal should be rounded rather than pointed, with a radius of curvature of at least 0.8 mm. This largely prevents scratching of the contact surfaces on the card. In addition, contacts with rounded tips provide significantly better conditions for applying contact forces to the contact surface than contacts with pointed tips.

According to the GSM specification, the maximum force exerted on a single contact surface must never be more than 0.5 N (the EMV specification allows 0.6 N). This is intended to protect the chip located beneath the contacts, since this piece of crystalline silicon could break under greater stress.

Although the location of the contacts on the card body is internationally standardized by ISO and should thus be the same worldwide, a French national standard (AFNOR) specifies a chip location closer to the top edge of the card. Consequently, some terminals are built with two contact heads in order to support both contact locations (ISO and AFNOR). This technically complex solution is desirable in systems in which both types of cards are used. This arrangement only applies to a transition period, since ISO specifies that the AFNOR location should not be used in new designs. Several French banking applications, for example, use terminals with dual contact heads. This allows smart cards with the old AFNOR location and new cards with the ISO location to be used during the transition period.

There is an immense range of variation in the mechanical construction of terminals and the service life of the contacts, which is also strongly affected by ambient conditions such as temperature, humidity, and so on. However, a mean time between failures (MTBF) of 150 000 insertion cycles is regarded as a perfectly normal value for terminals.

### 17.1.1 Contact unit with wiping contacts

The technically simplest terminals, which are thus the least expensive, have wiping contacts in the form of leaf or disc springs. No other mechanical contact elements are present in these

<sup>1</sup> See also Section 10.9, ‘Proximity Cards (ISO/IEC 14443)’, on page 297

<sup>2</sup> See also Section 10.10, ‘Vicinity Integrated Circuit Cards’, on page 344

<sup>3</sup> See also Section 10.12, ‘FeliCa’, on page 352

simple terminals. However, in a contact unit with spring contacts the contact elements always slide over part of the card and its contact surfaces when the card is inserted and withdrawn, which produces scratch marks. In addition to being unattractive, these scratch marks are undesirable for technical reasons.

Repeated scratching of the gold-plated contact surfaces on the smart card gradually wears away the gold plating, which protects the underlying material against oxidation. The exposed metal oxidizes, which degrades the electrical connection, so the user may have to insert and remove the card several times to rub off the oxide layer and obtain a satisfactory electrical connection.

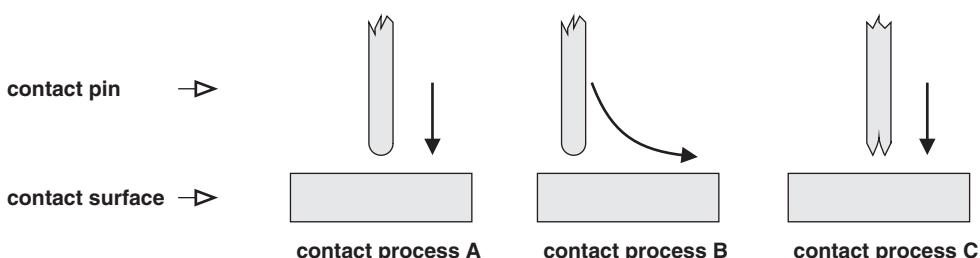
### 17.1.2 Mechanically driven contact unit

Instead of fixed wiping contacts, the next higher class of terminal has a mechanism that forces the contact unit against the contact surfaces of the card when the card is inserted in the terminal. A lever mechanism converts the smart card insertion force into a force perpendicular to the contact surface.

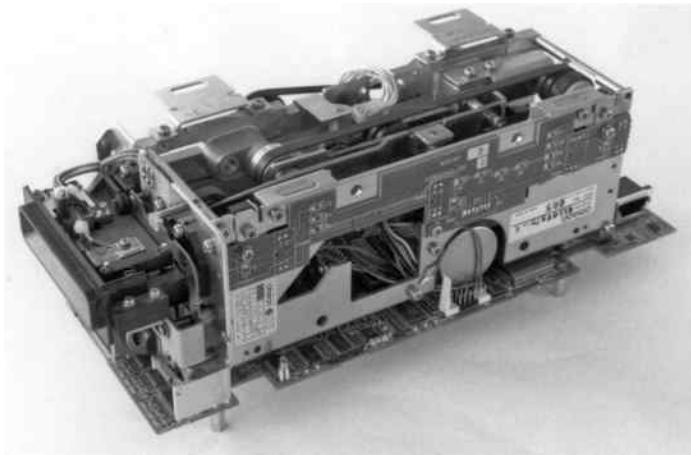
An optimally designed mechanism also produces a slight motion of the contacts along the length of the card while the contacts are being applied to the card (see Figure 17.4). This ensures reliable electrical contact with the card, since the sliding motion rubs away any light soiling of the contact surface. The individual contact elements are spring-loaded to ensure a well-defined contact force on each contact surface.

### 17.1.3 Electrically driven contact unit

The technically most complex solution, which is also the best mechanical solution, is a terminal with an electrically driven contact unit (see Figure 17.5). Here a set of contact elements in parallel guides are driven by a motor or solenoid to make perpendicular contact with the card from above, with a slight lateral motion. Due to the significant complexity of this electromechanical construction, the terminal is relatively large. However, these devices are quite suitable



**Figure 17.4** Methods for making electrical contact with a smart card. Option (a) with rounded contact elements is unfavorable because soiling of the contact surface adversely affects the reliability of the electrical connection. Options (b) and (c) provide good results with the two types of contact elements illustrated. The pointed contact pins shown in (c) slightly penetrate the contact surface, which can be seen under a microscope as small surface nicks



**Figure 17.5** A typical self-feeding reader for cash dispensers, with a shutter and a magnetic-stripe reader

for use in professional applications in which many millions of contact cycles must be carried out without maintenance. They are typically used in cash dispensers and in personalization machines employed in smart card production.

#### 17.1.4 Card ejection

Smart cards are usually inserted in a terminal manually, which means without any assistance from the terminal. Only cash dispensers have self-feeding card readers, which use a conveyor mechanism to feed the card to a contact unit inside the dispenser. Ordinary terminals do not have such devices, and they differ only in the manner in which the card is ejected. Simple terminals do not automatically eject the card, so the card must be manually removed from the reader. Two different techniques are used for this, called ‘push-push’ and ‘push-pull’. With a push-push contact unit, the card is inserted manually as usual, and it must be removed by again pushing it and then pulling it. This is ergonomically undesirable because this sequence of motions is unnatural, with the result that people often forcibly pull the card out of the terminal. This causes the contact elements to be dragged over the contact surfaces and the body of the card, since they have not yet been released by the mechanism. Push-pull contact units are a better match to customary motion sequences, since the card is simply pushed into the terminal to insert it and pulled out of the terminal to remove it.

Terminals with automatic card ejection have a spring that is tensioned by inserting the card. The terminal computer can actuate a solenoid to release the spring. This causes the card to be partially extended from the terminal, rather than fully ejected, so the user can grasp it and pull it out completely.

Card-ejecting readers have one major advantage over other types. Card ejection clearly signals the end of the session to the user and reminds the user to remove the card from the terminal. This reminder is often emphasized by an audible signal. This practical benefit is the main reason for using card-ejecting readers.

Cash dispensers in particular are usually able to retain smart cards if necessary. Since they normally have self-feeding card readers that transport the card to the contact unit, it is relatively easy to route the card to a special retention bin in the machine if necessary, instead of to the exit slot. If the terminal is large enough to hold the extra mechanism and the retention bin, card retention does not present any major problems from a technical perspective. However, it may present legal problems if the card user is also the legal owner of the card.

### 17.1.5 Ease of card withdrawal

The reliability of a system based on smart cards can suffer severely if users can withdraw their cards from the terminal at any desired time during a session. For one thing, this causes the card to be de-energized without following the specified deactivation sequence. It can also abnormally terminate read or write operations in nonvolatile memory, with the result that the content of a file is undefined. This could lead to total card failure. For these reasons, it is advantageous to use terminals with card-ejecting readers that make it impossible to pull the card out of the terminal manually.

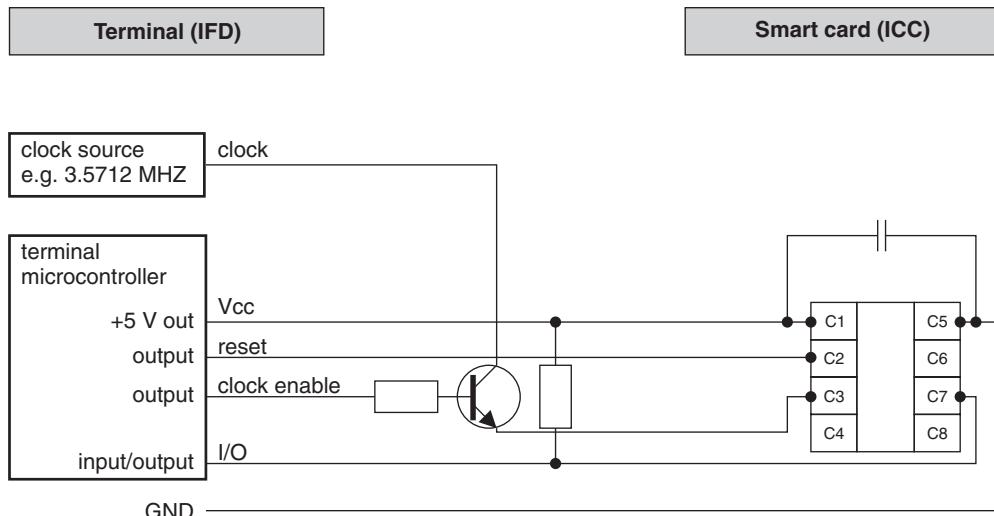
A concealed mechanical emergency ejection mechanism can be provided to allow the card to be removed from the terminal in case of power failure. However, under normal conditions the terminal can determine when to return the card to the user, thus preventing the user from disturbing ongoing processes.

## 17.2 ELECTRICAL PROPERTIES

With the exception of the contact unit, a terminal primarily consists of electronic components. They are used to provide the interfaces to the user and the background system and to control the contact mechanism. The electromechanical components of the terminal and the smart card must both be supplied with electrical signals. The only information that is provided directly by the contact unit is whether a card has been inserted. The only signal sent directly to the contact unit is the actuation of the automatic card ejector if such a device is present.

As illustrated in Figure 17.6, the card interface consists of the five contact surfaces for the supply voltage and ground, clock, reset, and data transmission. Once the electrical connections have been made by the contact unit, it is very important with regard to the service life of the smart card for the activation sequence specified by ISO/IEC 7816-3 to be followed exactly. Otherwise a high failure rate may occur due to electrical overloading of the semiconductor chip. It is equally important to observe the proper deactivation sequence, as otherwise similar problems may occur in the chip.

In this regard, there is an important aspect with simple terminals that allow the user to remove the card manually. If the contact unit detects that the card is being withdrawn, the circuitry in the terminal must immediately execute a deactivation sequence. This is the only way to avoid a situation in which energized contacts slide over the contact surfaces of the smart card, which would have very little in common with a standard deactivation sequence. The consequences of abnormal card withdrawal can be even more serious, since shorts between two leads can also occur if the contacts are worn or slightly bent. The resulting weak sparks due to the discharge of capacitors can damage the contact elements and the contact surfaces of the card.



**Figure 17.6** Typical electrical interface between a smart card and a terminal

With regard to the electrical circuitry, almost all terminal manufacturers have realized by now that short-circuit protection is indispensable. If this is neglected, a single smart card with shorted contact surfaces can cause the electrical demise of a large number of terminals. Incidentally, shorted cards crop up regularly, partly due to vandalism and partly due to technical defects.

Short-circuit protection should extend to the point that each contact can be connected to any other contact or set of contacts without any repercussions. Ideally, the circuitry that drives the smart card should be galvanically isolated from the other circuitry of the terminal. This is standard practice with public card phones in Germany, since it also largely protects the equipment against external voltages.

The smart card microcontroller uses an on-chip charge pump to generate the voltage needed for writing and erasing nonvolatile memory pages. This means that the smart card can draw currents of up to 100 mA for a few nanoseconds. The same effect, in reduced form, can be produced by transistor switching processes in CMOS integrated circuits. Even very fast voltage regulators in the power supply cannot handle these short spikes, with the result that the smart card supply voltage dips due to the heavy current load and the EEPROM write or erase operation fails. In extreme cases, the voltage dip can be so severe that the processor lands outside its stable operating area and a system crash occurs.

One remedy is to connect a capacitor as close as possible to the contacts for the smart card. A ceramic capacitor of around 100 nF is suitable, as this type of capacitor can release its charge very quickly. The leads to the smart card must be as short as possible, so that lead resistance and inductance do not significantly impair the provision of the required current in the required time. Short increases in current demand can be met by drawing charge from the capacitor until the voltage regulator can respond to the change. This is a simple and economical way to avoid power supply problems.

Particularly in electronic payment systems, it is now standard practice to equip the terminal with a real-time clock, which is needed for reasons of traceability and consumer protection.

According to the EMV specification for credit card terminals, the clock error must not exceed 1 minute per month. This is not technically difficult, since suitably accurate clock components are available as single-chip solutions. In addition, the clock can be synchronized each time the terminal makes an online connection to the higher-level system. Radio time signal receivers in terminals have so far not achieved any practical importance, since signal reception is too strongly affected by the screening effects of the site where the terminal is installed. Standard time code signals usually cannot be received inside a reinforced concrete building, for example.

### 17.3 USER INTERFACE

There are many ways to fashion the user interface of a terminal. With the simplest types of terminals, the user interface is provided by the PC to which the terminal is connected. Some terminals also have a display with a few buttons that can be used to confirm displayed messages or terminate the process. Point-of-sale terminals always have a display and a numeric keypad, as well as several special buttons, to enable PIN code entry and confirmation or cancellation of the transaction concerned. The ZKA classification of terminals (Table 17.1 on page 736) provides a good starting point for characterizing user interfaces.

### 17.4 APPLICATION INTERFACE

Technically sophisticated terminals usually have an operating system (often a Linux variant) with an extensive application programming interface (API). With such systems, applications can usually be programmed in C, C++, or Java. Relatively simple terminals generally cannot be programmed by external parties.

### 17.5 SECURITY

A wide variety of security mechanisms are used in terminals. The spectrum extends from secure mechanical enclosures to security modules and sensors for various card features. In pure online terminals, whose only function is to transfer the electrical signals between the background computer system and the smart card, there is normally no need for additional built-in security. In such cases, security is handled entirely by the computer that controls the terminal.

However, if data must be entered in the terminal or the terminal must operate independently of the higher-level system, suitable mechanisms to provide additional system security must be incorporated. The possibilities are almost unlimited, although they depend very strongly on the smart card in question and its security features.

Typical smart cards with very simple card bodies that only act as a carrier for the microcontroller usually do not have any security features on the card body. There is thus no need for the terminal to check such features. By contrast, smart cards for financial transactions are usually hybrid cards, which means they have a magnetic stripe in addition to a chip in order to maintain compatibility with older systems. Hybrid cards also have the usual card features that enable the terminal to check their genuineness independently of the chip. This means that suitable sensors must be present in the terminal.

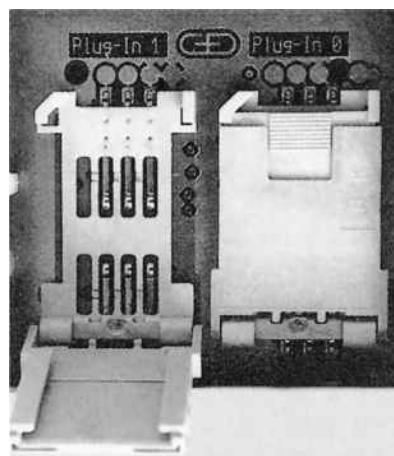
Terminals that operate offline, either entirely or occasionally, must contain master keys for the cryptographic algorithms that are used, since card-specific keys cannot be derived without these keys. These master keys are very sensitive with regard to security, since the security of the overall system is based on them. In order to ensure their security and confidentiality at all times, they are not stored in the normal electronic circuitry of the terminal, but instead in a separate security module inside the terminal that has special mechanical and electrical protection.

This security module may be a single-board computer encapsulated in epoxy resin, which can exchange data with the regular terminal computer only via an interface. The secret master keys are never allowed to leave the security module; they are used only inside the module to perform computations. In a typical application scenario, the security module receives an individual card number or chip number from the smart card via the terminal computer, and it uses this number to derive a card-specific key. This key is then used inside the security module to compute a signature or perform authentication.

Modern types of security modules, which are usually the size of a matchbox, have extensive sensor systems for detecting attacks. They are also largely independent electrically, so they can actively resist attacks, even in the absence of an external source of power. Detection of an attack usually causes all of the keys to be erased, so the attacker is left with only a circuit board encased in epoxy resin inside a metal case, with no data worth analyzing.

Due to the high cost of good security modules, the trend in recent years is to use smart cards instead. Although this results in certain limitations in terms of memory size, sensors and independence, the level of security is generally adequate, even for electronic payment applications. Smart cards in IC-000 (plug-in) format are used to minimize the physical size (see Figure 17.7).

As security modules in smart-card format are not permanently built into terminals, but can instead be exchanged, they are ideally suited to extending terminal hardware, as illustrated by the following example. Static unilateral RSA authentication will become much more important in the coming years, in part because it is stipulated in the international EMV specification for



**Figure 17.7** Two contact units for security modules in plug-in format, located side by side in a smart card terminal

smart credit cards. As RSA authentication is so compute-intensive that it cannot be performed within an acceptable length of time by the processors normally used in terminals, permanently installed security modules present a problem. However, if a plug-in smart card is used as a security module in the terminal, it can easily be exchanged. Relatively expensive smart cards with supplementary numerical coprocessors can then be used as security modules, and with suitable modifications to the terminal software they can perform RSA computations at high speed.

In addition to using a robust housing that can only be opened using special tools and incorporating security modules in terminals, a supplementary technique is often used to provide mechanical protection against unauthorized interception of data transfers to the smart card. This consists of a sort of guillotine arrangement that cuts through any wires or cables extending from a card inserted in the contact unit. The purpose of this device, which is called a shutter, is to prevent interception or manipulation of the messages between the card and the terminal. It can be actuated either electrically or simply by inserting the card. If any wire or cable cannot be cut through due to its thickness or composition, the shutter will not close completely. This is detected by the terminal electronics, with the result that no power is applied to the card, so no communication takes place.

Communication between the terminal and the smart card must always be designed such that interception or manipulation cannot impair the security of the system. This means that shutters should actually be unnecessary. However, security can certainly be improved somewhat by making things more difficult for would-be attackers. It makes a big difference whether an attacker can easily eavesdrop on data exchanges or must first overcome several hurdles. However, shutters make terminals bigger and more expensive, and very few models still close precisely after several ten-thousands of operating cycles. Consequently, the system design should not rely entirely on this sort of mechanical protection.

# 18

## Smart Cards in Payment Systems

The characteristics of smart cards make them particularly suitable for use in payment systems. They can easily and securely store data, and their convenient size and robustness make them easy for everyone to use. As smart cards can also actively perform complicated computations without being influenced by external factors, it is possible to develop totally new approaches to performing payment transactions. This is very clearly illustrated by smart card electronic purses, which are possible only with this medium.

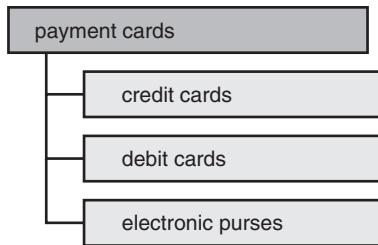
Electronic payment systems and electronic purses offer significant benefits for everyone involved. For banks and merchants, they reduce the cost associated with handling cash. Offline electronic purses largely eliminate the cost of data telecommunication for payment transactions. The risk of robbery and vandalism is reduced because electronic systems contain no cash to be stolen.

For merchants, the fact that transactions are processed more quickly is also a significant advantage, since it means that cash management can be optimized. Vending equipment can be made simpler and cheaper because modules to test coins and banknotes are not needed. Electronic money can be transferred via any desired telecommunication channel, so it is not necessary to regularly collect money from the machines.

Ultimately, the success or failure of a payment system is determined by its potential users. If the benefits for them are too marginal, they will not use the system and will choose other means of payment. After all, an electronic purse is simply a new means of payment that complements rather than replaces other existing means of payment, such as credit cards and cash. There is no reason to fear that these means of payment, which have provided reliable service for many years, will be entirely supplanted by electronic purses in the form of smart cards.

### 18.1 PAYMENT TRANSACTIONS WITH CARDS

The simplest way to use cards for payment transactions is to use magnetic-stripe cards holding data for online authorization. After the user's card has been checked against the blacklist and solvency has been verified, funds can be transferred directly from the cardholder's bank account to the merchant's account. With smart cards, the scenario is slightly different, but



**Figure 18.1** Types of payment cards

in principle it remains the same. The smart card is logically linked to a bank account, and after unilateral authentication of the card or mutual authentication of card and the background system, a previously entered sum is transferred. Naturally, PIN verification is also performed in the smart card or background system as part of the transaction.

Both of these scenarios are based on a background system that controls all of the activities. They do not by any means fully exploit the capabilities of smart cards. However, there are other means and methods of making payments that can be implemented by exploiting these capabilities. Some of them are described in this chapter.

### 18.1.1 Electronic payment transactions with smart cards

As shown in Figure 18.1, there are three fundamental models for electronic payment transactions using smart cards: (a) credit cards, in which payment is made after a service is rendered (pay later), (b) debit cards, in which payment is made when the service is rendered (pay now) and (c) electronic purses, in which payment is made before the service is rendered (pay before).<sup>1</sup> These models are described in more detail below.

#### 18.1.1.1 Credit cards

The original idea of using a plastic card to pay for goods or services comes from credit cards. The principle is simple: you pay using the card, and the corresponding amount is later debited from your account. The cost of this process is borne by the merchant, who usually pays a fee that depends on the purchase amount and is usually 2 to 5 % of the purchase price.

Most credit cards in use up to now do not have chips. The disadvantage of such cards is that they provide relatively poor protection against forgery. Consequently, card issuers experience significant losses due to counterfeit cards, since the merchant is guaranteed payment. Evidently, up to now these losses have been lower than the cost of introducing cards with chips. However, many credit cards will probably be supplemented with chips in the not too distant future in order to reduce the steadily increasing cost of fraud.

<sup>1</sup> This classification can be playfully augmented by the category ‘pay never’, which relates to fraud

### 18.1.1.2 Debit cards

Debit cards are especially widely used in Germany. A debit card, which may be a magnetic-stripe card or a smart card, allows the amount of the payment to be transferred to the account of the merchant or service provider directly during the payment transaction. With debit cards as well as credit cards, the payment transaction is normally authorized by a credit check performed by a background system. There is usually a threshold level above which this must occur, so it is not always necessary to make a connection to the background system for small purchases.

### 18.1.1.3 Electronic purses

With an electronic purse, electronic money is loaded into the card before any payment is made. This can be done in exchange for cash or using a cash-free process. When a payment transaction occurs, the balance in the card is reduced by the amount of the payment, and at the same time the balance of the electronic purse of the second party (usually the merchant) is increased by the same amount. The merchant can later submit the electronic money received in this manner to the operator of the electronic purse system and be credited with the corresponding amount of real money. The user of an electronic purse thus exchanges real money for an electronic form of money that is loaded in his or her smart card. When a payment is made, the cardholder exchanges this electronic money for goods or services.

This arrangement has three significant drawbacks for users. The first is that when the card is loaded, the user receives electronic money in exchange for real money. In economic terms, this means that the user gives the operator of the purse system an interest-free loan, since it may take several weeks for the user to actually spend the electronic money, while the real money is immediately transferred to the system operator. The amount of interest may be small for an individual user, but in total it represents a substantial source of income for the operator of the purse system.

In many field trials conducted up to now, it has been found that in industrialized countries the average amount of money in an electronic purse is around €75. The total average amount of money in an electronic purse system is called the float. Assuming that 10 million cards are issued and the interest rate is 5 %, the total annual interest on the float is €37.5 million, without any offsetting expenses. In this example, the amount of interest lost by an individual cardholder is only €3.75, which the user will not regard as a major drawback. In addition to the interest income from the float, the purse system operator receives income in the form of unspent electronic money in cards that end up in collections and defective cards that are not returned for refund.

A second drawback is that a real problem arises if the purse operator goes bankrupt. This is because the card user has exchanged real money, whose value is guaranteed by the state within certain limits, for electronic money in a smart card. If the purse operator goes bankrupt, the electronic money can suddenly become worthless, and the users will have lost their money. Consequently, efforts are being made in some countries to restrict the operation of electronic purse systems to banks and similar institutions. At minimum, lodging a security deposit with a government body is required, so that the amount of money loaded in the smart cards is covered in the event that the card issuer goes bankrupt.

There is yet a third significant drawback for users. What can the holder of an electronic purse card do if it no longer works? If the purse is anonymous, not even the purse system operator can determine the amount of money that was last loaded into the card. The purse holder will also find it practically impossible to provide convincing proof of how much money was still in the card. If the chip is entirely defective, the electronic money is thus irrevocably lost. Unfortunately, a smart card is much less robust than banknotes or coins, for understandable reasons.

In practice, this problem is usually dealt with by a compromise. Since the last amount loaded into the card online is known, as well as the purse balance at the time of this transaction, the approximate amount in the purse can be calculated. This amount is then paid to the customer. However, if a particular customer frequently makes claims due to a faulty smart card, the system operator will curb its goodwill. The risky customer is thus denied any further compensation in the hope that he or she will take better care of the smart card in the future. If the system supports shadow accounts,<sup>2</sup> the balance in the card is known to the system operator, although with a certain latency. In this case, it is possible to refund the amount of money stored in the card if the card becomes defective.

#### 18.1.1.4 Open and closed system architectures

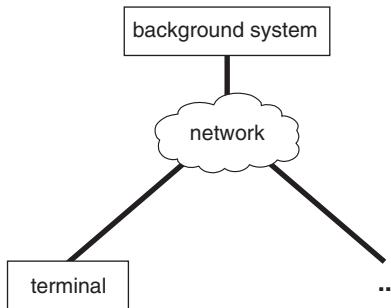
A distinction can be made between open and closed architectures with electronic payment systems. An open system is fundamentally available to multiple application providers, and it can be used for general payment transactions between various parties. In contrast, a closed system can only be used for payments in connection with a particular system operator.

The technical aspects of this can be briefly illustrated using a telephone card with a memory chip as an example. With memory cards, all that happens when a payment is made is that a counter is irreversibly decremented. The terminal does not have to keep an exact account of the number of units that have been debited; it only has to ensure that the counter in the card is always properly decremented when the service is used (that is, when a call is made using the card). In this situation, the terminal is a sort of machine for destroying units of electronic money. Of course, in practice a balance is kept for each terminal, but the debited amounts are only booked to the internal accounts of the purse system operator. Fraud in the settlement of the debited amounts between the terminal owner and the purse system operator is impossible in principle, since both parties are part of the same organization (in this case, the telephone company).

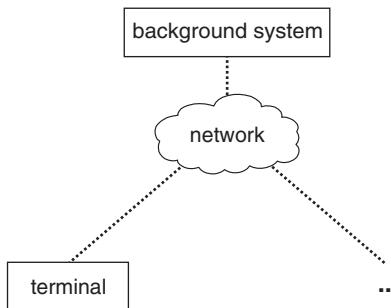
With an open system, the terminal owner and the purse system operator can be completely different companies. The purse system operator must therefore be able to verify that the terminal receipt accounts are correct and not manipulated. This must be taken into consideration from the very beginning in the system design, as otherwise account settlement between the terminal owner and purse operator will be very difficult or impossible.

In the above example using a memory card, the system concept makes it impossible for the terminal operator to convincingly guarantee the purse system operator that the claimed amount is correct. This is because the terminal operator can only present an invoice for a certain number of units, instead of fraud-proof signatures for the amounts paid, as would be possible with a genuine electronic purse system.

<sup>2</sup> See also Section 18.6.2, ‘The overall system in brief’, on page 785



**Figure 18.2** Basic architecture of a centralized system for electronic payments. The illustrated connections are permanent



**Figure 18.3** Basic architecture of a decentralized system for electronic payments. The illustrated connections can be established as needed

#### 18.1.1.5 Centralized and decentralized system architecture

The system architecture of an electronic payment system using smart cards can be either centralized or decentralized. With payment systems in particular, system security is the most important issue. There is thus frequently a tendency to use centralized systems, since this gives the system operator complete control of the system.

In concrete terms, a centralized system is an online system in which every payment transaction is performed with a direct online connection to the background system as illustrated in Figure 18.2 in contrast to a decentralized system where a connection to the background system is established only when necessary (see Figure 18.3). With a centralized system, payment is not possible if a communication link cannot be established. Nevertheless, a centralized system has certain advantages. For instance, incoming transactions can be directly compared with the current blacklist in real time. Key exchanges can be carried out directly by the background system without any delays. The software in the terminals and the general parameters in the cards can be updated directly and with little additional effort, since a direct link to the background system must be established for each transaction.

However, these advantages are offset by several major disadvantages. In many countries, telecommunication charges are so high that it is not worthwhile for merchants to have a permanent link to the background system, or to establish a dial-up link to the background system for every transaction. In addition, in some regions the telephone network is not sufficiently

**Table 18.1** Typical actions and conditions that trigger an online connection between a smart card and the background system

Trigger	Typical value
Type of transaction (e.g. paying out cash)	100 %
POS condition (e.g. PIN pad available)	—
Parameter-driven random selection	10 %
Manual request at the merchant terminal	1 %
First use of the smart card	100 %
Number of offline transactions performed since the last online transaction	10
Total offline amount since the last online transaction	€500
Time since the last online transaction	7 days
Payment amount exceeding a configurable threshold value	€200

reliable to allow an online link to a higher-level computer to be established at any desired time. Due to their active nature, smart cards are excellent for use in decentralized systems because they hold part of the system security in house. This is also their main advantage relative to passive magnetic-stripe cards, which cannot compel the system to perform specific processes.

In particular, a decentralized system is the only feasible option if electronic purses are used with automated equipment such as vending machines and ticket dispensers, since electronic purses can operate completely independently for weeks or months on end and the equipment concerned does not have any means to connect to an existing communication system. For all of these reasons, a decentralized system is often the better choice. In addition, a decentralized system has significantly better fail-safe characteristics. If the background system fails in a centralized system, all electronic payments are blocked. With a decentralized system, by contrast, the consequences of a temporary failure of the background system usually do not even extend as far as the merchant terminals.

Decentralized systems also have disadvantages, primarily with regard to system management. This is because online connections can only be established at certain times, and as a rule only on the initiative of the terminals (see Table 18.1). However, it is essential for system security that the terminals always work with the current blacklist. This is one of the reasons why many systems require each terminal to establish an online connection to the background system at least once a day. This is used to transmit the accumulated transaction data to the background system, with various types of management data being transmitted to the terminal in return. The management data sent to the terminal may include new terminal software, a new set of keys, the current blacklist, and data to be loaded into customer cards.

Mixed systems that are neither fully centralized nor fully decentralized are often used in practice in order to combine the advantages of the two architectures while avoiding their disadvantages. With a mixed architecture, both the terminals and the smart cards can compel an online connection under certain conditions. If this connection cannot be established, the payment transaction does not take place.

Some typical conditions for establishing an online connection are: (a) online authorization is required for payments above a certain amount, which can usually be set individually for each smart card by the system operator; (b) the number of offline transactions and the elapsed time since the last online transaction can be used to decide whether to go online; (c) a random number generator can be used to force a certain percentage of all transactions to occur online. Some

systems also have a special button on the terminal that forces an online transaction. This button can be pressed by the sales staff if they suspect that the customer is using a manipulated card.

All of these decision criteria ensure that on average, every card makes a direct connection to the background system within a defined and statistically computable time interval. The system operator thus recovers direct control over the system, which was initially lost by using a decentralized system. Terminals and vending machines with small turnover can be excluded from these online constraints, since even in case of fraud only small losses can occur. This saves the cost of a link to a communication network, since data exchange can be performed manually by service personnel.

### 18.1.2 Electronic money

Electronic money must have certain properties if it is to be used with the same flexibility as normal money. If these properties are absent either entirely or in part, the capabilities of electronic money are necessarily more or less limited. The essential properties necessary to minimize the difference between electronic money and real money are described below.

#### 18.1.2.1 Processable

Although it may seem obvious, an important property of electronic money is that it can be completely and automatically processed by machines. This is the only way in which large systems can be operated economically.

#### 18.1.2.2 Transferable

Electronic money must not be bound to a particular medium, such as smart cards. It must be possible to transfer electronic money using any desired medium, such as a network or a computer.

#### 18.1.2.3 Divisible

Electronic money must be divisible so that any desired sum can be paid without recourse to using real money. This is similar to normal money, which although not arbitrarily divisible is available in a sufficient variety of denominations that normal purchases can be made using a small number of coins and banknotes.

#### 18.1.2.4 Decentralized

Payment systems with centralized architectures can be easily monitored by the purse system operator, and the opportunities for fraud are very limited. The best example of such a system is the online authorization of credit card purchases. However, centralized systems have certain drawbacks. They are expensive, vulnerable to technical malfunctions, inflexible, and difficult

to modify or extend. Decentralized systems minimize these drawbacks. This can be seen very clearly with payments in the private sphere, in which money changes hands without any involvement by a central body. Electronic money should also have this property, as otherwise it cannot compete with normal money. With an electronic payment system, in concrete terms this means that it must be possible to make payments offline and to transfer money directly from one purse to another one. The ability to make direct transfers between purses (purse-to-purse transactions) is sometimes called transferability.

#### ***18.1.2.5 Monitorable***

Despite the demand for anonymity, electronic money must allow the purse system operator to monitor the system, since this is the only way in which manipulations and security gaps can be recognized and eliminated. This is exactly the same as the situation with normal money, in which every citizen is obliged to immediately report counterfeit money to the appropriate authorities. In the case of electronic money, the purse system operator is responsible for guarding against fraud and forgery, and the operator can and must monitor the consistency of payment flows.

#### ***18.1.2.6 Secure***

A fundamental property of electronic money must naturally be that it is secure against forgery. Any system will collapse within a short time if it is possible to forge or duplicate money in any form or manipulate payment flows. This is why cryptographic functions are used so extensively in electronic payment systems, since this is the only way to achieve the required level of security.

#### ***18.1.2.7 Anonymous***

Anonymity means that it is impossible for anyone to associate payments with particular persons. The significance of this property is very much a question of perspective. From a technical perspective, the purse issuer desires a system with as little anonymity as possible in order to optimize system monitoring. The opportunities for fraud are very limited in nonanonymous systems, since anyone who commits a fraud can quickly be identified. Government organizations, such as the police and tax authorities, have similar interests. Nonanonymous electronic money would give them considerably more scope for monitoring financial transactions than they have enjoyed up to now with normal money.

The position of purse users is diametrically opposite. They consider current payment methods using normal money to represent an excellent state of affairs, and they regard complete anonymity and nontraceability of payment transactions as the optimum solution. Particularly with regard to anonymity, operators of electronic purse systems often choose a compromise solution in the interest of system security. For instance, in most systems payments are anonymous, but loading electronic purses is not. This allows the system to be monitored reasonably well in a simple manner at relatively low cost.

At first sight, some of properties mentioned above appear to be contradictory. For instance, in many cases full anonymity and optimum system monitoring are mutually exclusive. However,

electronic money systems are still in the early stages of development, and there are already systems being planned in which these two properties can certainly be realized simultaneously.

#### ***18.1.2.8 Legal framework and retention of value***

There are two properties of real money that are not mentioned above, although they are highly significant. The first is that real money is legal tender that must be accepted by everyone in the country concerned. In almost all countries, vendors of goods or services are obliged to accept the legal currency of that country as a means of payment. The second property relates to the stability of the currency. Except for a few countries with high rates of inflation, the legal currency in circulation has a stable value. If this is not the case, people will resort to barter or use foreign currencies.

### **18.1.3 Basic system architecture options**

Many different architectures are possible for payment systems with smart cards. For economic reasons, they are often derived from existing systems, most of which are based on magnetic-stripe cards. However, there is no single basic model that applies to all electronic payment systems, since the individual requirements vary too widely. We can therefore only describe the basic principles of such systems in terms of their essential components and using a single example.

Large smart card payment systems basically consist of four components: the background system, the network, the terminals, and the cards.

#### ***18.1.3.1 Background system***

The background system consists of two parts: clearing and management. The clearing subsystem clears all incoming transaction data to settle accounts between the banks, merchants and cardholders participating in the system. It also provides the system monitoring functions. A simple example of such a function is maintaining a running balance to verify that the total of the amounts submitted for clearing does not exceed the total amount of money in the electronic purses. If it does, this means that an attacker has loaded money into smart cards without the knowledge of the background system.

The management portion of the background system looks after all administrative processes, such as distributing new blacklists, switching to new key versions, updating the terminal software, and so on. This subsystem also generates data sets for personalizing smart cards. The background system has overall control of the electronic payment system, regardless of the system architecture. Even with systems that operate entirely offline, the background system defines the global system parameters and monitors the security and operation of the system.

#### ***18.1.3.2 Network***

The network links the background system to the terminals. The connections may be circuit-switched (e.g. analog or ISDN) or packet-switched (e.g. X.25 or TCP/IP). As a rule, the

network is totally transparent with respect to the data packets, which are conveyed unchanged from the sender to the receiver.

#### 18.1.3.3 Terminals

The various types of terminals<sup>3</sup> can be classified as either loading terminals or payment terminals, depending on their functions with regard to financial transactions. They can also be classified as automated terminals or attended terminals. The classic example of an automated terminal is a cash dispenser (ATM). In electronic purse systems, automated terminals are generally used only to load cards. Naturally, it would also be conceivable for terminals to be used to unload purses and pay out the purse balance in cash. Attended terminals are typically found at point-of-sale locations in supermarkets or retail shops. They are always used to pay for goods. In some systems, terminals in banks can be used to load smart cards in exchange for cash payment. However, the customary practice now is to use cash dispensers to load purse cards.

#### 18.1.3.4 Smart cards

Smart cards are the most prolific component of the system. They are of course used as electronic purses, but they can also be used as security modules in various types of terminals. Another use is transporting data between various system components. Cards for this purpose are called transfer cards, and they are used to manually transfer transaction data from a terminal that operates completely offline to one that operates online, such as a cash dispenser or automated teller machine (ATM).

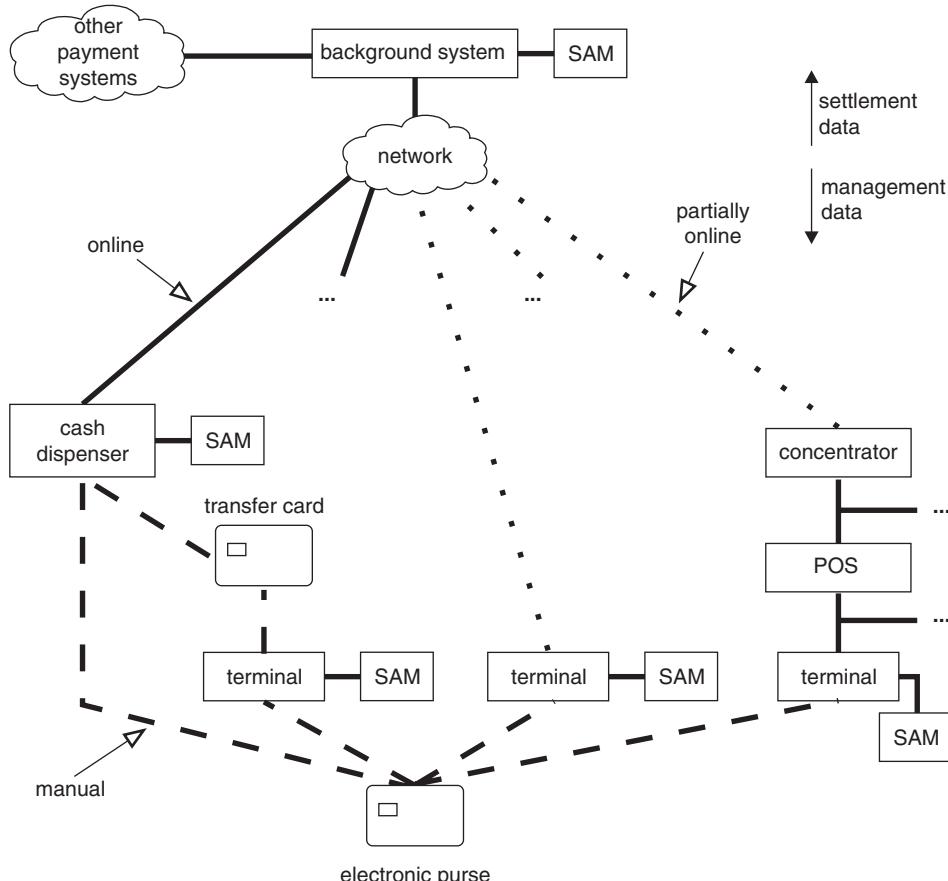
The example shown in Figure 18.4 on the facing page illustrates the system components and their logical connections. The background system is connected to the other components by a transparent network. It can be the background system of another operator or a dedicated component of the electronic purse system.

Cash dispensers, which generally operate online, but can also operate offline for a limited time in the event of a network failure, are most often used to load electronic purses. For this reason, they often have their own security modules, which hold all of the keys necessary for normal operation and the key derivation algorithms.

There are also electronic purse payment systems that operate fully offline. Two examples are parking meters and terminals in taxis. In such systems, transfer cards can be used to transport the transaction data from the security modules to a cash dispenser, which sends the data to the background system via the network. In the other direction, the terminals receive current management data such as blacklists and software updates.

A second type of payment terminal is linked to the network by online connections established as needed. This type of terminal usually operates offline, but it periodically connects to the background system in order to exchange accumulated transaction data and management data. A third type of payment terminal used in sales locations has no direct connection to the network. For example, it could be connected to a supermarket cash register that in turn is connected to a concentrator in the supermarket. This concentrator, which is usually a PC acting as a server, may connect to the background system once a day via the network. The necessary data exchanges occur during this connection interval.

<sup>3</sup> See also Chapter 17, ‘Smart Card Terminals’, on page 735



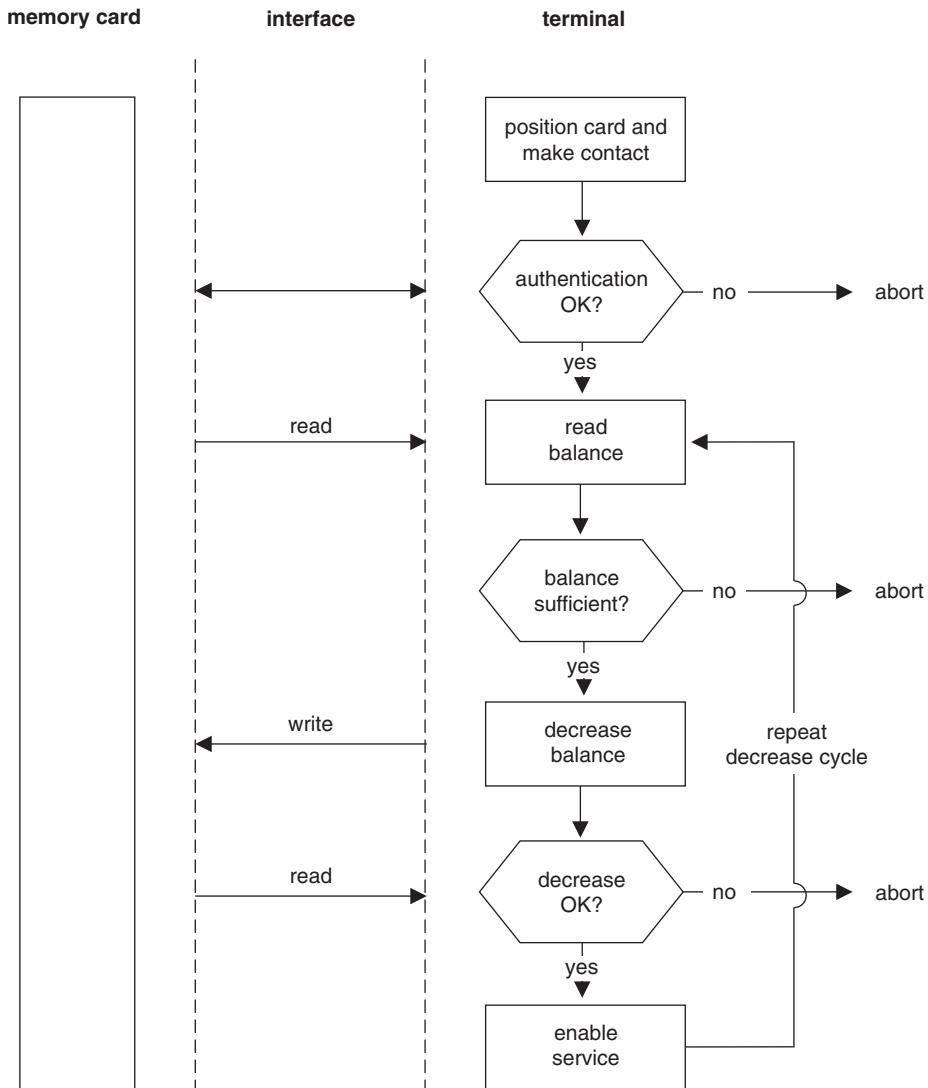
**Figure 18.4** Example architecture of an electronic purse system (SAM = security module)

The Quick system in Austria and the GeldKarte system in Germany are electronic purse systems that are similar to the example described above, and the Visa Cash electronic purse system is largely similar to this example. Relatively large systems often have a distributed system architecture with several background systems operating in parallel. With this architecture, several different purse systems with more than one system operator can be operated with mutual compatibility.

## 18.2 PREPAID MEMORY CARDS

With regard to electronic payment systems, we must not neglect memory cards. They are produced in very large numbers in the form of prepaid electronic purses, and they are used in many applications.<sup>4</sup> This situation will certainly not change over the next few years. Although memory cards will slowly but surely be replaced by microcontroller smart cards, their strength

<sup>4</sup> See also Section 21.2, 'Ski Passes', on page 878



**Figure 18.5** The debiting cycle of a prepaid memory card as seen by the terminal

lies in their unparalleled low cost. The most common application for prepaid electronic purse cards with memory chips is telephone cards, which are used in many countries and simply discarded when they are empty.<sup>5</sup>

A memory card<sup>6</sup> only needs a logic unit and an irreversible down counter to allow it to do its job. The typical debiting cycle in such cards is shown in Figure 18.5. More recent versions also support unilateral authentication of the card by the terminal. For this purpose,

<sup>5</sup> See also Section 19.1, ‘Public Card Phones in Germany’, on page 789

<sup>6</sup> See also Section 2.3.1, ‘Memory cards’, on page 20

the logic unit is enhanced with a simple encryption function, whose task is to encrypt a random number received from the terminal using a secret key stored in the card and return the result to the terminal. This is the only way the terminal can be sure that the memory card is genuine.

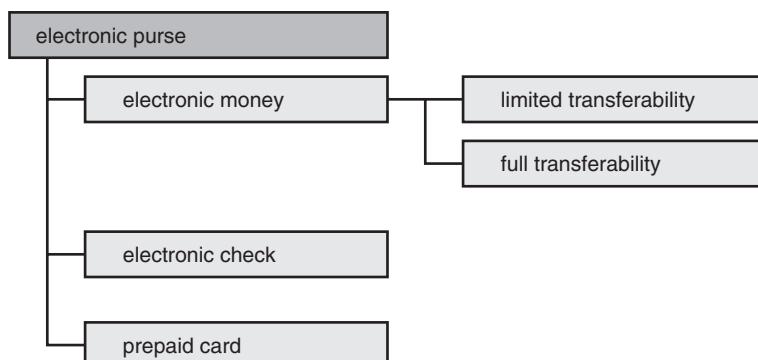
The difficulty of ensuring that memory cards are genuine is not the only factor that hinders their use as general-purpose electronic payment cards. Due to the ease of manipulating systems based on memory cards, it is difficult to construct a payment system that can support a variety of independent service providers, such as kiosks and taxis. This is because secure communication between the terminals and the cards is not possible, with the inevitable result that indirect means must be used to allocate payments to the various parties and perform the associated system monitoring.

Due to their low cost, memory cards have advantages relative to microcontroller cards in certain very limited application areas, but they are not very suitable for use in open payment transaction systems. Most future electronic payment systems will doubtless use microcontroller smart cards because they are significantly more versatile.

## 18.3 ELECTRONIC PURSES

The idea of implementing electronic purses in smart cards goes back to the early days of smart card technology. However, this concept only became real in the mid-1990s with the initial development of the first large systems.

If we take a normal purse containing coins and banknotes as a reference, we can easily see which properties electronic money has in the eye of the user. Money must be put into a purse card before it can be used, which means the card cannot be used like a debit or credit card with payment made on or after receipt of the goods or services. Instead, it must be used like a telephone card, which is paid in advance. The actual payment process must be quick and easy, as otherwise user acceptance will be low. Furthermore, all payments made using a normal purse are anonymous, so it is not possible to reconstruct who bought what at which time. The most annoying characteristic of a purse becomes apparent if it is lost, since the



**Figure 18.6** Types of smart card electronic purse systems. With electronic money, the money is not tied to specific cards and can be denominated as desired. By contrast, electronic check cards have fixed denominations. Prepaid cards contain electronic money that is tied to the card and can be freely denominated

money it contains is irretrievably gone. However, this does not necessarily have to be true of an electronic purse.

The primary advantage of a purse, or rather of the money it contains, is that it is accepted everywhere within a given country. It is precisely this factor that is missing in most existing electronic purse systems. With a telephone card, all you can do is make telephone calls, and nothing else. This is typical of a closed application. An ideal electronic purse, on the other hand, can be used in more than one sector, and thus allow its user to make purchases in many different businesses using a single purse.

### 18.3.1 Inter-sector electronic purses compliant with EN 1546

The European Commission resolved in 1990 to have the *Comité Européen de Normalisation* (CEN) produce a European standard for inter-sector electronic purse systems. Work on the standard commenced in 1991. Up to 1998, the individual project teams had spent around six man-years on developing this standard. As a variety of independent experts participated in this effort, the standard is probably free of security gaps and is thus already quasi-evaluated.

The EN 1546 standard is a public standard, and the processes of its individual functions are described in great detail. It is therefore very suitable for demonstrating loading and payment processes in an electronic purse system. With many existing systems, these processes cannot be described at the same level of detail because the relevant commands, processes and internal functions are confidential. This standard thus provides a very useful illustration of the fundamental external and internal processes of an electronic purse system.

The typical application areas are clearly indicated by the first systems based on this standard. The Danish system operator Danmønt has introduced a purse compliant with this standard in its existing system. In Austria, the Eurocheque card issued throughout the country also includes an electronic purse called Quick that is based on the EN 1546 standard. However, the largest international application based on this standard is Visa Cash, which is one of several electronic purses offered by Visa. The essential features of the Common Electronic Purse Specifications (CEPS), which are the international specifications for electronic purse systems, are also based on EN 1546.<sup>7</sup>

The EN 1546 standard is titled ‘Inter-Sector Electronic Purse’ and is divided into four parts. The first part, ‘Definition, concepts and structures’, describes the overall system (see Figures 18.7 and 18.8). This basic document defines and explains, in abstract form, all of the logical components and their interconnections.

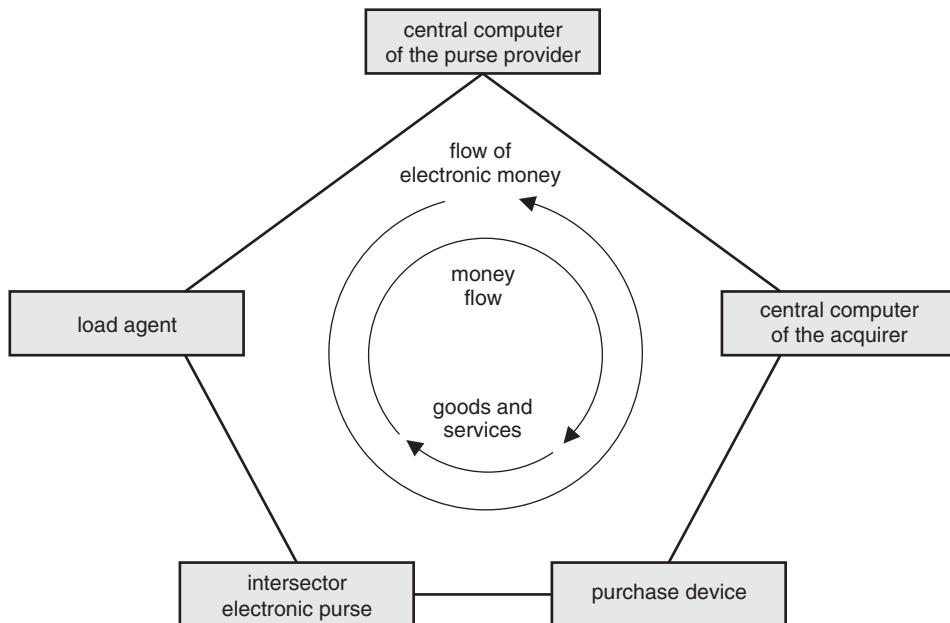
In the second part, these basic concepts are used to describe the security architecture of the overall system and its individual components. This includes not only mechanisms for maintaining security, but also possible attacks and the necessary countermeasures.

Part three, ‘Data elements and interchanges’, contains descriptions and definitions of the data elements needed for the electronic purse system. It also describes the commands for the smart cards and security modules and their associated responses.

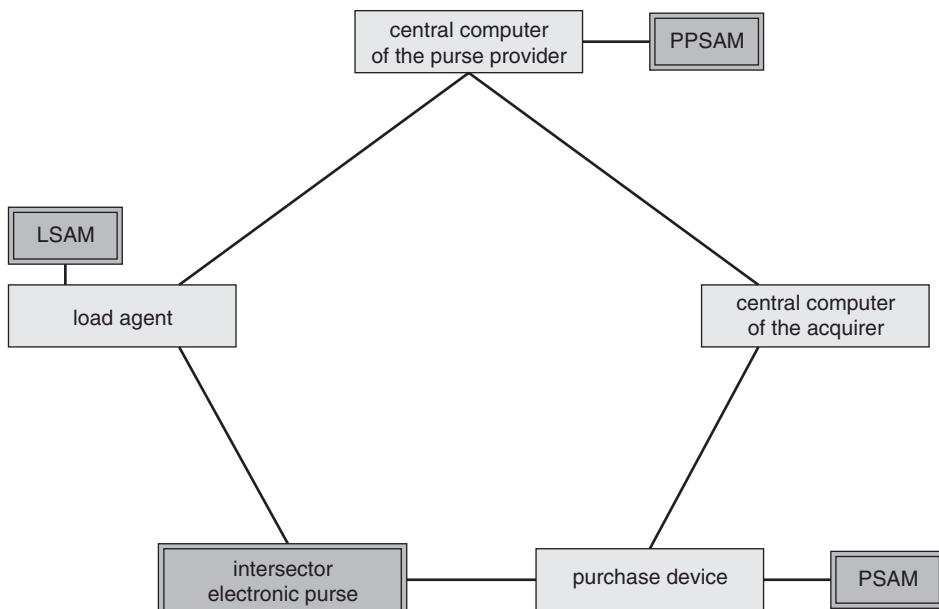
The final part describes the state machines and states of the devices used. It employs a symbolic representation similar to the well-known flowchart diagrams. This formal notation, which is known as SDL notation, originates from the CCITT Z.100 recommendation.<sup>8</sup>

<sup>7</sup> See also Section 18.3.2, ‘CEPS’, on page 774

<sup>8</sup> See also Section 6.3, ‘SDL Notation’, on page 117



**Figure 18.7** Basic structure of an inter-sector electronic purse system and associated payment flows according to EN 1544



**Figure 18.8** Components and internal connections of electronic purse systems according to EN 1546. The components with a single outline are not secure, while those with a double outline are secure

This standard, which encompasses around 300 pages, contains a complete description of an electronic purse system, including the smart cards, the terminals with their security modules, and the background and clearing systems. Its objective is to establish a common standard for large electronic purse systems with very many smart cards and wide geographical distribution.

The EN 1546 standard provides considerable freedom for actual implementation and is intended to serve more as a framework than a precise specification of individual bits and bytes. It is thus perfectly possible for two different systems to be fully compliant with this standard but mutually incompatible, for example because they use different cryptographic algorithms.

The purse provider bears the overall responsibility for the system and is also the system manager. It is comparable to a GSM network operator. The standard defines the term ‘purse holder’ to refer to the user of an electronic purse. This is the person who makes payments using the electronic purse application in the card and receives goods or services in return.

There are also three other parties that have roles in the system. The service providers provide goods or services that are taken by the user and paid using the electronic purse. The acquirer is responsible for establishing and managing the data links between the purse issuer and the service providers. The acquirer may also consolidate individual transactions originating from the payment devices, so that the purse provider only receives collective certificates. A load agent is a counterpart of a service provider in that it can re-load electronic purses in exchange for payment.

These five parties do not have to be real; they may also be virtual. However, real technical components classified according to their level of security are allocated to each of these parties. Components regarded as secure prevent any external manipulation of the data that is processed or stored in them. With components regarded as nonsecure, such manipulation is at least theoretically possible. However, the system as a whole is designed such that manipulation of any of the components marked as nonsecure in Figure 18.8 on the preceding page will not affect the overall security of the system.

The abbreviation ‘IEP’ stands for ‘inter-sector electronic purse’ and refers to the electronic purse application in the smart card. A payment device is used to pay for received goods or services. It is a terminal with a keypad and a display, and it must also have a security module. The term ‘secure application module’ (SAM) is used in the standard as a general designation for all types of security modules. A SAM contains all the secret keys necessary for transactions between the IEP and the central computer of the purse provider. Naturally, the keys never leave the security module, but are used only inside the SAM by the cryptographic algorithms.<sup>9</sup> Accordingly, many systems have only direct links between the secure components. The nonsecure components are only used to convey sensitive data transparently.

The concept of an electronic purse system can only be implemented using smart cards. They are thus the central component of the system description in the CEN EN 1546 standard. All of the associated files, commands, states and processes are defined and described in this standard. In order to define the entire system, there are similar parts that address the security module and the other components.

As this is a standard rather than a specification, the purse provider is given a large degree of freedom and many options. A variety of functions can be used to construct the purse. For example, a simple system that only supports loading and paying with the purse can be implemented easily. This can be further extended with functions for refunding payments,

<sup>9</sup> An example key management system for an EN 1546 electronic purse system is described in Section 7.7.6, ‘Key management example’, on page 185

modifying purse parameters, and currency exchange. The exact selection of the many complex options is largely left to the purse provider, which must choose the options that best meet its particular needs.

The most important aspects of an inter-sector electronic purse system with regard to the IEP, which is the smart card, are described below.

### 18.3.1.1 Data elements

Designations of all data elements were introduced to allow the data used in the entire system for the electronic purse application to be specified unambiguously (see Table 18.2). Thanks

**Table 18.2** Summary of the most important standard data elements of EN 1546

Data element	Description
ALG <sub>IEP</sub>	Cryptographic algorithm used by an IEP
AM <sub>IEP</sub>	Authentication mode required by an IEP
AP <sub>IEP</sub>	Application profile of an IEP
BAL <sub>IEP</sub> , BAL <sub>PSAM</sub> , BAL <sub>PPSAM</sub>	Balance of an IEP, PSAM or PPSAM
BALmax <sub>IEP</sub>	Maximum balance of an IEP)
CC <sub>IEP</sub> , CC <sub>PSAM</sub> , CC <sub>PPSAM</sub>	Completion code from the IEP, PSAM or PPSAM
CT	Collection status
CURR <sub>IEP</sub> , CURR <sub>LDA</sub> , CURR <sub>PDA</sub>	Actual currency for an IEP, LDA or PDA
DACT <sub>IEP</sub>	Activation date of an IEP
DD	Discretionary data
DDEA <sub>IEP</sub>	Deactivation date of an IEP
DEXP <sub>IEP</sub>	Expiry date of an IEP)
ID <sub>IEP</sub> , ID <sub>PSAM</sub> , ID <sub>PPSAM</sub>	Identifier for an IEP, PSAM or PPSAM
IEP	Inter-sector electronic purse
IK <sub>IEP</sub> , IK <sub>PSAM</sub> , IK <sub>PPSAM</sub>	Key information for an IEP, PSAM or PPSAM
LDA	Load device application
LSAM	Load SAM
M <sub>LDA</sub> , M <sub>PDA</sub>	Transaction amount for load or purchase
MTOT <sub>IEP</sub> , MTOT <sub>PSAM</sub>	Total transaction amount for a purchase
NC	Number of collection (designates a final sum)
NI	Number of individual transactions
NT <sub>IEP</sub> , NT <sub>LSAM</sub> , NT <sub>PSAM</sub>	Transaction number for an IEP, PSAM or PPSAM
PDA	Purchase device application
PP <sub>IEP</sub> , PP <sub>PSAM</sub> , PP <sub>PPSAM</sub>	Purse provider identifier for an IEP, PSAM or PPSAM
PPSAM	Purse provider SAM
PSAM	Purchase SAM
R	Random number
S <sub>1</sub>	IEP signature
S <sub>2</sub>	PSAM/PPSAM signature
S <sub>3</sub>	IEP signature
S <sub>4</sub>	PSAM signature
SAM	Secure application module
TM	Total amount
TRT	Transaction type and status

to the brevity of these designations, data flows and processing can be represented simply and unambiguously in mathematically correct notation. The standard also includes a simple data dictionary that describes the contents and associated formats of the standardized data elements.

### 18.3.1.2 Files

The entire electronic purse application is contained in a separate DF in the smart card. All of the files necessary for proper operation are located in this DF. In addition, data related to the card, the chip, other applications and the like is held in several files directly under the MF.

The data elements needed for operating the electronic purse are contained in six EF files located in a DF for the purse. These files are listed and briefly described in Table 18.3 on the next page.

The EF<sub>IEP</sub> file specifies the general parameters of the purse, which form the basis for all purse transactions. The EF<sub>IK</sub> file holds specific data for each available key. EF<sub>BAL</sub> holds the current purse balance available to the user. The log files are used exclusively to record all transactions, separated by function. The data in these files is essential for refunding payments and handling errors. There are separate log files for loading, purchasing, modifying the purse parameters, and currency exchanges. All log files have a cyclic structure in order to maintain a record of the most recent transactions.

### 18.3.1.3 Commands

The files form the foundation of the purse, and the commands<sup>10</sup> are built on this foundation. Eight commands are necessary for operating the purse system. Three of these commands are defined in the ISO/IEC 7816-4 standard: SELECT, READ BINARY, and READ RECORD. These commands are only used to select the electronic purse application using its AID and subsequently read various purse data items from the files as necessary.

The other five commands were developed specifically for use with electronic purses. They are always used in pairs for individual transactions, since in principle they act as a sort of mutual authentication. Data needed for the transaction is also exchanged during authentication. These commands and responses are structured such that any manipulation at the interface between the card and terminal can be detected immediately, resulting in immediate termination of the transaction and logging of the event.

All purse commands directly access data elements in the purse files for both reading and writing. The files are automatically selected by the operating system prior to these accesses. For instance, basic purse data is sometimes needed while a command is being processed. In this case, the operating system selects the EF<sub>IEP</sub> file, and the desired data element is provided to the command. All transactions, as well as the most important data, are recorded in suitable log files during the command-response cycles. EN 1546 defines the commands listed in Table 18.4 on page 766 and specifies their operation inside the card in detail.

The standard does not provide commands for verifying or changing PINs, since these functions are not needed for proper operation of the purse. However, additional commands for PIN verification and management can be included in the purse application as needed without causing any interference or problems with existing purse commands.

<sup>10</sup> See Section 11.14, ‘Commands for Electronic Purses’, on page 402 for detailed descriptions of the commands

**Table 18.3** The files and associated data elements needed for an electronic purse in accordance with EN 1546. Log files for currency exchange transactions and parameter changes are not shown

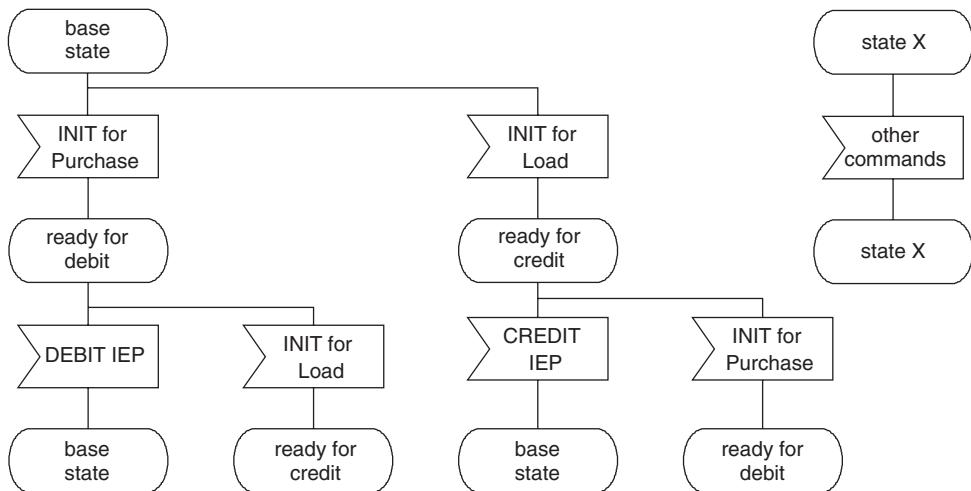
File	Function	Data element and description	
EF <sub>IEP</sub>	Fixed data and purse parameters	PP <sub>IEP</sub>	IEP purse provider identifier
		ID <sub>IEP</sub>	IEP identifier
		DEXP <sub>IEP</sub>	IEP expiry date
		DACT <sub>IEP</sub>	IEP activation date
		DDEA <sub>IEP</sub>	IEP deactivation date
		AM <sub>IEP</sub>	IEP authentication mode
		AP <sub>IEP</sub>	IEP application profile
		DD	User-determined data
EF <sub>IK</sub>	Information related to all keys	ALG <sub>IEP</sub>	IEP cryptographic algorithm
		IK <sub>IEP</sub>	IEP key information
		DD	User-determined data
EF <sub>BAL</sub>	Purse balance	BAL <sub>IEP</sub>	IEP balance
		CURR <sub>IEP</sub>	IEP currency
		BALmax <sub>IEP</sub>	IEP maximum balance
		DD	User-determined data
EF <sub>TFIELD</sub>	Transaction field	NT <sub>IEP</sub>	IEP transaction number
EF <sub>LLOG</sub>	Log file for load transactions	TRT	Transaction type and status
		NT <sub>IEP</sub>	IEP transaction number
		BAL <sub>IEP</sub>	IEP balance (new balance)
		M <sub>LDA</sub>	Transaction amount for loading
		CURRLDA	Currency for loading
		ID <sub>PPSAM</sub>	Purse provider identifier for PPSAM
		CC <sub>IEP</sub>	IEP completion code
		DD	User-determined data
EF <sub>PLOG</sub>	Log file for payment transactions	TRT	Transaction type and status
		NT <sub>IEP</sub>	IEP transaction number
		BAL <sub>IEP</sub>	IEP balance (new balance)
		M <sub>PDA</sub>	Transaction amount for purchase
		CURRPDA	Currency for purchase
		ID <sub>PSAM</sub>	Purse provider identifier for PSAM
		NT <sub>PSAM</sub>	PSAM transaction number
		CC <sub>IEP</sub>	IEP completion code
		DD	User-determined data

#### 18.3.1.4 States

As can be seen from the summary description of the commands, each transaction is composed of an introductory initialization command and a subsequent command that completes the transaction. In order to specify the command sequence, state diagrams are used to define the necessary states and state transitions in the purse application, as illustrated in Figure 18.9. This

**Table 18.4** Specific commands for electronic purses defined in EN 1546

Command	Function
INITIALIZE IEP	• initialization for a subsequent purse command
CREDIT IEP	• loading the purse • refunding a previous payment • error correction
DEBIT IEP	• paying with the purse • confirming payment
CONVERT IEP CURRENCY	• currency conversion
UPDATE IEP PARAMETER	• changing the general purse parameters

**Figure 18.9** Simplified state diagram for loading and paying with an electronic purse compliant with EN 1546

naturally requires the card to contain a state machine. The card can accept or reject various commands according to its current state.

#### 18.3.1.5 Cryptographic algorithms

The entire security of the system is based on a cryptographic algorithm. All the messages exchanged between the components have appended signatures to allow manipulations to be detected. This is the only protection for messages, which are always exchanged in plaintext.<sup>11</sup> Message exchange is fashioned such that any desired cryptographic algorithm can be used to generate the signatures. The symmetric DES algorithm is most often used at present, but the

<sup>11</sup> If the DES algorithm is used, the messages are protected by a four-byte MAC at the end of the message that acts as a signature. However, updated security requirements specify an eight-byte MAC with triple DES for this purpose (as used with the German GeldKarte)

standard also allows asymmetric algorithms such as RSA or DSS. This algorithm independence is a great advantage, since it considerably extends the useful life and flexibility of the standard. An example of a possible key hierarchy with key derivation is described in Section 7.7.6, ‘Key management example’, on page 185.

#### 18.3.1.6 General processes

In addition to specifying the files, commands and states, the standard describes and explains the associated processes. They are specified in detail in terms of their data elements, using a pseudolanguage similar to Basic. This is necessary because the security of some processes strongly depends on the sequence of the operations used for command processing in the card. For example, during a transaction the corresponding log file must always be updated before the response is sent to the terminal. The processes and transactions for all components are also specified precisely. For smart cards, descriptions are provided for the following electronic purse processes:

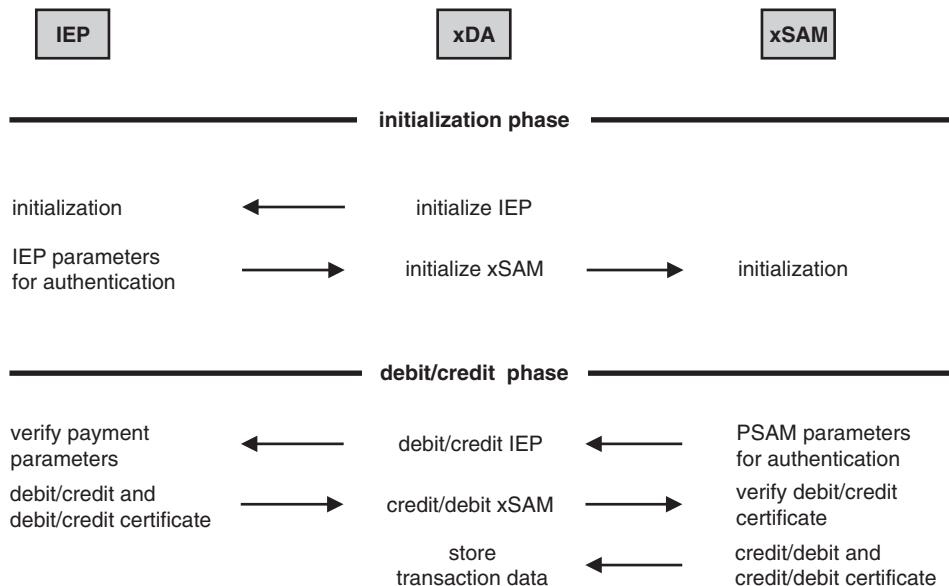
- loading;
- paying;
- refunding a payment;
- error correction;
- currency conversion;
- changing the purse parameters.

The commands for reading files can of course also be used for monitoring purposes. However, the actions involved may vary, depending on the purse provider and the objective of monitoring.

The structures of all specified procedures are governed by a fundamental principle, which is that it must never be possible to create electronic money by manipulating data transmission between the components or by abnormally terminating a transaction. In the worst case, such actions result in the destruction of electronic money. This automatically excludes certain types of attack. Incidentally, this principle is incorporated in the design of nearly all electronic purse systems.

As shown in Figure 18.10, each transaction is essentially divided into three phases. Basic initialization of the components concerned occurs in the first phase. Actual execution of the transaction takes place in the second phase. The third phase, which is optional, confirms the previous actions. Successful completion of the first two phases amounts to a unilateral (or optionally, mutual) authentication of the two components.

In all of the processes, unilateral or mutual authentication is interleaved with the actual purse functions (payment, loading, etc.). This minimizes the time required for the purse transactions and increases security, since it significantly reduces the number of necessary commands. This is similar to the situation with the standardized ISO commands INTERNAL AUTHENTICATE and EXTERNAL AUTHENTICATE, which can be replaced by the nonstandardized MUTUAL AUTHENTICATE command without affecting functionality or security.



**Figure 18.10** Basic process of an EN 1546 electronic purse transaction (phases 1 and 2)

### 18.3.1.7 Load process

Before an electronic purse can be used to make payments, it must be loaded. The transaction sequence in Table 18.6 on the next page illustrates the option of loading the electronic purse (IEP) from a terminal with a direct online link to a background system with a security module (PPSAM), using the abbreviations defined in Table 18.5. The standard also allows other options, such as loading the purse from a terminal with its own security module (LSAM). However, the process illustrated is commonly used in current systems because it gives the system operator full control over loading.

The transaction sequence illustrates how a background system with a security module (PSAM) can load an electronic purse (IEP) via a terminal (LDA). The card user first inserts the card in the terminal, which executes a reset. In the ATR, the card sends the terminal various general parameters for the subsequent communication process.

**Table 18.5** Abbreviations of the functions and processes used to depict payment transactions according to EN 1546

Abbreviation	Meaning
Ax	Unique designation for an action
Cx	Unique designation for a command
Parameters [...]	Request to a participant, with the indicated data elements
Response [...]	Response to a previous request, with the indicated data elements
Rx	Unique designation for a response
Sign (...)	Generate the signature of the specified data elements
Verify (...)	Verify or the indicated data elements a function
Write (...)	Write the indicated data elements to a file

**Table 18.6** Transaction sequence for loading an electronic purse (IEP) online via a terminal (LDA) with a security module (PPSAM)

IEP (purse)	LDA (terminal)	PPSAM (security module)
R1:	← C1:	
Response [ATR]	→ RESET	
R2:	← C2:	
Response [CC <sub>IEP</sub> ]	→ SELECT Parameters [DF <sub>IEP</sub> ] A1: Input [M <sub>LDA</sub>    CURR <sub>LDA</sub> ] C3: INITIALIZE PPSAM for Load Parameters [M <sub>LDA</sub>    CURR <sub>LDA</sub> ]	→ A2: Verify (CURR <sub>LDA</sub> ) Verify (BAL <sub>PPSAM</sub> ≥ M <sub>LDA</sub> ) Generate R
		← R3: Response [PP <sub>PPSAM</sub>    ID <sub>PPSAM</sub>    R]
A3: Verify (PP <sub>PPSAM</sub> ) Verify (CURR <sub>LDA</sub> )	← C4: INITIALIZE IEP for Load Parameters [PP <sub>PPSAM</sub>    ID <sub>PPSAM</sub>    R    M <sub>LDA</sub>    CURR <sub>LDA</sub> ] Verify (BAL <sub>IEP</sub> + M <sub>LDA</sub> ≤ BAL <sub>max,IEP</sub> ) NT <sub>IEP</sub> := NT <sub>IEP</sub> + 1 KSE <sub>IEP</sub> = f(KD <sub>IEP</sub> , DEXP <sub>IEP</sub> , NT <sub>IEP</sub> ) S <sub>1</sub> := Sign (PP <sub>IEP</sub>    ID <sub>IEP</sub>    DEXP <sub>IEP</sub>    NT <sub>IEP</sub>    M <sub>LDA</sub>    CURR <sub>LDA</sub>    BAL <sub>IEP</sub>    ID <sub>PPSAM</sub>    R) Write (EF <sub>LLOG</sub> )	
R4:	→ Response [PP <sub>IEP</sub>    ID <sub>IEP</sub>    ALG <sub>IEP</sub>    IK <sub>IEP</sub>    DEXP <sub>IEP</sub>    NT <sub>IEP</sub>    BAL <sub>IEP</sub>    S <sub>1</sub>    CC <sub>IEP</sub> ]	→ A4: Verify (PP <sub>IEP</sub> ) Verify (ID <sub>IEP</sub> )
	C5: DEBIT PPSAM Parameters [PP <sub>IEP</sub>    ID <sub>IEP</sub>    ALG <sub>IEP</sub>    IK <sub>IEP</sub>    DEXP <sub>IEP</sub>    NT <sub>IEP</sub>    BAL <sub>IEP</sub>    S <sub>1</sub> ]	Verify (ALG <sub>IEP</sub> ) Verify (IK <sub>IEP</sub> ) Verify (DEXP <sub>IEP</sub> ) KD <sub>PPSAM</sub> = f(ID <sub>IEP</sub> , VK <sub>IEP</sub> , KM <sub>PPSAM</sub> )

(Continued)

**Table 18.6** (Continuation) Transaction sequence for loading an electronic purse online

IEP (purse)	LDA (terminal)	PPSAM (security module)
		KSES <sub>PPSAM</sub> = f(KD <sub>PPSAM</sub> , DEXP <sub>IEP</sub> , NT <sub>IEP</sub> ) Verify (S1) S <sub>2</sub> := Sign (PP <sub>PPSAM</sub>    ID <sub>IEP</sub>    NT <sub>IEP</sub>    M <sub>LDA</sub>    CURR <sub>LDA</sub> ) BAL <sub>PPSAM</sub> := BAL <sub>PPSAM</sub> - M <sub>LD</sub>
		← R5: Response [IK <sub>PPSAM</sub>    S <sub>2</sub>    CC <sub>PPSAM</sub> ]
A5: Verify (S2) BAL <sub>IEP</sub> := BAL <sub>IEP</sub> + M <sub>LDA</sub> S <sub>3</sub> := Sign (PP <sub>IEP</sub>    ID <sub>PPSAM</sub> , R, CC <sub>IEP</sub> ) Write (EF <sub>LLOG</sub> )	← C6: CREDIT IEP Parameters (IK <sub>PPSAM</sub>    S <sub>2</sub> )	
R6: Response [S <sub>3</sub>    CC <sub>IEP</sub> ]	→	
	C7: PPSAM Load Acknowledgment Parameters [S <sub>3</sub> , CC <sub>IEP</sub> ]	→ A6: Verify (S3) Verify (CC <sub>IEP</sub> )
		← R7: Response [CC <sub>PPSAM</sub> ]

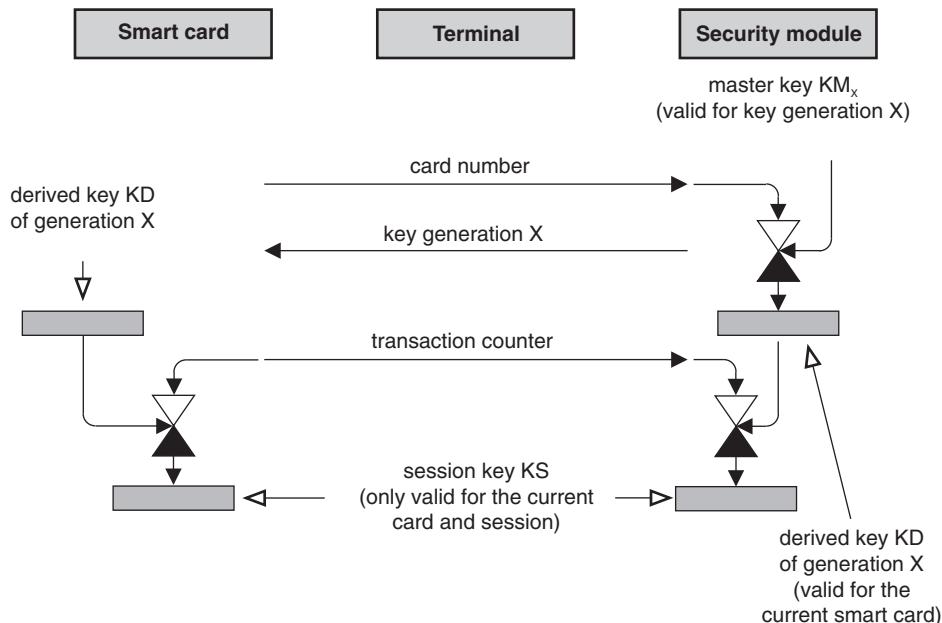
Figure 18.11 shows a key derivation process that can be used in an electronic purse system compliant with EN 1546.

After this, the terminal selects the electronic purse DF in the IEP card. Once this has been done successfully, the card user enters the load amount in the terminal, using an acceptable currency. This data is sent to the PPSAM with the first purse command. The PPSAM verifies the specified currency and the maximum available load amount. In response, it returns three data elements to the terminal.

The terminal appends the load amount (M<sub>LDA</sub>) and the associated currency (CURR<sub>LDA</sub>) to the data elements received from the PPSAM and sends all of this data to the IEP with the INITIALIZE IEP for Load command. The IEP checks, among other things, whether the new purse balance after the load amount is added would exceed the maximum allowable amount in the purse (BALmax<sub>IEP</sub>). If it would not, the IEP increments a transaction counter (NT<sub>IEP</sub>), computes the session key (KSES<sub>IEP</sub>), and generates a signature (S<sub>1</sub>). It returns these items to the terminal along with several other data elements.

After this command, the terminal simply conveys the received data elements to the PPSAM. Here they are checked against the permitted range of values, and a card-specific key (KD<sub>PPSAM</sub>) and a session key (KSES<sub>PPSAM</sub>) are generated.

If the subsequent verification of signature S<sub>1</sub> is successful, the card has been authenticated because it must know the secret key for computing S<sub>1</sub>. The PPSAM then generates signature



**Figure 18.11** A possible key derivation process for an EN 1546 electronic purse system. The key depends on the combination of a card-specific key belonging to the key generation specified in the data sent to the card and a session-specific transaction counter. Derived keys can be used for various purposes, such as making purchases or debiting monetary units

$S_2$  and sends it to the terminal along with the key data ( $IK_{PPSAM}$ ). The terminal again only conveys these data elements to the card, this time using the CREDIT IEP command.

The IEP now verifies signature  $S_2$ . If this is successful, the PPSAM has also been authenticated by the IEP. The balance of the electronic purse ( $BAL_{IEP}$ ) is then increased. The IEP next generates a third signature ( $S_3$ ), which is sent to the terminal as confirmation that the balance has been successfully increased. The final command transfers this signature to the PSAM, which completes the entire loading transaction.

The process described here is one of several options. It is often used in practice because it is quite common to perform purse loading transactions online. EN 1546 also describes options for loading with the aid of a special loading security module (LSAM). This type of security module can be installed in specific decentralized loading terminals, such as cash dispensers.

### 18.3.1.8 Payment process

The transaction sequence in Table 18.7 on the following page illustrates the process of making a purchase with the necessary components: an electronic purse card (IEP), a terminal (PDA), and a security module in the terminal (PSAM).

After the purse card is inserted in the terminal, the terminal performs a reset sequence to request an ATR from the PSAM and from the IEP. If either of these ATRs is not as expected, the terminal terminates the payment process. If the ATRs are as expected, the terminal selects

**Table 18.7** Transaction sequence for online payment with an electronic purse (IEP) using a terminal (LDA) and a security module (PSAM) in accordance with EN 1546

IEP (purse)	PDA (terminal)	PSAM (security module)
	C1: RESET	→ R1: ← Response [ATR]
R2: Response [ATR]	← C2: → RESET	
R3: Response [CC <sub>IEP</sub> ]	← C3: → SELECT (DF <sub>IEP</sub> )	
A1: NT <sub>IEP</sub> := NT <sub>IEP</sub> + 1 KSES <sub>IEP</sub> = f(KD <sub>IEP</sub> , DEXP <sub>IEP</sub> , NT <sub>IEP</sub> ) S <sub>1</sub> := Sign (PP <sub>IEP</sub>    ID <sub>IEP</sub>    DEXP <sub>IEP</sub>    NT <sub>IEP</sub> ) Write (EF <sub>PLOG</sub> ) R4: Response [PP <sub>IEP</sub>    ID <sub>IEP</sub>    ALG <sub>IEP</sub>    IK <sub>IEP</sub>    DEXP <sub>IEP</sub>    CURR <sub>IEP</sub>    AM <sub>IEP</sub>    NT <sub>IEP</sub>    S <sub>1</sub>    CC <sub>IEP</sub> ]	← C4: INITIALIZE IEP for Purchase Parameters []	
	C5: INITIALIZE PSAM for Purchase Parameters [PP <sub>IEP</sub>    ID <sub>IEP</sub>    ALG <sub>IEP</sub>    IK <sub>IEP</sub>    DEXP <sub>IEP</sub>    CURR <sub>IEP</sub>    AM <sub>IEP</sub>    NT <sub>IEP</sub>    S <sub>1</sub> ]	→ A2: Verify (PP <sub>IEP</sub> ) Verify (ID <sub>IEP</sub> ) Verify (ALG <sub>IEP</sub> ) Verify (IK <sub>IEP</sub> ) Verify (DEXP <sub>IEP</sub> ) Verify (CURR <sub>IEP</sub> ) Verify (AM <sub>IEP</sub> ) NT <sub>PSAM</sub> := NT <sub>PSAM</sub> + 1 KD <sub>PSAM</sub> = f(ID <sub>IEP</sub> , VK <sub>IEP</sub> , KM <sub>PSAM</sub> ) KSES <sub>PSAM</sub> = f(KD <sub>PSAM</sub> , DEXP <sub>IEP</sub> , NT <sub>IEP</sub> ) Verify (S <sub>1</sub> ) MTOT <sub>PSAM</sub> := 0 S <sub>2</sub> := Sign (PP <sub>PSAM</sub>    ID <sub>PSAM</sub>    NT <sub>PSAM</sub>    MTOT <sub>PSAM</sub>    ID <sub>IEP</sub>    AM <sub>IEP</sub>    NT <sub>IEP</sub> ) R5:
	A3: Input [M <sub>PDA</sub>    CURR <sub>PDA</sub> ]	← Response [ID <sub>PSAM</sub>    NT <sub>PSAM</sub>    IK <sub>PSAM</sub>    S <sub>2</sub>    CC <sub>PSAM</sub> ]

(Continued)

**Table 18.7** (*Continuation*) Transaction sequence for online payment with an electronic purse

IEP (purse)	PDA (terminal)	PSAM (security module)
A4:	← C6: DEBIT IEP	
Verify (CURR <sub>PDA</sub> )	Parameters [ID <sub>PSAM</sub>	
Verify (BAL <sub>IEP</sub> ≥ M <sub>PDA</sub> )	NT <sub>PSAM</sub>    IK <sub>PSAM</sub>    S <sub>2</sub>    M <sub>PDA</sub>    CURR <sub>PDA</sub> ]	
Verify (S <sub>2</sub> )		
MTOT <sub>IEP</sub> := MTOT <sub>IEP</sub> + M <sub>PDA</sub>		
S <sub>3</sub> := Sign (PP <sub>IEP</sub>    ID <sub>IEP</sub>    AM <sub>IEP</sub>    NT <sub>IEP</sub>    ID <sub>PSAM</sub>    NT <sub>PSAM</sub>    MTOT <sub>IEP</sub>    CURR <sub>IEP</sub> )		
BAL <sub>IEP</sub> := BAL <sub>IEP</sub> - M <sub>PDA</sub>		
Write (EF <sub>PLOG</sub> )		
R6:	→ C7: CREDIT PSAM Parameters [S <sub>3</sub> , M <sub>PDA</sub> ]	→ A5: Verify (S <sub>3</sub> ) MTOT <sub>PSAM</sub> := MTOT <sub>PSAM</sub> + M <sub>PDA</sub>
Response [S <sub>3</sub>    CC <sub>IEP</sub> ]		← R7: Response [CC <sub>PSAM</sub> ]
C8:	→ A6: PSAM Complete Purchase Parameters []	TM := TM + MTOT <sub>PSAM</sub> S <sub>4</sub> := Sign (PP <sub>PSAM</sub>    ID <sub>PSAM</sub>    N    NI(NC)    TM(NC)    CURR(NC)    CT(NC))
		← R8: Response [S <sub>4</sub> , CC <sub>PSAM</sub> ]

the electronic purse DF in the IEP. If this file cannot be selected, the process is terminated. For the sake of clarity, these processes and general error handling are not shown in the figure.

After selecting the purse DF in the IEP, the terminal sends the initialization command INITIALIZE IEP for Purchase. The IEP receives this command, increments the transaction counter, and generates the signature (S<sub>1</sub>) of several data elements. It then sends these data elements and the signature to the terminal.

The terminal next sends the initialization command INITIALIZE PSAM for Purchase to the PSAM. This command simply conveys the data elements received from the card to the PSAM. The PSAM verifies this data, which means that the expiry date (DEXP<sub>IEP</sub>), the currency (CURR<sub>IEP</sub>), the cryptographic algorithm (ALG<sub>IEP</sub>), and the other supplied data elements are compared with values stored in the PSAM. If all the comparisons are successful, the transaction counter (NT<sub>PSAM</sub>) is incremented. If any of the comparisons fails (for example, if card is past its expiry date), command processing is immediately terminated and an appropriate return code is sent to the terminal (PDA).

The PSAM next uses the data supplied by the IEP to generate a derived key and a session key, and it verifies signature S<sub>1</sub>. If S<sub>1</sub> is correct, all the supplied data elements are authentic,

and the IEP has been authenticated by the PSAM. This means that the PSAM knows that the card with the electronic purse is genuine.

Next, the PSAM generates a signature ( $S_2$ ), which is sent to the terminal along with several other data elements. The user now enters the amount to be paid ( $M_{PDA}$ ) and the associated currency ( $CURR_{PDA}$ ) in the terminal. The terminal then issues the DEBIT IEP command to send the entered amount ( $M_{PDA}$ ) and the data elements previously received from the PSAM to the card. The IEP checks whether there is enough money in the purse to make the payment. If there is, it verifies the supplied signature  $S_2$ . If the signature is correct, the data has not been manipulated during transmission and the PSAM has been authenticated, since only a genuine PSAM can have the secret key needed to generate signature  $S_2$ . Now the appropriate amount is subtracted from the purse balance, a log file is updated, and a signature ( $S_3$ ) is generated to confirm the debit transaction just performed.

Signature  $S_3$  and the debited amount are sent via the terminal to the PSAM, which verifies  $S_3$ . If this signature is correct, the amount debited in the IEP is added to an internal data element ( $MTOT_{PSAM}$ ). The next command, PSAM Complete Purchase, updates the PSAM balance by adding  $MTOT_{PSAM}$  to the purse balance ( $TM$ ). Finally, the PSAM receives a signature ( $S_4$ ) to confirm that the payment transaction has been completed successfully.

The process described is a very simple example of the various payment processes described in EN 1546. There are several other options, including a special fast debiting process for card phones and an option that allows a receipt to be generated at the end of the transaction.

Files, commands and processes are also specified for the other major system components. This applies especially to the security modules, since system security depends entirely on these modules. Statistical methods can be used to monitor the overall system, which in a large application may include tens of thousands of terminals and several hundred thousand smart cards. Maintaining a full account for each smart card would violate the requirement for anonymity and would anyhow require far too much computation. However, experiments have shown that the security of the overall system can be monitored continuously at an acceptable cost by random sampling the payment streams.

The EN 1546 European standard is one of the foundations for inter-sector electronic purse systems using smart cards. Almost all processes and functions that were in common use and regarded as worthwhile when the standard was generated were included in the standard. There is only one function that has not yet been described, although card users may find it attractive in certain circumstances. This is purse-to-purse transactions, which means transferring electronic money directly from one purse to another.

### 18.3.2 Common electronic purse specifications

In the mid-1990s, many electronic purse systems based on smart cards were developed independently of each other in various European countries. Some typical examples are Quick in Austria, GeldKarte in Germany, and Proton in Belgium and the Netherlands. All of these purse systems have similar functionality, but they are mutually incompatible. The need to make these purse systems mutually compatible has become increasing compelling, in part due to the introduction of a common European currency in 2002. As all of the electronic purse cards in these systems are usually valid for a period of only three years, it is in principle possible to make gradual modifications to the purse systems over the course of several years using a defined migration path.

The fundamental prerequisite for mutual compatibility between several electronic purse systems is a document that specifies the required common features. This document bears the name ‘Common Electronic Purse Specifications’ (CEPS), and the first version was published in 1999 by CEPSCO [CEPSCO]. However, from the very start CEPS has focused on an internationally interoperable electronic purse system instead of a system tailored to European interests.

CEPS includes the usual functions of a modern electronic purse system, such as offline payment, online loading, and online currency exchange. Although CEPS is based on the EN 1546 European standard for electronic purses,<sup>12</sup> it includes several extensions and modifications. For example, in contrast to EN 1546 it uses RSA-based hierarchical certificates for the authentication of terminals and smart cards. Triple DES is recommended as the symmetrical cryptographic algorithm.

Like many electronic purse systems, CEPS is optimized for simple smart card microcontrollers. A typical CEPS implementation in assembly language or C requires 8 KB of ROM, 4 KB of EEPROM, 1 KB of RAM, and a numeric coprocessor for the asymmetric cryptographic algorithm. In the future, most European electronic purse systems will be compatible with CEPS, so in the reasonably near future it should be possible to make payments in various European countries using a single purse card.

### 18.3.3 Proton

Proton is an international electronic purse system that was originally developed by Bull and whose development dates back to 1995. It originated in Belgium and the Netherlands (where it is known under the brand name ‘Chipknip’), which is where it is most widely used and presently has the status of a national electronic purse system. There are also relatively large purse systems based on Proton in Switzerland and Sweden under the ‘Cash’ brand name. In the spring of 2002, there were approximately 40 million cards issued worldwide and around 360 000 terminals. Bull originally gave this electronic purse system the name ‘CC 60’.

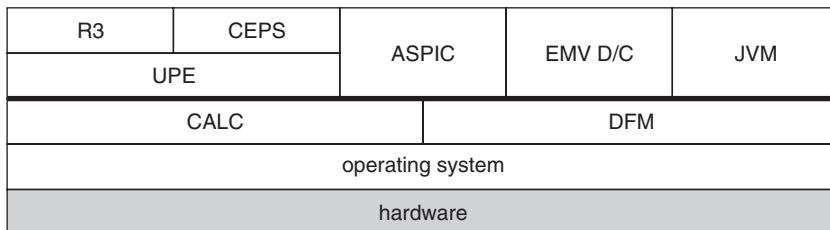
R3 is a purse system optimized for inexpensive smart card microcontrollers, and it can readily be implemented in chips with 16 KB of ROM, 2 KB of EEPROM, and 256 bytes of RAM. R4 is an extended version of R3 and is compatible with CEPS.<sup>13</sup> Beside the actual purse system, the R4 specification defines an operating system for multiapplication smart cards that supports debit and credit functions compatible with the EMV specification (see Figure 18.12),<sup>14</sup> a digital signature application, and Java functionality. There is also a contactless version of Proton, which is primarily intended for local public transport systems. The publisher of the specification, which is confidential, is Proton World [Proton] application smart cards that supports debit and credit functions compatible with the EMV specification,<sup>15</sup> a digital signature application, and Java functionality. There is also a contactless version of Proton, which is primarily intended for local public transport systems. The publisher of the specification, which is confidential, is Proton World [Proton].

<sup>12</sup> See also Section 18.3.1, ‘Inter-sector electronic purses compliant with EN 1546’, on page 760

<sup>13</sup> See also Section 18.3.2, ‘CEPS’, on page 774

<sup>14</sup> See also Section 18.4, ‘EMV Application’, on page 776

<sup>15</sup> See also Section 18.4, ‘EMV Application’, on page 776



**Figure 18.12** Schematic representation of the basic architecture of the Proton R4 smart card operating system. The following abbreviations are used here: CALC (card application life cycle), DFM (data file management), UPE (universal purse engine), ASPIC (application for secure personal identification = PKI functionality), EMV D/C (EMV debit/credit application), and JVM (Java virtual machine)

The electronic purse system includes the usual functions, such as loading, single payment, sliced (incremental) payment, and refunding a payment. The main use for sliced payment is public card phones, which require small amounts to be repeatedly debited from the purse balance at short intervals during a session. Transactions are stored in record-oriented log files. The purse parameters are also stored in files. To allow the system to be implemented in very inexpensive microcontrollers with relatively little memory, the files can be located directly below the MF without a DF. Both DES and triple DES are used as cryptographic algorithms in R3. Many of the smart card commands used in the system are based on or compatible with the ISO/IEC 7816-4 and EN 1546 standards. They are supplemented by several application-specific commands. The actual purse function is similar to standard EN 1546 processes in many respects.<sup>16</sup> However, it is obvious that Proton is several years older than EN 1546.

## 18.4 EMV APPLICATION

Specifications and standards for smart cards in a wide variety of application areas have been available for many years. However, there was traditionally a strong focus on telecommunication applications (telephone cards and GSM cards). A dramatic shift in focus began in the mid-1990s. The European electronic purse standard (EN 1546) is a good example of this, but the most important specification in the payment transactions area is the EMV specification, which is named after its three initiators (Europay, MasterCard and Visa) and contains detailed descriptions of all aspects of credit cards with microcontroller chips. There is also a related specification for the corresponding terminals.

In the autumn of 1993, the three internationally active credit card companies Europay, MasterCard and Visa started work on a specification titled ‘IC Card Specifications for Payment Systems’. At the end of 2004, the version number was increased to 4.1, which is the presently applicable version. It is available via the Internet [EMV], and it can be recommended without reservation to interested parties as a worthwhile subject of study.

Several factors motivated these credit card issuers to generate a specification for credit cards with chips within such a short time. First, existing credit cards with magnetic stripes can be counterfeited very easily. The only real obstacle now is the hologram, which is still

<sup>16</sup> See also Section 18.3.1, ‘Inter-sector electronic purses compliant with EN 1546’, on page 760

moderately secure against forgery. All other card features can be copied with relatively little effort. The second important factor is the additional functions that a microcontroller card can offer. Electronic purses, bonus points, and telephone functions are only some of the possibilities.

The specification for credit cards with chips is divided into four parts, which are called books. Book 1, ‘Application-independent ICC to terminal interface requirements’, draws heavily on the ISO/IEC 7816-1/2/3 family of standards. It describes the electromechanical properties, the logical interface and the data transmission protocol, which are the application-independent factors. According to this part of the specification, the smart card has an ID-1-format<sup>17</sup> with an ISO contact location. Among other things, it must have a supply voltage of  $5 \pm 0.5$  V, a maximum current consumption of 50 mA, and a clock rate of 1 to 5 MHz. The maximum contact force is 06 N per contact. Data transmission at the physical level is essentially the same as ISO/IEC 7816-3. This applies to the bit interval,<sup>18</sup> the ATR,<sup>19</sup> and the two transmission protocols ( $T = 0$  and  $T = 1$ ).<sup>20</sup> The specification of the APDU<sup>21</sup> is identical to that in ISO/IEC 7816-4.

The second part of the specification (Book 2, ‘Security and key management’) defines the security mechanisms and key management. Book 3, ‘Application specification’, defines the data elements, commands and processes necessary for debit and credit transactions in accordance with the EMV specification. Book 4, ‘Cardholder, attendant, and acquirer interface requirements’, contains many general user-related and merchant-related specifications for EMV-compliant payment systems.

As credit cards are typically mass-produced items, their production costs must of course be low. These costs predominantly depend on the implanted microcontroller, so the credit card application is specifically designed to minimize memory usage. A conventional EMV application without supplementary functions fits into a microcontroller with 48 KB of ROM, 4 KB of EEPROM, and 1 KB of RAM. Even in the smart card domain, these values represent the lower end of the scale, and they enable inexpensive production.

### 18.4.1 Files and data elements

The specification for credit cards with chips only stipulates that a tree structure must be used for the files.<sup>22</sup> The actual application is located in its own DF, which is selected using an application identifier (AID) and contains all the data elements for the credit card application. These data elements are held in EFs with standard file structures compliant with ISO/IEC 7816-4. EFs can usually be selected implicitly using short FIDs. The main difference with respect to similar specifications is that no particular EFs or FIDs are specified. This is anyhow not necessary for the functions used in payment transactions, since all the necessary data can be processed using the defined commands. There is only one file directly below the MF (EF<sub>DIR</sub> in accordance with ISO/IEC 7816-5), which holds all the data about the applications in the card.

<sup>17</sup> See also Section 3.1, ‘Card Formats’, on page 29

<sup>18</sup> See also Section 9.1, ‘Physical Transmission Layer’, on page 243

<sup>19</sup> See also Section 8.1, ‘Answer to Reset’, on page 203

<sup>20</sup> See also Section 9.3, ‘ISO Transmission Protocols’, on page 254

<sup>21</sup> See also Section 8.3, ‘Message Structure: APDUS’, on page 221

<sup>22</sup> See also Section 12, ‘Smart Card File Management’, on page 421

However, terminals often use the SELECT command to systematically try all AIDs available to them.

All data elements of the terminal–card system are specified using unambiguous templates and tags. They can be accessed in the application using the specified commands without any need to know their exact locations in the file tree or in a file. This allows the details of the file structure to be left to the card issuer, since they do not affect the payment transaction process.

### 18.4.2 Commands

In theory, only three commands are necessary for the actual payment process. Additional commands are needed for personalization, management, special functions, and supplementary functions, but these commands fall outside the scope of the EMV specification. Pursuant to the requirements of the EMV specification, the following commands must be available, with return codes similar to ISO/IEC 7816-4.<sup>23</sup>

- APPLICATION BLOCK (specific to EMV)
- APPLICATION UNBLOCK (specific to EMV)
- CARD BLOCK (specific to EMV)
- EXTERNAL AUTHENTICATE (as a subset of ISO/IEC 7816-4)
- GENERATE APPLICATION CRYPTOGRAM (specific to EMV)
- GET CHALLENGE (ISO/IEC 7816-4)
- GET DATA (specific to EMV)
- GET PROCESSING OPTIONS (specific to EMV)
- INTERNAL AUTHENTICATE (as a subset of ISO/IEC 7816-4)
- PIN CHANGE/UNBLOCK (specific to EMV)
- READ RECORD (as a subset of ISO/IEC 7816-4)
- SELECT (as a subset of ISO/IEC 7816-4)
- VERIFY (as a subset of ISO/IEC 7816-4)

### 18.4.3 Cryptography

The cryptographic mechanisms used in an application are naturally highly dependent on the associated general requirements. This can be seen especially well in the EMV application. A basic initial premise for the system design was that terminals do not necessarily contain security modules, depending on the system operator. This makes it impossible to use symmetric cryptographic algorithms, since the keys cannot be kept secret. The reasons for not using

<sup>23</sup> See also Section 11.15, ‘Commands for Credit and Debit Cards’, on page 405

security modules are that they would significantly increase the cost of the terminal and that worldwide key management for the terminals, some of which operate offline, would be very complicated and expensive.

In order to make smart card authentication by the terminal possible under these conditions, an asymmetric cryptographic algorithm must be used. EMV uses static unilateral authentication with card-specific keys.<sup>24</sup> This does not allow the card to check the authenticity of the terminal, but this is not essential in the EMV application because no debiting is performed in the card. The card only generates a transaction cryptogram for the terminal. This transaction cryptogram is not anonymous with respect to the cardholder, and it can be submitted to the relevant card issuer only by an authorized (known) merchant. This largely excludes most possible forms of fraud, since only authorized merchants known to the card issuer can exchange valid transaction cryptograms for money.

In principle, any desired cryptographic algorithm can be used, since both the associated data elements and the algorithm itself are unambiguously identified by TLV-coded data structures. Version 2 of the EMV specification allows either SHA-1 (as specified in FIPS 180-1) or the ANSI X9.30-2 hash function to be used.<sup>25</sup> The cryptographic algorithm used is not DSS, as might be expected with SHA-1, but instead the RSA algorithm (ANSI X9.31-1).<sup>26</sup> The key size can be adjusted by the card issuer, and it is indicated by a suitable code (signature ID). Small numbers, such as 3 or  $2^{16} + 1$  (the fourth Fermat number), are recommended for the public key in order to minimize computation time.

If the smart card has established an online connection to the background system, data transmission can be protected by secure messaging in accordance with ISO/IEC 7816-4.<sup>27</sup> A symmetrical cryptographic algorithm (triple DES) is used for this end-to-end communication. The fact that both the background system and the card can securely store the secret key is what makes system security possible in this case without any restrictions.

#### 18.4.4 System architecture and transaction processes

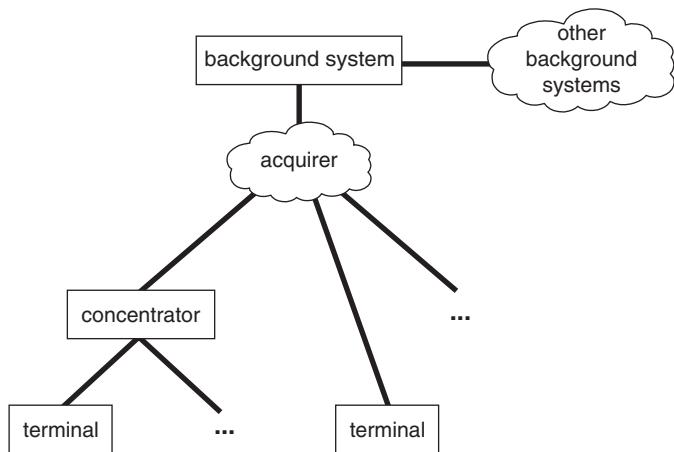
As illustrated in Figure 18.13, the system architecture of payment systems in the credit card sector is traditionally highly centralized. There are usually several background systems, which are individually or collectively responsible for a certain region (such as Germany). The computer centers for the background systems, which are equipped with high-performance computers, are interconnected by the independent network of the card issuer. This network supports data exchange for clearing and increases operational reliability, since if one center fails, its activities can be taken over by other centers. The individual terminals are connected to the background system via the public telephone system and data networks. An acquirer that bundles and routes transaction data may be located between the terminals and the background systems. However, this is strongly dependent on the individual card issuer and country. At the terminal level, there are two options: data may be exchanged directly with the acquirer, or a concentrator belonging to a merchant or chain of shops may be used. Both of these options are possible, and both are used in practice.

<sup>24</sup> See also Section 7.4.3, ‘Static asymmetric authentication’, on page 170

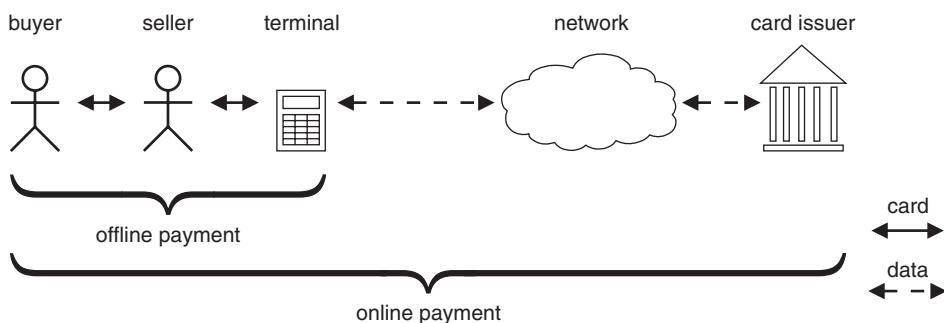
<sup>25</sup> See also Section 7.2, ‘Hash Functions’, on page 156

<sup>26</sup> See also Section 7.1.2, ‘Asymmetric cryptographic algorithms’, on page 145

<sup>27</sup> See also Section 8.4, ‘Secure Data Transmission’, on page 225



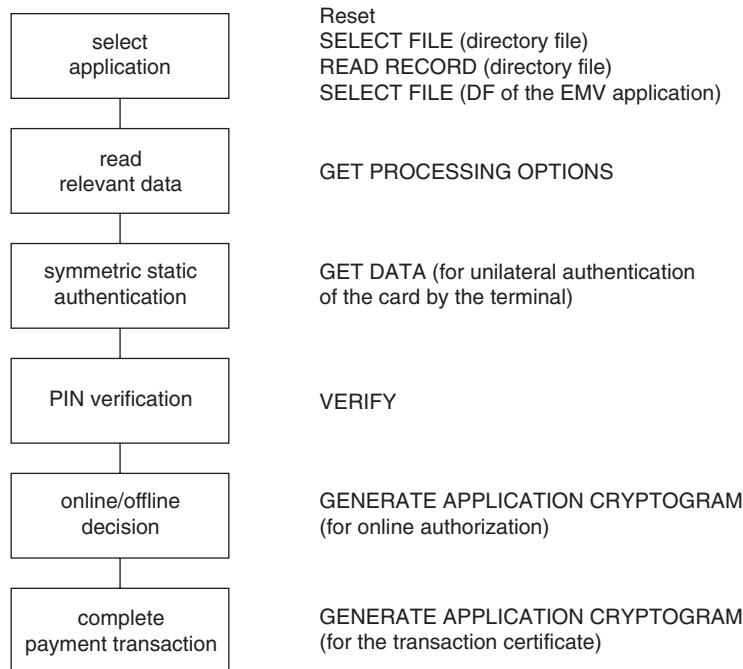
**Figure 18.13** Basic architecture of the payment system for EMV cards



**Figure 18.14** Basic infrastructure of a smart card payment process according to the EMV specification

The process of making a purchase with a smart credit card is essentially the same as making a purchase with a traditional credit card. The customer presents the card at the point of sale, and it is inserted into a smart card terminal. If a terminal is not available, payments can be made in the conventional manner using the magnetic stripe or embossed characters, which are still present on the card. Even if a terminal is present, it is still possible to verify the identity of the card user by means of a signature. In this case, the associated transaction receipt has a code that indicates that the card user was identified in this manner. The other option is for the card user to enter a four-digit PIN. This can be checked online by the background system or offline by the smart card. If a PIN is used for identification, the transaction cryptogram indicates which type of PIN check was performed. The payment process is shown graphically in Figure 18.14.

Of course, the individual functions and the payment transaction process using a smart card are somewhat more complicated. Figure 18.15 on the facing page shows an example that illustrates the basic mechanisms. The card and the terminal determine the exact course of the transaction based on various transaction data, such as the amount involved. For instance, if the purchase amount exceeds a certain sum, the card requests online authorization of the



**Figure 18.15** A highly simplified smart card payment process according to the EMV specification, as seen by the terminal

purchase by the background system. The terminal must then establish a link to the background system and have the purchase approved. Only after the background system has approved the request does the card generate a valid transaction cryptogram, which the merchant can submit for clearing. The purpose of online authorization is to minimize the financial risk to the card issuer. Since a smart credit card must regularly report to the background system in accordance with a number of conditions, the maximum loss in the event that a card is stolen or manipulated can be held within precisely calculable limits. However, an online terminal can also block an EMV card for subsequent payment transactions by sending the APPLICATION BLOCK or CARD BLOCK command to the card. Naturally, both of these commands require suitable prior authentication.

### 18.4.5 Future developments

In terms of its structure and contents, the EMV specification allows the card issuer considerable room for individual initiative, which makes further enhancements and individual versions possible. This flexibility will doubtless be utilized extensively by various companies. For instance, each of the three credit card issuers involved in preparing the EMV specification has generated specifications for one or more electronic purses<sup>28</sup> that can be included in the

<sup>28</sup> See also Section 18.3.1, 'Inter-sector electronic purses compliant with EN 1546', on page 760

smart card as necessary. Particularly with vending machines and small purchases, a prepaid electronic purse has distinct advantages because it avoids the fee that is always charged with a credit card payment.

The EMV specification has achieved the same significance in the payment transactions sector as the TS 51.011 specification for smart cards in the telecommunication sector. All new smart card operating systems for payment transaction applications aim to be EMV-compatible. Due to the anticipated number of smart credit cards in the future, this specification effectively sets the minimum standard for all future applications.

The EMV specification describes the interfaces to the outside world of the smart card operating system and the underlying mechanisms for applications. Visa and MasterCard generated supplementary specifications for their own applications: Visa Smart Debit/Credit (VSDC) and MChip. These applications were not fully compatible with each other, which led to the development of the Common Payment Application (CPA) specification. CPA established a common basis for future versions of EMV payment transaction applications. This specification is public and can be downloaded from the EMV website at [EMV].

## 18.5 PAYPASS AND PAYWAVE

Contactless smart cards are somewhat more convenient for users than contact cards, and they allow noticeably shorter transaction times. These are among the reasons why contactless smart cards are used almost exclusively in the public transport sector. Card issuers and system operators also wish to offer their customers these advantages with payment cards. At the start of the present millennium, MasterCard and Visa, as well as other parties, conducted several field trials with the objective of expanding existing systems to include contactless cards. Naturally, the focus of these trials was on credit cards, with the aim of making their basic functions available with contactless cards by means of this evolutionary development.

Visa's product for this purpose is called payWave, while MasterCard's product is called PayPass. From a technical perspective, these two products are quite similar. Their operating principle is based on contactless smart cards compliant with ISO/IEC 14443 Type A or Type B, which transfer all data necessary for the financial transactions to the terminal using cryptographic protection. The transferred data is essentially the same as the data on tracks 1 and 2 of the magnetic stripe, which is processed in the same form with magnetic-stripe cards. The main advantage of this is that almost no modifications are necessary in the system. It is only necessary to equip the terminals with a supplementary contactless interface. To perform a payment transaction, the contactless card must be held next to this interface, which customers know by the name 'landing zone'. The terminals must support both ISO/IEC 14443 Type A and Type B, while individual cards only have to support one of the two transfer methods.

The transaction process is virtually the same as with magnetic-stripe cards. After the application is selected with the SELECT command and its AID, the terminal reads the necessary data, which corresponds to the data on tracks 1 and 2 of the magnetic stripe. This data is secured with a dynamically generated MAC to ensure the authenticity of the smart card and the data. The terminal sends the received data to the background system, which authorizes the transaction if everything is OK. After this the customer receives the goods or service. PIN entry by the user is normally required above a certain threshold value, such as €20.

This form of contactless payment has a rather simple structure from a technical perspective, but requires only minimal technical changes to existing systems and thus minimizes the investment cost. In addition to good user acceptance of contactless smart cards, this is the

main reason for the steadily increasing worldwide presence of systems such as payWave and PayPass.

## 18.6 THE EUROCHEQUE SYSTEM IN GERMANY

With regard to card-based payment transactions, Germany differs from other countries in that direct debit cards (formerly called ec-Karte, now called Girocard) have traditionally been used much more than credit cards. This type of card can be used in many locations to make purchases by entering a four-digit PIN, with the amount of the purchase being debited immediately from a bank account linked to the card. The merchant must pay a fixed fee for each payment transaction, but it is not especially high. The acceptance of credit cards is only moderate in Germany because the credit card fee is a percentage of the purchase amount. There is also a widely used variant of the direct debit method called *POS ohne Zahlungsgarantie* (POZ), in which the customer agrees to have the purchase amount transferred from his bank account to that of the merchant by a direct debit process. Here the Girocard only provides a reference to the customer's bank account, which is checked online to see whether the balance is sufficient to cover the purchase amount. However, in this case the merchant does not receive a payment guarantee from his bank, as he would with a credit card transaction or a Girocard transaction (ec-Cash or GeldKarte).

In 1993, the *Zentraler Kreditausschuss* (ZKA), a working group of the national associations of the German banking industry, issued a call for design submissions for a 'multifunctional chipcard'<sup>29</sup> (MFC) that would be suitable for electronic payment systems. Several companies submitted designs, and one of them was selected and granted the rights to the design by the ZKA. Due to changes in the general technical requirements, extensive revisions to the design were made afterward. As a result of these revisions, there is now a family of specifications for Girocards with chips, each of which addresses a particular area. Unfortunately, these specifications are confidential, so it is not possible to publish detailed information. Here we can briefly describe the following topics:

- SECCOS operating system
- GeldKarte
- ec-Cash with chip
- digital signatures
- EMV
- electronic tickets
- electronic marketplace
- personalization

The specifications describe a payment card whose functionality corresponds to that of the current Girocard and includes a supplementary electronic purse. It is also possible to load any desired new applications in the card after it has been issued.

<sup>29</sup> The term 'multifunctional chipcard' was coined at this time and in this context

Prior to the nationwide launch of the new cards in Germany, a large-scale field trial with approximately 100 000 cards and 500 terminals was conducted in the spring of 1996 in the Ravensburg and Weingarten region (near Lake Constance), which has 250 000 inhabitants. Following this, wide-scale distribution of the cards throughout Germany began in the fall of 1996.

Around 68 million Girocards with chips have now been issued in Germany, all of which must be replaced every four years. The total number of installed terminals in Germany was 250 000 in 2006, and 900 million transactions take place each year with these terminals. The combined turnover of magnetic-stripe and chip-based transactions with ec-Cash and GeldKarte was approximately €55 billion in 2006. Almost all of the 55 000 automated cash dispensers in Germany are equipped with a smart card terminal. This field trial was thus the precursor to one of the largest smart card payment systems in the world.

The figures for GeldKarte [GeldKarte] in 2007 provide a clear indication of the current extent of use of this electronic purse system. With 68 million issued smart cards, the GeldKarte application booked 52.8 million payment transactions with a turnover of €148 million. This was less than 0.1 % of the total turnover in the retail trade sector during this period. There are more than 600 000 acceptance facilities for GeldKarte in Germany, although 460 000 of these are cigarette vending machines. Due to an amendment to the German Youth Protection Act that took force in early 2007, GeldKarte is used with cigarette vending machines to verify the purchaser's age, which is why its use is concentrated in this area. There were a total of 6.5 million loading transactions in 2007, and the total amount loaded into the GeldKarten was €172 million.

### 18.6.1 User functions

The German Girocard with a chip has a variety of user functions. It can be used to make purchases online or offline after entering a PIN code in a suitable terminal. The purchase amount is then debited from the associated account by the bank that issued the card. This application is generally referred to as 'ec-Cash' in this chapter, although there are also other names for it.

Naturally, a Girocard with a chip can also be used to obtain cash from the cash dispensers of various banks, but in functional terms this belongs to the realm of ec-Cash. Naturally, the smart card supports a variety of lobby-machine functions, such as printing an account statement. It also has a prepaid electronic purse called GeldKarte, which can be used to make purchases without entering a PIN code. This purse is available in anonymous and nonanonymous versions. It can be reloaded using suitable loading terminals at locations such as bank counters and cash dispensers, either against cash payment or with a corresponding ec-Cash transaction.

An EMV-compliant credit card application in accordance with the CPA is also often implemented, and many cards provide an electronic signature function. For card-based home banking, a Home Banking Computer Interface (HBCI) and a TAN generator for online transactions are present in some cards. Some cards also include the electronic ticket and marketplace applications. The latter application acts as a collector for simple value-added applications such as bonus point systems and concert tickets.

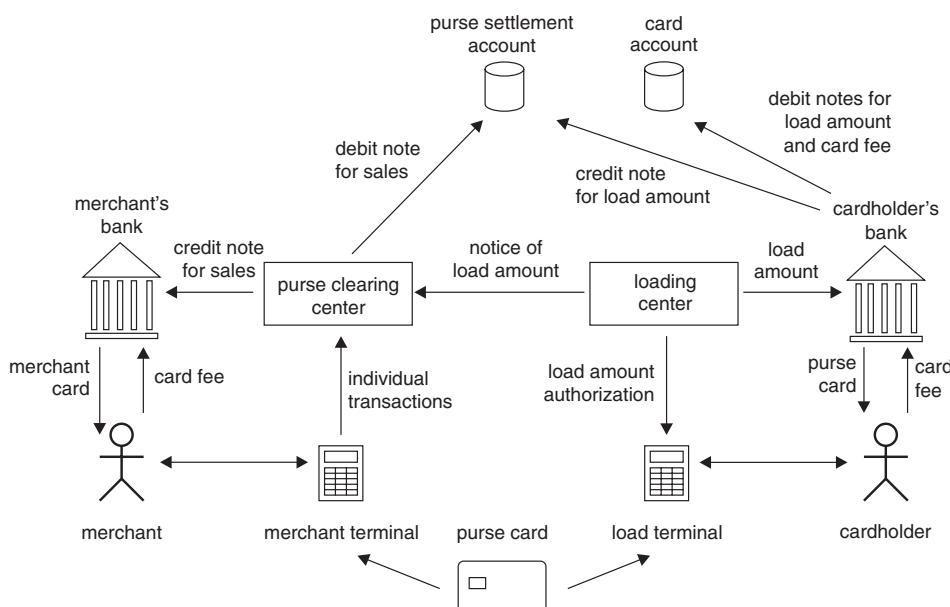
Another basic function supported by the Girocard is downloading supplementary applications with all of their associated files and commands after the card has been issued to the

cardholder. However, this function has seen little use up to now because the coordination needs and approval conditions for new applications are relatively demanding.

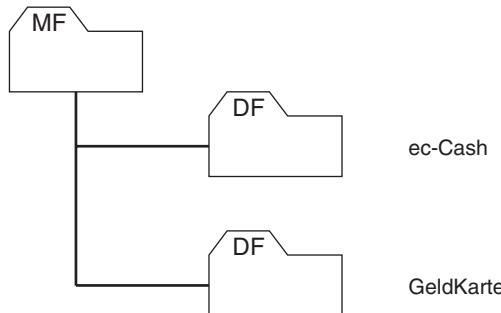
### 18.6.2 The overall system in brief

As usual with large payment systems, there is no central clearing system for German Giro-cards with chips. There are four computer centers that clear transactions between merchants, cardholders, and the participating banks. Figure 18.16 provides an overview of clearing for the nonanonymous version of the electronic purse application. The clearing entity in Germany is called the *Börsenevidenzzentrale* (BEZ). There are also approximately 25 loading centers that perform loading transactions for GeldKarte cards in coordination with the BEZ.

All transactions are based on two accounts: the purse settlement account, which always reflects the current balance of the electronic purse in the card, and the card account, which for example may be the cardholder's current account. The purse settlement account is a shadow account that is maintained in parallel with the electronic purse. When money is loaded into the electronic purse, a corresponding booking is made to the purse settlement account (shadow account) at the same time, since this transaction must always be performed online. Depending on whether the payment transaction occurs online or offline, the purchase amount is booked to the purse settlement account, either at the same time or later. The main advantage of this account-linked system is that the purse balance can be reconstructed (with a certain time delay) if the electronic purse is lost or becomes unusable. Of course, with this approach it is not possible to ensure that payments are anonymous. However, there is also an anonymous



**Figure 18.16** Basic structure of the payment system for the German GeldKarte, which is an account-linked electronic purse system and thus not anonymous



**Figure 18.17** Basic file tree of a German Girocard smart card with the ec-Cash and GeldKarte applications

version of the electronic purse that uses a shadow account with no reference to a customer account. This is called a prepaid card.

### 18.6.3 Girocard with chip

Several types of cards are used in the German Eurocheque system. The smart cards for normal bank customers can be further classified into several categories:

- Girocard (ec-Cash and GeldKarte linked to an account and thus nonanonymous)
- GeldKarte linked to an account (nonanonymous)
- GeldKarte not linked to an account (anonymous)

As shown in Figure 18.17, a Girocard has two DFs, one of which holds the ec-Cash application and the other the GeldKarte application.<sup>30</sup> The DF for ec-Cash is not present in cards with an electronic purse. The account link, which determines whether the card is anonymous or nonanonymous, is only provided by certain data elements.

There is a merchant card in ID-000 (plug-in) format for use in merchant terminals. It contains all of the commands and files necessary for processing payment transactions. This can be regarded as the security module of the merchant terminal. One of the unusual and technically interesting features of this system is that it has both real and virtual merchant cards. A real merchant card is a normal smart card in plug-in format. A virtual merchant card is simply a software simulation of a real merchant card, which runs in the protected environment of a security module (SAM) in a merchant terminal.

This solution was originally an expedient to allow terminals without sockets for plug-in cards to be used in the new system. However, it has proven to have some very positive technical features. For instance, a virtual merchant card can easily be replaced by remote maintenance because it consists only of software. Ultimately, a good hardware security module is at least as secure as a smart card because its security mechanisms are always active thanks to its built-in power source.

<sup>30</sup> Unfortunately, the term GeldKarte must always be understood in context. It can refer to an electronic purse application in a smart card or the smart card itself

The entire information technology concept and the security model of the card are strongly based on the ISO/IEC 7816 family of standards. The original version, which is now called Type 1 GeldKarte, included several application-specific mechanisms, but they have been eliminated in more recent versions. A full-fledged smart card operating system with PKI functions called Security Card Operating System (SECCOS), which supports all of the essential mechanisms of the ISO/IEC 7816 standard, has now been specified. The security and access mechanisms of ISO/IEC 7816-9 are included in a very elaborate form, with the result that at present the German Girocard smart card application is probably the most complete implementation of this standard anywhere in the world. For compatibility reasons, the Girocard specifications also reflect some elements of the EMV specification.

In terms of the general technical parameters stipulated by the specification, the card is in line with many existing standards. Naturally, the dimensions correspond to the ID-1 format and are thus the same as the current Girocard. In addition, it is designed as a hybrid card with both a chip and a magnetic stripe in order to avoid compatibility problems during the transition from currently standard terminals with magnetic-stripe readers to new terminals with smart card contact units. The card uses the T = 1 transmission protocol with PPS and the extended length option, which up to now has rarely been used in practice. The cryptographic algorithm is triple DES with SHA-256, SHA-384 and SHA-512. One of the interesting security aspects is that the entire software and hardware of the smart card must have security certification in accordance with the ZKA criteria catalog.<sup>31</sup>

The file management system supports several levels of DFs, as well as file selection using short FIDs, FIDs, and AIDs. The usual file structures (transparent, linear fixed, linear variable and cyclic) are supported, and files can be selected implicitly by the appropriate commands. The maximum size of the two record-oriented file structures is 254 records of 255 bytes each.

The file management system uses a special mechanism to associate EFs with specific applications. This function, which is implemented using two nonstandardized commands, allows EFs to be assigned to applications across DF boundaries. This makes it possible to use a short FID within a particular DF to select an EF located in a different DF. Consequently, a single EF can be associated with several different DFs. This corresponds in principle to the alias mechanism in many PC operating systems. The objective is to make EFs containing general information available to several applications across application boundaries without using complicated selection processes. The EFs associated with several applications in this manner can be selected using a short FID or an FID and then read or written if the necessary security state has been attained. All files have specific access conditions, and this makes reading and writing dependent on previously attained states (such as successful PIN entry). With this object-oriented system, it is also possible to make file access dependent on the use of secure data transmission. This means that there are file attributes that can compel the use of secure messaging for any access.

The available commands can be divided into four classes. The first class consists of commands compliant with ISO/IEC 7816-4, although they have reduced functional scope compared with the standard. The second class consists of Eurocheque-specific administration commands, which are used for management purposes in the card. They can be used to create new files, to delete existing files, and to add new commands to the card. The third class is extension commands, which are used to achieve the functionality needed for the ec-Cash and GeldKarte applications. The administration and extension commands are entirely specific to Eurocheque

<sup>31</sup> See also Section 15.5, 'Evaluation of Hardware and Software', on page 659

cards, and in principle they have no relationship to international standardization. The fourth class consists of the initialization and personalization commands.

As can be seen from this brief description, Girocards have a relatively large scope of functions. This unavoidably results in a large memory requirement. Accordingly, the presently used target hardware largely consists of microcontrollers with 200 KB of ROM, 40 KB of EEPROM, and 6 KB of RAM. The net consequence of these memory requirements is that relatively large chips are needed to hold the large volume of program code, along with 1.8 KB of application data for the GeldKarte electronic purse, 1.6 KB for ec-Cash, 2.5 KB for CPA, and 20 KB for the digital signature function.

#### 18.6.4 Supplementary applications

The Girocard operating system includes commands and mechanisms for downloading executable program code. However, this code must be tailored to the specific microcontroller and operating system that is used, since only address-dependent machine code can be downloaded. The resulting level of logistical effort for downloading new commands is the main reason why this mechanism is presently not used.

However, supplementary applications do not necessarily require loading a program in the smart card. In most cases, it is only necessary to have files available with suitable access privileges. The Girocard specification includes commands for creating files. However, the administrative effort necessary to implement supplementary applications in individual cards online via a clearing center is very large, so this mechanism is very rarely used.

Instead, all of the necessary files necessary for installing new applications in the cards after they have been issued are created in the Girocards when they are personalized.

#### 18.6.5 Summary

The German Girocard system is presently one of the largest and most complex payment systems using smart cards. This includes not only the transaction processes, but also the underlying logistics of semiconductor fabrication, card personalization, and card issuing. After all, every four years approximately 30 million cards must find their way into customers' hands in less than three months.

The security evaluations of the microcontroller hardware, operating system software and application software also set new standards because the acceptance criteria are strict and are constantly adapted to new circumstances, such as DFA, SPA/DPA and so on. Another interesting aspect is the technically sophisticated compatibility tests, which are intended to ensure that the software implemented with different masks in different microcontrollers works smoothly with a wide variety of terminals.

The sheer volume of 68 million smart cards in the German Girocard system affects all large payment system projects based on smart cards. The experience gained in Germany from using these multiapplication smart cards will act as a stimulus in many areas for considerable enhancement and refinement of future generations of smart card operating systems and the associated microcontroller hardware.

# 19

## Smart Cards in Telecommunication Systems

The success of microcontroller cards is closely linked to the success of mobile telecommunication technology. This is the smart card application area with the largest volume of cards worldwide and the largest functional scope. Only a few technologies up to now have achieved such rapid acceptance and worldwide penetration as the GSM mobile communication standard. For smart cards, this rapid development was the perfect driver for advancing the technology and enlarging the market at a rapid pace.

### 19.1 PUBLIC CARD PHONES IN GERMANY

Starting in the summer of 1989, the German state telephone company Telekom began introducing public card phones throughout the entire country after several field trials had been carried out with systems from various manufacturers. These field trials began as early as 1983 in four regions in Germany with different characteristics (conurbations, cities, and rural areas). Several types of cards were tested in these trials, including magnetic-stripe cards, hologram cards, and cards made from different materials (paper and plastic).

The conclusion from these initial field tests was that plastic memory cards were the best option for use with card phones. The decisive factors were the achievable level of security and upward compatibility relative to other types of cards. In December 1986 and July 1987, large-scale field trials of a system based on memory cards began in sixteen major cities. They were successfully concluded in May 1989.

The card phones installed by Telekom in Germany can in principle be used with two different types of cards. The first type is prepaid phone cards, which are produced and used in very large numbers. In financial terms, a prepaid card is sort of prepaid electronic purse. In technical terms, it is a memory card with an irreversible counter, a security feature, and synchronous data transmission. The other type of telephone card, which is no longer available,

is the charge card. It resembles a credit card in that the user pays for the services (telephone calls) only after they have been received, by monthly debiting of the user's bank account. In technical terms, a charge card is a smart card with a microcontroller that uses the block-oriented, asynchronous T = 14 transmission protocol for data transfer to the card phone. As this type of card never achieved a significant market presence, it is not discussed any further here.

The overall card phone system is designed as a decentralized structure with several successive layers of computers. In the event of a system crash, the lower levels can operate fully autonomously for several days without any impairment of telephone service. The two elements of the system that the normal user sees, which are the phone cards and the card phones, are described below.

In order to survive as long as possible under the severe operating conditions to which it is exposed, a card phone has a sturdy metallic enclosure with openings for the controls and indicators, such as the keypad and display. The terminal is galvanically isolated from the rest of the electronic assemblies. It is also short-circuit-proof to protect it against potential vandalism and attempts to destroy the terminal.

The card phone is controlled by a high-performance microcontroller. The control software can be updated remotely, and the control computer of the card phone can exchange data directly with the higher levels of the system during a conversation by using a data-over-voice (DOV) modem.

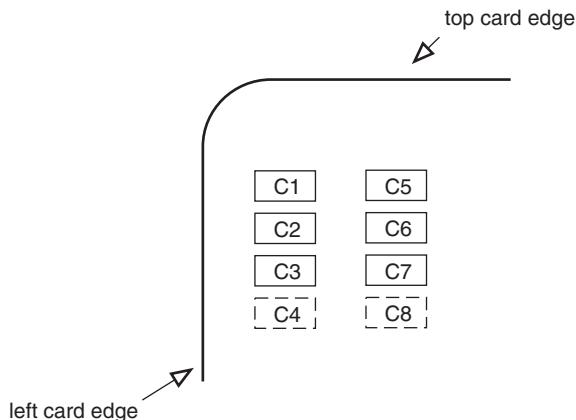
The built-in terminal can supply the memory card with an external programming voltage adjustable between 5 and 25.5 V in 255 steps. However, practically all new memory cards have an internal voltage converter that generates the programming voltage from the normal supply voltage, so this capability is no longer necessary. It is only present for compatibility with older generations of phone cards containing balances that are still valid.

If a microcontroller card is used, the terminal can select a clock frequency in the range of 1.2 to 9.8 MHz. With synchronous memory cards, the clock frequency must be reduced to around 20 kHz to establish a usable communication link. The terminal has a massively constructed shutters for protection against dummy cards. The shutter has an impact blade that cuts through any wires or cables extending through the card slot. This prevents tapping or manipulating the communication between the card and the terminal.

The chips used for phone cards have ROM that can be mask-programmed by the semiconductor manufacturer as well as EEPROM memory for card-specific data and the card balance counter. A charge pump on the chip generates the programming voltage for the EEPROM, so it is no longer necessary to supply this voltage externally. This means that the chip needs only the normal operating voltage. For protection against fraud using counterfeit cards, the chip has a hardware security feature whose operation is secret. In the future, memory chips that enable unilateral authentication of the card by the outside world will also be used.

Most modern phone card chips have only six contacts. This is because only five contacts are actually needed for the full functionality of the memory chip, with all other contacts being unused. Using eight contacts would increase production costs because the module would be larger and thus more expensive. It would also take longer to mill the cavity for the module in the card body, thus reducing the throughput of the production equipment and increasing the unit cost. Consequently, practically all new phone cards have only six contacts. The signal assignments of the six or eight leads brought out from the chip to the contacts are shown in Figure 19.1 on the facing page.

The chip memory in prepaid Telekom phone cards contains the data described below.



**Figure 19.1** Contact assignments of a phone card. The unlisted contacts are not used. C1: supply voltage ( $5 \text{ V} \pm 5\%$ ); C2: control input; C3: clock input (approx. 20 kHz); C5: ground; C7: data transmission

### Serial number

Each phone card has a seven-digit serial number. This allows individual cards to be blocked if there is suspicion of fraud. The serial number is stored in a portion of the EEPROM that is blocked against overwriting or erasing.

### Date of manufacture

When the chip is manufactured, the year and date of its fabrication are permanently written to the EEPROM. This date refers only to the implanted chip, not the card.

### Producer

After the chip has been implanted in the card body, the card producer writes a specific code to the memory of the card. This code, which is also nonalterable, identifies the producer.

### Initial value

In addition to the current value, the initial value of the card is stored in every phone card.

### Remaining balance

This is the only data element in the card that is functionally necessary for prepaid phone cards. The card phone successively deducts individual billing units from a five-digit irreversible octal counter in the EEPROM. The counter thus consists of five bytes of eight bits each, which yields a maximum count of 32 768 ( $8^5$ ). During card completion, the counter is set to the desired initial value of the card, which means that the initial count is equal to the value of the card in eurocents. After this, the counter can be decremented toward zero during use. When the counter reaches zero, the card is empty.

### Transaction process

When a phone card is inserted in a card phone, the shutter first closes the card slot, while at the same time the terminal's contacts are applied to the contact surfaces of the card. The

card phone then executes an ISO activation sequence<sup>1</sup> to activate the card. Following this, the terminal sets the address pointer in the memory card to zero and attempts to read the first 16 bits. If this is successful, the phone card is functional. The 16 bits are interpreted by the terminal as a sort of ATR containing various operating parameters for the card. The terminal checks these parameters to determine whether the inserted card is one of the allowed card types whose functions it can actually use. The card number, which the terminal has also read from the card, is at the same time sent via the DOV modem to the next higher level of the system, where it is compared with the blacklist of blocked cards. If the blacklist has no entry for the card, it can be used, and the card phone receives a corresponding message.

The address pointer is now set to the charge counter region, and the current balance of the card is read from the EEPROM and shown on the display. After the user has entered the telephone number and a connection has been established, the units to be deducted from the card balance are sent periodically to the card phone via the DOV modem. Each time this occurs, the card phone decrements the counter in the memory card and immediately reads its new value. This allows it to check whether the amount was correctly deducted. If this is not the case, the connection is immediately broken. If the card phone calculates that the remaining balance in the card will be used up within the next 20 seconds, it generates a warning tone, and the card can be exchanged for a new one without interrupting the conversation.<sup>2</sup>

## 19.2 TELECOMMUNICATION

Telecommunication, or communication technology, is the technology used to exchange messages between persons and/or machines over arbitrary distances. Over the centuries, it has developed from human messengers and visual signaling techniques to communication using electrical circuits and radio signals. The pioneers in this field were primarily military organizations and businessmen for whom it was a matter of survival to be able to convey messages as quickly as possible. Nevertheless, the biggest growth spurt in telecommunication only came in recent years during which it became a true mass application as the result of a variety of favorable conditions, such as market deregulation, inexpensive manufacturing of terminal equipment, and a general economic and technology boom.

Although the proposals for using electricity to convey messages were published as early as 1753, the first telegraph line between Paris and Lille did not go into service until 1794. In the following decades, the efforts of numerous individuals in many countries led to the development of inventions for conveying messages using electricity. The most important and most interesting of these inventions were the design of the electrochemical telegraph by S. Soemmerring in 1809, the invention of induction telegraphy by Carl Friedrich Gauß and Wilhelm Weber in 1833, and the design of the recording telegraph by Samuel Morse in 1835, along with the publication of the Morse code in 1840.

However, all of these inventions and designs, many of which were ingenious, could only transmit coded electrical signals instead of human speech. Speech transmission first became possible around 1860. As so often with major inventions, it is not possible to credit a single person with the invention of the telephone. Still, there were essentially two persons who

<sup>1</sup> See also Section 4.6, ‘Activation and Deactivation Sequences’, on page 70

<sup>2</sup> See also Section 18.2, ‘Prepaid Memory Cards’, on page 757 for a description of the devaluation cycle in memory cards

publicly demonstrated devices that could transmit speech using electrical signals: Johann Philip Reis, a teacher at a school for the deaf, and Alexander Graham Bell, a teacher of the deaf and dumb.<sup>3</sup> Both of them originally conceived their inventions as teaching aids for helping deaf people learn to speak. The major strength of Alexander Graham Bell was his aggressive marketing of his invention, with the result that he came to be the better known of the two inventors of the telephone.

The first local telephone network was constructed in 1878 in New Haven in the USA. In Berlin, the first local public telephone system was put into service in 1881. It had human operators, and naturally it had all the characteristics of the simple electronic technology of the time. The first automatic telephone exchange in Germany, using rotary lifting selector switches, was installed in 1908 in Hildesheim. This marked the start of the worldwide installation and interconnection of what is now known in technical terms as the public switched telephone network (PSTN).

After Guglielmo Marconi demonstrated the first wireless transmission of data over a distance of several kilometers in 1896, it took only five years for the Atlantic Ocean to be bridged by wireless telegraphy. The invention of amplifier tubes (valves) stimulated the technical development of transmitters and receivers for voice communication, with the result that the first successful transmission of speech over the Atlantic Ocean took place in 1915. The first picture phone was presented at the 1929 radio exhibition in Berlin. It had the dimensions of three modern-day telephone booths.<sup>4</sup>

The technological evolution of telecommunication and mobile radios accelerated in the following years, leading to the inauguration of the first mobile telecommunication network at a relatively early date.

The first public land mobile networks (PLMNs) came into being around 1950, at which time they were reserved for a small and above all wealthy class of users. For example, the first public mobile telephone network in Germany (the A-Netz) started operation in 1958. It had human operators and used analog signals in the 150-MHz band. With a geographic coverage of approximately 80 % of the Federal Republic of Germany, it had a maximum capacity of 10 500 subscribers. This first mobile telephone network in Germany did not have a cellular architecture, but was instead based on a few distributed high-power transmitters and sensitive receivers within whose coverage area it was possible to make mobile telephone calls. It was also necessary to know the approximate location of the mobile telephone subscriber when placing a call to the subscriber, so that the proper radio zone could be selected by the human operator. The A-Netz remained in operation until 1977, when it was replaced by the B-Netz and subsequently the C-Netz, both of which were also analog systems.

At the beginning of the 1980s, mobile telephone networks with cellular architectures simultaneously came into being in many parts of the world. However, these systems were mutually incompatible, and due to the cost of the terminal devices (mobile telephones) and high charge rates, they were aimed at relatively well-to-do customers. These networks are now designated as first-generation (1G) mobile telecommunication networks. Although they

<sup>3</sup> According to an anecdote that is now impossible to verify, the first words that Johann Philip Reis transmitted by telephone in 1863 were *Das Pferd frisst keinen Gurkensalat* ('Horses do not eat cucumber salad'). He presumably chose this somewhat remarkable sentence to ensure that the transmission of human speech actually worked. A phrase that the person at the other end could easily guess, such as the presently popular welcome message 'Hello world', would not provide this assurance

<sup>4</sup> A good condensed summary of the history of historical telecommunication techniques can be found in (among other sources) *Nachrichtentechnik* by Oskar Blumtritt [Blumtritt 97]

were cellular systems, data transmission over the air interface was still analog. Subscribers were identified by personalized mobile telephones, which meant that each mobile telephone was directly and permanently linked to a particular subscriber. One of the first 1G systems to support the use of cards was the German C-Netz, which is also designated C-450). Magnetic-stripe cards were used in the first C-Netz mobile telephones, but they were quickly replaced by smart cards. This broke the bond between the terminal (telephone) and the subscriber, so personalization of the telephones was no longer necessary. As a consequence, telephones (which are very expensive compared with smart cards) became readily interchangeable mass-production items. Due to the resulting strong competition, the prices of mobile telephones decreased rapidly, which served the interests of the network operators with regard to having the largest possible potential market.

In the early 1980s, the European national telecommunication authorities wanted to transform Europe's many heterogeneous, country-specific, first-generation mobile telecommunication systems into a set of homogeneous systems. The intention behind this was to make it technically possible to use the same mobile telephone in more than one country, and especially to achieve substantial reductions in the prices of system components and terminals for a uniform European system as a result of significantly larger production volumes.

The end result of several years of specification work was a standard for the international Global System for Mobile Communication (GSM).<sup>5</sup> The first GSM systems underwent trials in 1991 and were put into regular service in 1992, and they quickly spread far beyond their original European boundaries. However, in many parts of the world there were other types of mobile telecommunication systems that were incompatible with GSM. Already in 1985, the International Telecommunication Union (ITU) [ITU] had commenced its foresighted efforts to achieve worldwide standardization and functional extension of all existing mobile telecommunication systems. Its concept was called the Future Public Land Mobile Telecommunication Service (FPLMTS). However, this concept was unable to produce an internationally standardized solution as originally intended, so in 1995 it was transformed into the IMT-2000 concept (International Mobile Telecommunication at 2000 MHz). IMT-2000 provides significantly more scope for implementation options, and it now forms the basis for the current UMTS and other third-generation mobile telecommunication systems.

It is common practice throughout the world to classify the technology of mobile telecommunication systems by using a generation number. These generations are counted starting at '1', and they include only cellular networks. According to the logic of this scheme, early types of mobile telecommunication networks such as the German A-Netz and B-Netz belong to generation zero, but this designation is not commonly used. The designation 'first generation' (abbreviated as '1G') applies to cellular mobile telecommunication networks with an analog air interface. Some typical examples of 1G networks are AMPS and the German C-Netz. A second-generation (2G) system is understood to be a cellular mobile telecommunication network with digital data transmission on the air interface. The two most widely used 2G systems are GSM and CDMA. Systems with functional extensions to GSM such as GPRS (General Packet Radio System) and EDGE (Enhanced Data Rates for GSM and TDMA Evolution), which are intermediate steps on the way to third-generation systems, are typically called 2.5G systems. The third generation (3G) encompasses cellular mobile telecommunication networks with digital air interfaces that provide major enhancements in mobile data communication

<sup>5</sup> See also Section 19.4, 'The GSM System', on page 802 for a brief history of the development of GSM

capability and Internet-compatible services compared with 2G systems. Typical 3G systems are UMTS and CDMA 2000, both of which are members of the IMT-2000 family.

The sections of this chapter devoted to the GSM and UMTS mobile telecommunication systems describe the smart card aspects of the development of these systems in more detail.

Efforts to develop concepts for fourth-generation (4G) mobile telecommunication systems are already underway. Future telecommunication systems will doubtless have significantly higher bandwidth efficiency than present systems, since frequency spectra are a limited resource and in some countries a very expensive resource due to the auctioning of spectrum allotments. In the GSM system, a frequency bandwidth of approximately 5 Hz is needed for a transmission bandwidth of 1 bit/s, yielding a frequency efficiency of 0.2 bit/s per Hz. In the UMTS system, this is improved to approximately 1 bit/s per Hz, and some systems in the research stage have already achieved efficiencies of up to 30 bit/s per Hz.

There are many ideas and proposals for 4G systems, but they have not yet achieved a confirmed technical status or a uniform development direction. Nevertheless, the basic features of the two main requirements have already been established: international interoperability of the mobile equipment and high data transmission rates when necessary. Interestingly enough, these two desires were already stated in the mid-1980s as major requirements for FPLMTS and IMT-2000.

## 19.3 OVERVIEW OF MOBILE TELECOMMUNICATION SYSTEMS

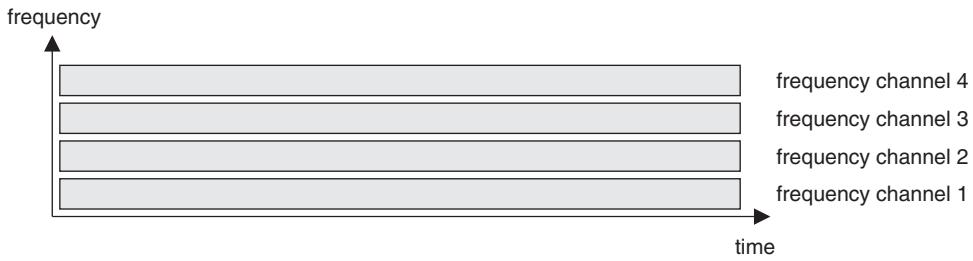
This section provides a technical summary of current mobile telecommunication systems, to the extent that this is necessary for understanding the use of smart cards in this area. Significantly more detailed descriptions of all of the technical aspects of currently used mobile telecommunication networks can be found in Jörg Eberspächer *et al.* [Eberspächer 00], Bernhard Walke [Walke 00], and Raymond Steele *et al.* [Steele 2001].

In this chapter, the term ‘mobile telecommunication system’ is used instead of ‘mobile telephone system’ because simple voice transmission is only one of many services provided by current systems, with the transmission of various types of data becoming more and more prominent.

### 19.3.1 Multiple access methods

The frequency bandwidth available to a mobile telecommunication system, which is also called its frequency spectrum, is typically limited to a few tens of megahertz. In order to make this limited bandwidth available quasi-concurrently to as many subscribers as possible, multiple-access methods must be used. The purpose of such methods is to allow the greatest possible number of mobile telephones within a cell to access the network with acceptable quality by suitably utilizing radio transmission techniques and information technology.

There are basically four multiple-access methods, which differ in the cost of their implementation and the efficiency of their utilization of the available bandwidth. These four methods are called frequency division multiple access (FDMA), time division multiple access (TDMA), code division multiple access (CDMA), and space division multiple access (SDMA). They are briefly described below.



**Figure 19.2** Frequency–time diagram of frequency division multiple access (FDMA). In this example, each frequency channel corresponds to one transmission channel. The regions between the frequency channels are guard bands for reducing interference between individual frequency bands

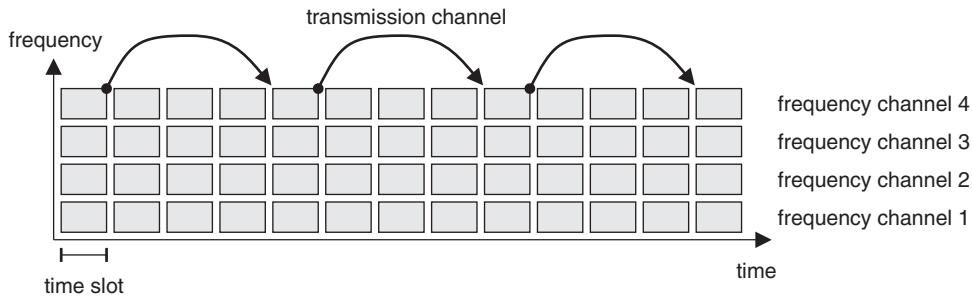
### 19.3.1.1 Frequency division multiple access (FDMA)

With frequency division multiple access, each transmitter is assigned a reserved frequency band in the total available frequency range. The transmitter can transmit continuously and exclusively within its assigned frequency band (see Figure 19.2). With FDMA, each transmitter in a cell transmits on a different frequency. This method is commonly used with conventional radio equipment, which uses a single shared channel for communication (half-duplex link). With a full-duplex link as is commonly used in telephone systems (simultaneous uplink to the base station and downlink to the mobile station), two frequency channels are required for each connection. Due to its limited technical complexity, FDMA is relatively well suited to mobile telecommunication with analog data transmission.

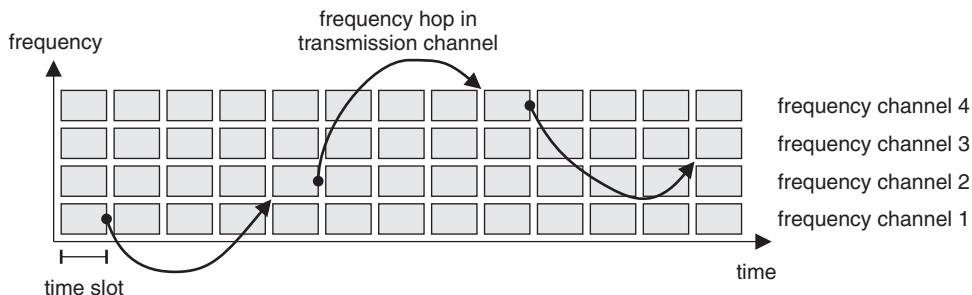
The German C-Netz was an example of a system using frequency division multiple access for the air interface between fixed and mobile telephones. In this system, separate 4.44-MHz frequency bands were reserved for uplink and downlink, with each band divided into 222 frequency channels with a width of 20 kHz each.

### 19.3.1.2 Time division multiple access (TDMA)

With time division multiple access, data is transmitted quasi-concurrently from several transmitters to a single receiver on a single frequency. Each transmitter is assigned a particular time slot that is reserved for its exclusive use, but it cannot transmit continuously (see Figure 19.3). In the GSM system, for example, the time slot allocated to a signal burst is 577 µs (15/26 ms), of which 546 µs is occupied by the signal burst transmitted in this interval. The difference between these two values (31 µs) serves as a guard time to accommodate small timing variations. Maintaining the necessary exact timing of the time slots requires very precise and technically complex synchronization between the transmitter and the receiver. Furthermore, the signal propagation time between the transmitter and the receiver must be taken into account when time division multiple access is used. For example, the signal propagation time difference between a mobile telephone close to the base station and a mobile telephone 30 km from the base station is approximately 100 µs. In practice, the propagation time differences must be offset by early transmission so that the signals transmitted by the mobile telephones always arrive at the base station exactly within the time slots reserved for them.



**Figure 19.3** Frequency–time diagram of time division multiple access (TDMA). Here each frequency channel has four transmission channels. Each grey rectangle represents one signal burst. The regions between the frequency channels and time slots are guard bands and guard times for reducing interference between the individual signal bursts

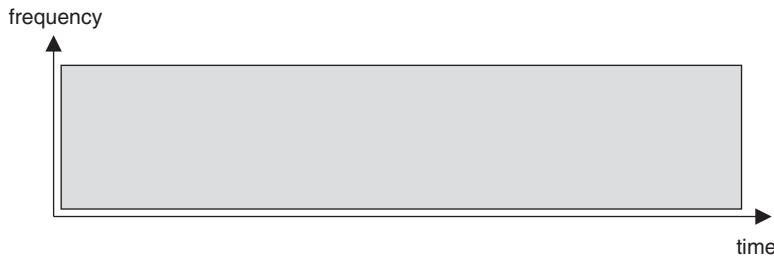


**Figure 19.4** Frequency–time diagram of time division multiple access (TDMA) with frequency hopping. Here each frequency channel has four transmission channels. Each grey rectangle represents one signal burst. The regions between the frequency channels and time slots are guard bands and guard times for reducing interference between the individual signal bursts

This requirement for offset transmission timing in order to compensate for propagation time differences is what determines the maximum diameter of a cell in the GSM system. The maximum available interval for equalizing the propagation times between the base station and the mobile telephones is  $116.3 \mu\text{s}$ , which is the maximum time that a transmission can be sent early and still arrive at the receiver within the specified time slot. This yields a maximum cell radius in GSM systems of approximately 70 km. Early transmission is also called ‘timing advance’.

In order to reduce the effects of frequency-selective interference, time division multiple access can be combined with frequency hopping, in which both the transmitter and the receiver change to a different frequency channel after each time slot in a predefined sequence (see Figure 19.4). As a result, there is a high probability that interference in particular frequency ranges will affect only isolated signal bursts. In many cases, the interference effects can be counteracted by using error-correcting transmission codes.

A combination of time division multiple access and frequency division multiple access is used for the air interface between base stations and mobile telephones in the GSM system. Here the available frequency band of 25 MHz is divided into 124 channels, each with a width of 200 kHz. Each of these frequency channels is divided into eight voice channels. This allows



**Figure 19.5** Frequency–time diagram of code division multiple access (CDMA). Within the grey rectangle representing the available frequency spectrum, several transmitters concurrently send transmitter-specific spread-spectrum signals, which the receiver can convert back into the original signals

up to eight mobile telephones to transmit at the same time on a single frequency channel, with each mobile telephone having access to the frequency channel for an interval of 0.577 ms every 4.615 ms.

#### 19.3.1.3 Code division multiple access (CDMA)

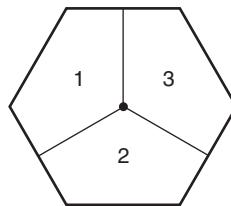
Code division multiple access is a multiple-access method in which multiple transmitters concurrently send data to a receiver while utilizing the entire available frequency spectrum (see Figure 19.5). Code division multiple access is based on spread-spectrum technology, in which a transmitter-specific mapping scheme is used to expand the original narrow-band signal into a wideband signal that is then transmitted as a wideband radio signal. This wideband signal is captured by the receiver, where it can be transformed back into the original narrow-band signal by employing the known mapping scheme used by the transmitter. A variant known as wideband code division multiple access (WCDMA) uses separate frequency bands for uplink and downlink, for which reason it is often called frequency division/code division multiple access (FD/CDMA). Another variant, time division/code division multiple access (TD/CDMA), uses different time slots to obtain the necessary separation between the uplink and the downlink paths.

Code division multiple access has the advantage of being very insensitive to frequency-selective interference. It also provides weak protection against unauthorized interception if the attacker does not know the mapping scheme used by the transmitter.

CDMA is used in the UMTS system in the WCDMA variant, with uplink and downlink channel bandwidths of 5 MHz.

#### 19.3.1.4 Space division multiple access (SDMA)

Space division multiple access is a multiple access method for concurrent data transmission from multiple transmitters to receivers using a single frequency. For this purpose, each transmitter uses a directional (adaptive) antenna aimed at the receiver concerned (see Figure 19.6). The technical complexity of this method is relatively high, so it is presently used only to a limited extent with mobile telecommunication base stations. The directional antennas are usually antenna arrays with electronic beam-steering capability, which makes it unnecessary to



**Figure 19.6** Schematic representation of space division multiple access (SDMA) with a single radio cell. The base station in the middle of the cell has three antenna arrays arranged at  $120^\circ$  around a circle, each covering one-third of the overall cell area with a certain amount of overlap

physically aim the antenna at the receiver. Space division multiple access can basically be combined with other multiple access methods, but it is rarely used in the mobile telecommunication sector at present due to its poor cost/benefit ratio.

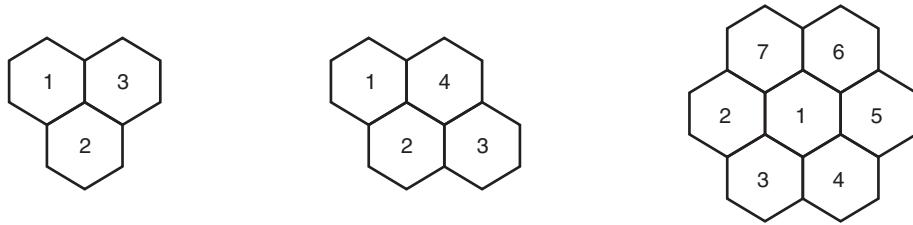
### 19.3.2 Cellular technology

The frequency band available to a mobile telecommunication system must serve a very large number of mobile telephones. If a system operator wants to provide service to a relatively large area with a large number of potential subscribers, it is not enough to simply use one or even several multiple-access methods.

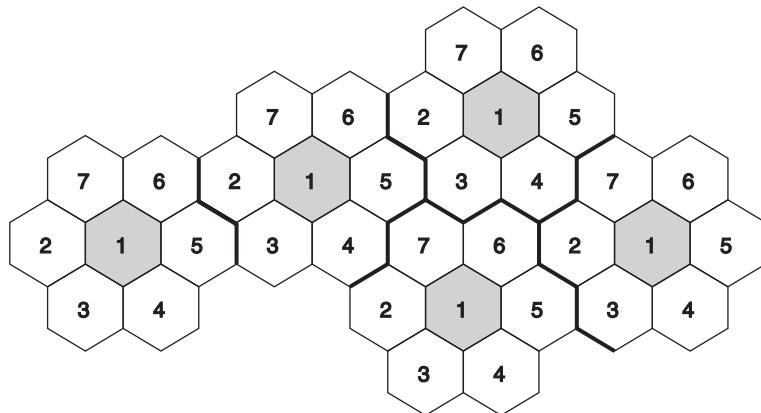
If we take the GSM system as an example, this can be quickly illustrated by a few key figures. Given the bandwidth available to the GSM system and the multiple-access method used, we can calculate that in theory a maximum of approximately 1000 subscribers can conduct telephone conversations at the same time (128 channels with 8 time slots each yields 992 voice channels). In practice, the number is significantly lower because a few channels must be used for other purposes, such as signaling.

To enable several million subscribers to talk at the same time, the entire mobile telecommunication system of a network operator is divided into cells, and a certain number of frequency channels are allocated to each cell. This approach, which was developed around 1970, is called cellular technology. Adjacent cells are allocated different frequency channels from the available frequency spectrum in order to avoid the interference that would otherwise occur if the same frequency channels were used at the same time in neighboring cells. These frequency channels can be used in other cells separated by one or more adjacent cells without causing interference problems. This frequency reuse is one of the basic foundations of cellular technology. A beneficial side effect of cellular technology is that the transmit power required from the mobile telephone depends on the cell size, so using small cells reduces power consumption and the level of ‘electrosmog’ generated by mobile telephones.

A group of cells with different frequency allocations is called a cluster (see Figure 19.7). Clusters of 3, 4, 7, 12, and 21 cells are customary. As illustrated in Figure 19.8, the network plan blankets the entire coverage region of the network with clusters. In this plan, individual cells are represented in simplified form as hexagons instead of circles, so the network plan resembles a honeycomb. Actual network plans are usually composed of cells with different sizes and have an irregular structure, as illustrated in Figure 19.9.



**Figure 19.7** Graphic representation of typical clusters with 3, 4, and 7 cells. Each of the sequentially numbered cells is allocated a certain number of frequency channels that are different from the channels allocated to the adjacent cells



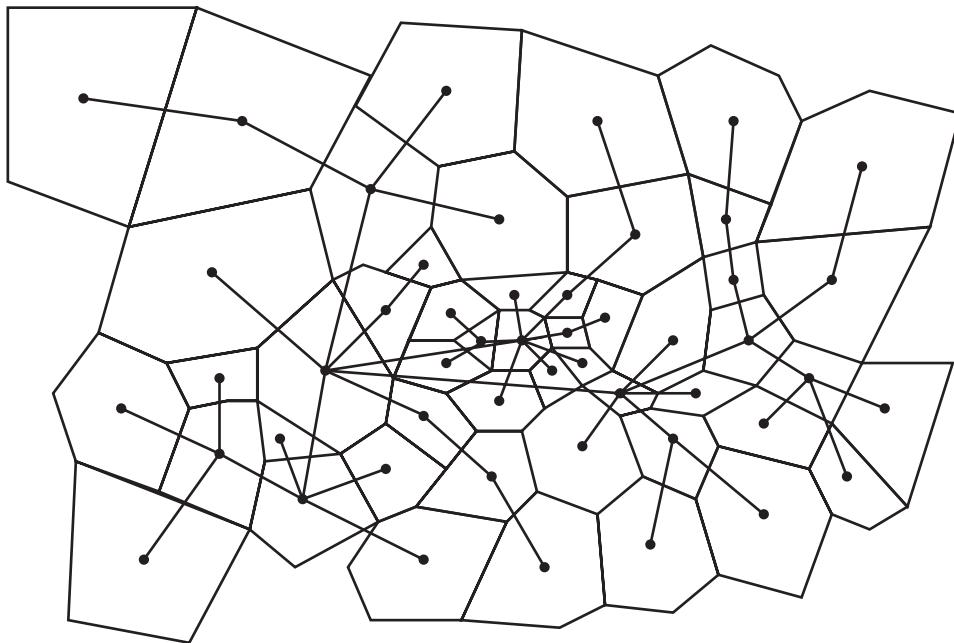
**Figure 19.8** Graphic representation of full radio coverage of a region with cells using clusters of seven cells. It can be clearly seen that any two cells with the same frequency channels are separated by at least two other cells with different frequency channels

### 19.3.3 Cell types

A decisive factor in obtaining an economical network structure with optimal network coverage is the types of cells that are used. The following figures for cell sizes are rough approximations that are primarily intended to illustrate the basic uses of the various cell types.

Large cells with a typical diameter of 10 to 40 km are used for large-scale coverage of regions with low network loading and terrain with low signal screening. They are typically used in rural areas with low population densities and flat terrain. The next smaller cell size is a macrocell, which typically has a diameter of 1 to 10 km. Macrocells are commonly used in the centers of relatively small settlements and in many suburbs of agglomeration regions.

Microcells with a typical diameter of 0.1 to 1 km are often used at traffic intersections and in inner-city areas with high call volumes. The smallest type of cell is called a picocell and has a diameter of 50 to 100 m. Picocells are often used inside buildings, such as in offices and classrooms.



**Figure 19.9** Example section of a cell plan for an actual network. Different cell sizes are used to ensure that acceptable network coverage is achieved despite differences in network loading (which usually depends on subscriber density). In this example, a traffic intersection with high mobile telephone density is located in the middle of the cell plan

An umbrella cell is a special type of cell that contains one or more smaller cells. Umbrella cells service areas that cannot be covered by the smaller cell or cells it contains. This special type of cell can be used when an area with a low average network load has a highly localized region with high call volume. Naturally, the umbrella cell and the smaller cells it contains must use different frequency channels.

An example application for an umbrella cell is an alpine region with a single ski hut and snack bar in the middle of a large ski area. In this situation, a picocell would be provided to service the ski hut, while the rest of the surrounding area would be serviced by an umbrella cell. In some ski areas, it would also be a good idea to install at least one picocell at each of the lower lift stations so skiers can while away the time spent waiting for the lift in (telephone) conversation.

Another special type of cell is the selective cell, which covers only part of a circle. This type of cell is typically installed facing an underpass, and thanks to its highly directional cell pattern, which resembles the beam of a spotlight, it can provide very good mobile telecommunication coverage inside the underpass. Selective cells are also a proven means to provide coverage inside tunnels.

In a country the size of Germany (all sixteen federal states) with an area of approximately  $360\,000\text{ km}^2$ , around 3000 base stations are needed to provide full coverage for approximately 5 million subscribers. With 20 million subscribers, the number of base stations rises to around 12 000 for practically full network coverage. A network of this size requires approximately 60 mobile switching centers (MSCs). In mid-2001, the four German GSM networks had a combined total of approximately 52 000 base stations with 164 000 cells.

### 19.3.4 Bearer services

The task of bearer services in a mobile telecommunication system is to transport voice and data between the terminal and the background system. The acoustic transmission of voice signals at customary telephone quality requires a minimum frequency bandwidth of 3100 kHz, which is designated ‘telephone bandwidth’. This bandwidth results from the fact that human speech primarily uses the frequency range between 300 and 3400 Hz. In the GSM system, this continuous signal is converted into a compressed, discontinuous data stream, and in full-rate mode this data stream requires a data transmission rate of 13 kbit/s. This transmission rate is sufficient for a half-duplex telephone connection with standard telephone quality, which means an uplink or a downlink. If both parties want to be able to speak at the same time as usual, a full-duplex link is required, as is commonly used in telephony. This yields a net transmission rate of 26 kbit/s for the two directions. In practice, the required transmission rate is considerably larger due to the need for control data and elaborate error correction codes. In the GSM system, the required gross transmission rate for uplink and downlink is approximately 34 kbit/s.

Ideally, bearer services for data transmission enable the transparent exchange of data between the two communicating parties, which means that these parties do not need to be concerned with the communication protocols used by the intermediate parties. The bearer services used in the GSM system are SMS, USSD, CSD, HSCSD, and GPRS. The availability of these services depends on the technical configuration of the actual terminal and the GSM system that is used.

## 19.4 THE GSM SYSTEM

The smart cards used in GSM mobile telephones, which are called subscriber identity modules (SIMs), set the pace for smart cards with regard to functionality and memory size. The high rate of evolution in the entire telecommunication sector is a decisive factor in smart card technology. The pioneering role of SIMs in terms of technology and standardization, relative to all other current smart card applications, provides suitable grounds for discussing this topic here in considerable detail.

The GSM system, which was commercially inaugurated in 1992, developed into an international standard for mobile telecommunication systems within only a few years. This includes voice communication as well as data communication. At the start of 2008, there were a total of 500 mobile telecommunication networks based on the GSM standard with more than 3 billion subscribers. More than 20 billion text messages are sent every month.<sup>6</sup>

Specification of the GSM system began in 1982 under the auspices of the Conférence Européenne des Postes et Télécommunications (CEPT). The objective was to generate a specification for a trans-national, interoperable mobile telephone network. In the course of time, these efforts led to the conclusion that a specification could be generated for a trans-national, interoperable and ISDN-compatible digital cellular mobile telecommunication system operating in the 900-MHz band. The Groupe Spécial Mobile was founded for this purpose, which gave rise to the original abbreviation ‘GSM’. In 1986, the GSM Permanent Nucleus with headquarters in Paris was established to coordinate the generation of the specification. It was

<sup>6</sup> A good overview of current statistical figures and network operators can be found at GSM World [GSM]

later responsible for specifying a wide variety of tests for system components. From a technical perspective, it is interesting to note that some of the technologies chosen for GSM at that time were entirely new and untested in practice. For instance, the air interface with a combination of time division multiple access and frequency division multiple access together with digital data transmission was virgin territory for large-scale mobile telecommunication applications. These decisions led to many technical problems, especially in the system development stage, but from the present perspective they can be regarded as a fortunate choice because GSM proved to be an innovative system that was not burdened with the technological ballast of the early days of mobile telecommunication.

The common contractual basis for operators of GSM networks is the Memorandum of Understanding (MoU), which was first signed by fifteen European network operators in 1987. The GSM Association is an international body for the coordination of mobile telecommunication systems, with offices in Dublin and London. It was founded in Copenhagen in 1987, and it is responsible for the development and use of the GSM standards. The GSM Association represents more than 500 network operators, manufacturers, and suppliers in the GSM industry. In 1989, the specifications developed by the various working groups under the leadership of the GSM Permanent Nucleus were transferred to the newly founded European Telecommunication Standards Institute (ETSI), where they have been maintained and extended since then. In 1990, all of the GSM Phase 1 specifications were complete in an acceptable form and were frozen.

In 1998, the Subscriber Identity Module Expert Group (SIMEG) began work on a specification for the GSM smart card, which is called the subscriber identity module (SIM). This group was composed of representatives of card producers, mobile telephone manufacturers, and network operators. Working under the auspices of the ETSI, the SIMEG generated the specification for the interface between the smart card and the mobile telephone. This specification now bears the designation TS 51.011. In 1994, the SIMEG was transformed into the newly founded Special Mobile Group 9 (SMG9) with the same tasks and scope of authority, and it was given the mandate of maintaining and refining all of the SIM specifications until 2000. The SMG9 was dissolved in 2000, and its tasks were divided between two newly founded expert groups. The ETSI Project Smart Card Platform (EP SCP) expert group handles all generic topics regarding smart cards for telecommunication, while the ETSI SCP Expert Group is responsible for the application-specific interface between the mobile telephone and the SIM or USIM.<sup>7</sup>

The first operating GSM network was demonstrated at the ITU Telecommunication Fair in Geneva in 1991. During the fair, approximately 11 000 calls were routed without any major problems. In 1992, the first GSM systems were put into service in several European countries (Denmark, Finland, France, Germany, Italy, Portugal, and Sweden). At that time, there were approximately 250 000 subscribers. Also in that year, the first roaming agreement between two network operators was signed and the first non-European network operator signed the MoU, which meant that the operator officially decided to use the GSM system. Only one year later, at the end of 1993, the millionth subscriber was registered. In that year, the first GSM-1800 network began operation in Great Britain and the first roaming agreement was concluded. The first GSM-1900 network began operation in the USA in 1995, and the number of registered subscribers reached the 100 million mark at the end of July 1998. This rose to 1 billion in

<sup>7</sup> An article by Klauss Vedder, ‘The Subscriber Identity Module: Past – Present – Future’ in [Hillebrand 2002] provides a comprehensive overview of the interesting history of the expert groups for standardization of the SIM, USIM, and UICC

the first quarter of 2004. At the end of 2008, there were 2.5 billion subscribers worldwide, and this is expected to increase to 4 billion in the first quarter of 2010. By the end of 2007, 10 billion SIMs had been issued, which gives an impressive picture of the size of this smart card application.

The GSM specifications were extended in 1991 and 1992. They now cover the 1800-MHz frequency band (1710–1785 MHz uplink, 1805–1880 MHz downlink; wavelength approximately 16.6 cm) with GSM 1800 (previously called Digital Cellular System or DCS) and the 1900-MHz band (1850–1910 MHz uplink, 1930–1990 MHz downlink; wavelength approximately 15.8 cm) with GSM 1900 (previously called Personal Communication System or PCS). Since then, GSM in the original 900-MHz frequency band (880–915 MHz uplink, 925–960 MHz downlink; wavelength approximately 33.3 cm) is called GSM 900. Due to the higher frequency and lower transmit power, the maximum diameter of a cell in the higher-frequency systems is only 20 km. Consequently, they are primarily used in regions with high subscriber density, and less often in regions with low subscriber density. The main differences between GSM in the original 900-MHz frequency band and GSM in the 1800 or 1900-MHz frequency band are found in the transmitter and receiver components on each side of the air interface.

The designated successor to GSM is UMTS,<sup>8</sup> whose basic architecture is based on GSM. It thus does not represent a fundamentally new mobile telecommunication technology, as did GSM when it was originally developed.

### 19.4.1 Specifications

A large number of coordinated and mutually dependent specifications are necessary to fully describe the GSM system in technical terms. In total, there are approximately 130 individual specifications with a total scope of more than 6000 pages. The most important standards are listed in Table 19.1 with their current numbers (TS) and their former numbers (GSM).

Particularly in connection with the GSM system, the terms specification and standard are often used interchangeably. In the case of GSM, both terms are justified. The specification documents formally have the status of standards because they are published by the ETSI standardization organization. At the same time, their technical descriptions are so strict that all implementations based on them are mutually compatible, which is a typical characteristic of a specification. We generally use the less ambiguous term ‘specification’ in the following descriptions.

The course of development of the GSM system is characterized by a series of evolutionary phases. The basic services: voice transmission, call forwarding, roaming and text messaging (SMS) were implemented in Phase 1, which began in 1992. Additional services, including conference calls, call handover, call number negotiation and GSM in the 1800-MHz frequency band, were added in Phase 2, which began in 1996. This was followed by Phase 2+, in which these services were augmented by (among other things) the SIM Application Toolkit functions, high-speed circuit switched data (HSCSD), and the General Packet Radio System (GPRS).

As usual with specifications, the GSM specifications introduce their own technical vocabulary. It is precisely defined in technical terms in various lists of abbreviations and glossaries, and it is only applicable to the GSM field. Due to this technical vocabulary, it is generally

<sup>8</sup> See also Section 19.5, ‘The UMTS System’, on page 848

**Table 19.1** The most important standards for the SIM and SIM-related services. A more general listing of all ETSI standards in the SIM environment is provided in the directory of standards in the appendix. All ETSI standards can be obtained free of charge from the ETSI web server [ETSI]

Number	Former number	Title
TS 42.009	GSM 02.09	Security Aspects
TS 42.017	GSM 02.17	Subscriber Identity Module (SIM); Functional characteristics
TS 42.019	GSM 02.19	Subscriber Identity Module Application Programming Interface (SIM API); Stage 1
TS 42.048	GSM 02.48	Security mechanisms for the SIM Application Toolkit; Stage 1
TS 43.019	GSM 03.19	Subscriber Identity Module Application Programming Interface (SIM API) for Java Card; Stage 2
TS 43.048	GSM 03.48	Security Mechanisms for SIM Toolkit Application; Stage 2
TS 51.011	GSM 11.11	Specification of the Subscriber Identity Module – Mobile Equipment (SIM–ME) interface
TS 51.013	GSM 11.13	Test specification for Subscriber Identity Module (SIM) Application Programming Interface (API) for Java Card
TS 51.014	GSM 11.14	Specification of the SIM Application Toolkit for the Subscriber Identity Module – Mobile Equipment (SIM–ME) interface
TS 51.017	GSM 11.17	Subscriber Identity Module (SIM) test specification

relatively difficult for newcomers to become familiar with GSM, since constant reference to the explanations of the abbreviations used in the text is necessary when studying the specifications.

The specification forming the basis for the GSM security module in the terminal is designated TS 42.017 (SIM Functional Characteristics) and contains a relatively abstract description of the functional requirements on the SIM. The most important card-specific document in the GSM system, TS 51.011 ('Specification of the Subscriber Identity Module – Mobile Equipment (SIM – ME) interface'), is based on the SIM specification. In more than 170 pages, TS 51.011 precisely and unambiguously specifies the interface to the SIM. This is a pure interface specification with no information regarding the actual implementation.

In addition to these specifications, which primarily describe the basic functionality of the SIM, there is TS 51.014 (Specification of the SIM Application Toolkit for the Subscriber Identity Module – Mobile Equipment (SIM – ME) Interface), which describes a platform for secure value-added services in the SIM. This is called the SIM Application Toolkit (SAT). This specification was published in 1996, and it primarily offers network operators the opportunity to load their own telephone control applications into the smart card. TS 51.014 specifies in detail how functions such as driving the display, polling the keypad, sending text messages (SMS), and other functions related to corresponding value-added applications must be implemented in the SIM.

The TS 02.48 requirements specification (Specification of security mechanisms for the SIM application toolkit, stage 1) and its companion specification TS 03.48 (Specification of security mechanisms for the SIM application toolkit, stage 2) introduce two essential security mechanisms for the SIM. The first topic they address is specifying security mechanisms for end-to-end communication between the background system and the SIM with protection against interception and manipulation. In practice, these mechanisms are primarily used for secure data transmission over the air interface (OTA). The second topic addressed by TS 3.048 is a basic mechanism for remote file management (RFM) and remote applet management ((RAM)).

This description is essentially bearer-independent, although it is described in TS 43.048 using SMS transmission as an example.

Smart cards with Java established a position in the telecommunication sector very quickly, and the consequences of this were reflected relatively early in the GSM specifications. The basis for all smart card operating systems that support executable program code is provided by the TS 42.019 specification. It contains a list of all basic services of a language-independent API for executable program code in the SIM. Based on this standard, TS 43.019 specifies a detailed implementation of a Java Card API for SIMs based on the Java Card 2.1 specification. This standard is the key document for using Java Card in GSM systems. It is supplemented by TS 51.013, which specifies the test environment, test applications, test procedures, test coverage, and individual test scenarios. All of the described tests address the information technology aspects of Java Card SIMs for GSM.

The GSM specifications related to the SIM are not being developed any further because the functionality of the SIM is fully adequate for the current needs of GSM systems. The only changes that are still routinely made to the relevant specifications consist of clarifications of passages subject to interpretation. Since 1999, the focus has been on standardizing the UICC (universal integrated circuit card) and the USIM (universal subscriber identity module) application, which is primarily being conducted under the auspices of the 3GPP.

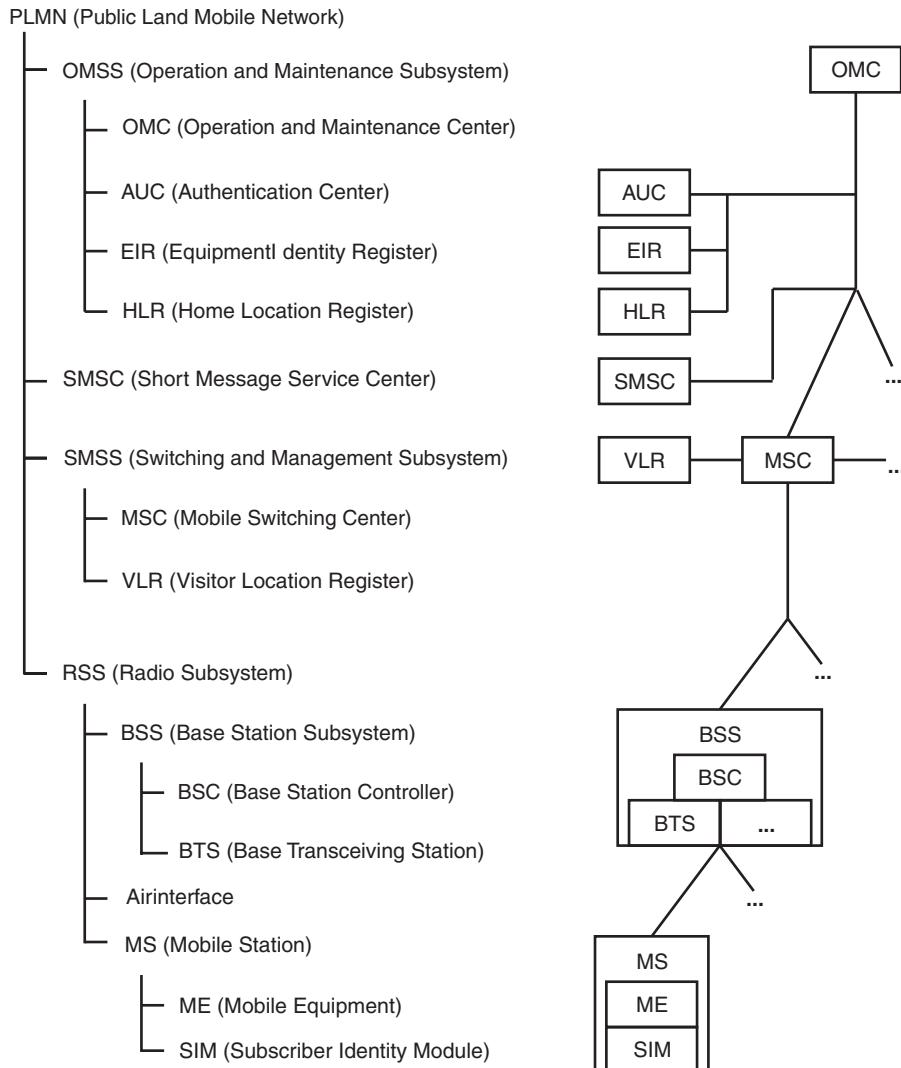
### 19.4.2 System architecture and components

Every GSM network can be divided into three general subsystems, which are described in general terms in the GSM 01.02 specification (General description of a GSM Public Land Mobile Network (PLMN)) and depicted in Figure 19.10. The three subsystems are the radio subsystem (RSS), the network and switching subsystem (NSS), and the operation subsystem (OSS).

The radio subsystem is composed of the terminal (mobile station or MS) and the base station subsystem (BSS). The mobile station consists of two physically and logically separate components, which are called the mobile equipment (ME) and the subscriber identity module (SIM). The mobile equipment is the radio and encryption component with the user interface, while the SIM is the proper designation (in GSM nomenclature) of a GSM-specific smart card. These two components together form the operational mobile telephone.

As a rule, a base station subsystem in the form of a base station is located at the center of each cell. The tasks of the base station subsystem are to establish contact with the mobile telephones over the air interface and to route traffic to the higher-level components of the network. A base station consists of one or more base transceiver stations (BSTs) and a base station controller (BSC). The base station transceiver, with its antenna and associated radio-frequency components, is the actual transmission and reception component.

A typical receiver module of a base station transceiver has eight 200-kHz channels, so in theory it can concurrently maintain eight active links to mobile stations. Only seven active links are normally used in practice, with one channel normally reserved for administrative communication. One, three or six receiver modules are usually installed in each base transceiver station. One or more base transceiver stations are in turn managed by a base station controller. A typical arrangement consists of three base station transceivers configured at 120° to each other and connected to a base station controller. If a mobile station moves from the send/receive region of one base transceiver station to the region of another base transceiver



**Figure 19.10** Basic architecture of a typical mobile telecommunication system compliant with the GSM 01.02 specification. In this example, the EIR and HLR databases are centralized. As many aspects of the PLMN configuration are left to the discretion of the network operator, distributed databases divided among several MSCs can also be used if necessary (such as due to high network loading). For ease of understanding, a link to a text messaging service center (short message service center – SMSC) is also shown here, although it is not an essential component of the GSM system. One or more radio subsystems can be combined to form a location area (LA), and one or more network and switching subsystems can be combined to form a service area (SA)

station and both base transceiver stations are assigned to the same base station controller, the base station controller can independently initiate the handover after signaling this to the responsible mobile switching center.

Data transmission over the air interface is encrypted and has a net transmission rate of 13 kbit/s in full-rate mode. It employs a lossy compression method with technically sophisticated error correction mechanisms such as frequency hopping, convolutional coding, and interleaving.

The network and switching subsystem essentially consists of the mobile switching center and the visitor location register (VLR). Each mobile switching center (MSC) manages several base station subsystems. The mobile switching center forms the link between the base station subsystems connected to it, other mobile switching centers, and the public switched telephone network. It is responsible for setting up, managing, and shutting down connections, handling call charges, and supporting value-added services such as call forwarding, call blocking, and conference calls. The visitor location register (VLR) contains information about all mobile stations currently within range of the associated mobile switching center. This information is needed for functions such as routing a call to a particular mobile telephone via the appropriate base station subsystem and radio cell. The VLR also maintains a list of mobile stations of subscribers from other networks that have logged into the network of the associated mobile switching center via roaming.

The top hierarchical level in a GSM system is the operation subsystem. It consists of the operation and maintenance center (OMC), the authentication center (AuC), the home location register (HLR), and the equipment identity register (EIR). The operation and maintenance center is responsible for regular network operation, subscriber management, and call billing. The authentication center is the security entity on the network side, which in a manner of speaking makes it the counterpart of the SIM on the mobile side. It generates and manages all keys and algorithms needed for system operation, in particular for the authentication of the mobile stations (i.e. the SIMs). Another core entity is the home location register, which holds all of the subscriber data as well as the localization data of each of the mobile stations. The equipment identification register (EIR) is the equivalent to the HLR for mobile stations instead of subscribers. It contains essential data, such as the serial numbers of all mobile units present in the network.

### 19.4.3 Important data elements

This section describes a selection of important data elements that are primarily related to the SIM and its functions. The coding of the described data elements can be found in the description of the typical files of a SIM. The primary GSM databases and data elements are described in Table 19.2.

#### 19.4.3.1 Coding of alphanumeric characters

The GSM system was originally designed for use in the central European countries, and in this system alphanumeric characters were and still are represented by an ASCII-based seven-bit coding scheme. It is defined in the TS 23.038 specification. However, the spread of GSM to other countries made it necessary to extend the character set. Consequently, the UCS-2 16-bit subset of the UCS character set is used for characters that are not present in the Western

**Table 19.2** The databases essential for the operation of a GSM system and the most important data elements in these databases

Database	Data elements
HLR (home location register)	<ul style="list-style-type: none"> <li>• Subscriber information</li> <li>• IMSI (international mobile subscriber identity)</li> <li>• MSISDN (mobile station ISDN number)</li> <li>• Service restrictions (e.g. roaming not allowed)</li> <li>• Subscribed services</li> <li>• Parameters for value-added services</li> <li>• Information about the subscriber's equipment</li> <li>• Authentication data (i.e. RAND, SRES, Kc triplet) (implementation-dependent)</li> <li>• Localization data (mobile location information)</li> <li>• MSRN (mobile station roaming number)</li> <li>• Address of current VLR (if available)</li> <li>• Address of current MSC (if available)</li> <li>• TMSI (if available)</li> </ul>
VLR (visitor location register)	<ul style="list-style-type: none"> <li>• Subscriber information</li> <li>• IMSI (international mobile subscriber identity)</li> <li>• MSISDN (mobile station ISDN number)</li> <li>• Parameters for value-added services</li> <li>• Information about the subscriber's equipment</li> <li>• Authentication data (i.e. RAND, SRES tuple) (implementation-dependent)</li> <li>• Localization data (mobile location information)</li> <li>• MSRN (mobile station roaming number)</li> <li>• LAI (location area information)</li> <li>• TMSI (if available)</li> </ul>
EIR (equipment identity register)	<ul style="list-style-type: none"> <li>• IMEI (international mobile equipment identity) of all mobile stations</li> <li>• IMEI of mobile stations to be reported (greylist)</li> <li>• IMEI of blocked mobile stations (blacklist)</li> </ul>

European character set used in GSM. This character set can be used to represent the most important characters of all living languages.<sup>9</sup>

To minimize memory space requirements, three different schemes are specified for character coding using UCS-2. The preferred scheme (Scheme 1), which however requires the most memory, is identified by the value '80' in the first byte. This is followed by the 16-bit USC-2 character code with the most significant byte first. Unused bytes are set to 'FF'.

Scheme 2 is identified by the value '81' in the first byte. The second byte contains the number of characters in the character string. The two following bytes are a 16-bit pointer to the UCS character set, which serves as an offset to a language-specific selection of characters within UCS. Bits 1–7 and bit 16 of this pointer are set to 0. If bit 8 of one of the following bytes has a value of 1, bits 1–7 of that byte must be added to the pointer value, and the resulting

<sup>9</sup> See also Section 6.2, 'Encoding Alphanumeric Data', on page 115

16-bit pointer indicates the actual character in the UCS character set. If bit 8 has a value of 0, the character in question is a member of the 7-bit character set specified in TS 23.038.

Scheme 3 is identified by an initial byte value of '82'. As with Scheme 2, the second byte contains the length of the character string, while the third and fourth bytes form a complete 16-bit pointer to the USC character table. If bit 8 of the following byte has a value of 1, the following seven bits must be added to the pointer value to uniquely determine the UCS character. If bit 8 has a value of 0, the character in question is a member of the 7-bit character set specified in TS 23.038.

#### **19.4.3.2 SIM service table (SST)**

The SST is a table of services that can be used with or enabled in addition to voice service, such as the text message service or a fixed dialing number service.

#### **19.4.3.3 Fixed dialing numbers (FDN)**

Fixed dialing numbers are a special type of dialing numbers that can be selected even when all other dialing numbers are blocked in the mobile telephone.

#### **19.4.3.4 ICC identification (ICCID)**

The ICCID is a unique smart card identification number. It is BCD-coded and ten bytes long, and it can be right-padded with 'F' as necessary.

#### **19.4.3.5 International mobile equipment identity (IMEI)**

The IMEI is the unique device number of the mobile station. It consists of 15 digits and thus usually occupies 8 bytes. The IMEI is composed of a six-digit type approval code, a two-digit manufacturer code, a six-digit serial number, and a check digit. The IMEI is stored in the mobile telephone and in the equipment identification register (EIR) in a central location.

#### **19.4.3.6 International mobile subscriber identity (IMSI)**

The IMSI is the unique subscriber identity within the GSM system. It is BCD-coded and has a length of nine bytes, which is right-padded with 'F' as necessary. It consists of the mobile country code (MCC), the mobile network code (MNC), and a serial number assigned by the network operator. The IMSI is normally never transmitted over the air interface in plaintext, in order to prevent illicit tracking of the location of a mobile station with a known IMSI. Instead of the IMSI, the TMSI is normally used together with the LAI for identification purposes.

#### **19.4.3.7 Ki (key individual) and Kc (key cipher)**

The keys Ki and Kc are secret keys for symmetric cryptographic algorithms. Ki is the card-specific key for the cryptographic computation of the authenticity of the SIM, while Kc is

used for encrypting data transmitted between the mobile station and the base station over the air interface.

#### **19.4.3.8 Short message service (SMS)**

The short message service allows text messages with a maximum length of 160 alphanumeric characters to be transmitted between the network and the mobile station over the signaling channel. The SMS service is used not only for conveying text messages for subscribers, but also as a bearer service for transmitting data to the mobile telephone or the SIM, for instance for the WAP and OTA services.

#### **19.4.3.9 Abbreviated dialing numbers (ADN)**

Abbreviated dialing numbers are dialing numbers stored in the mobile telephone or the SIM along with supplementary information, which can be easily and quickly selected using a menu or special buttons.

#### **19.4.3.10 Location area information (LAI)**

The LAI is the unique location information of the mobile station. It is used in combination with the TMSI to generate a unique subscriber identity. The LAI consists of a three-digit country code (CC), a two-digit mobile network code (MNC), and a location area code (LAC), which has a maximum length of five digits.

#### **19.4.3.11 Mobile station ISDN number (MSISDN)**

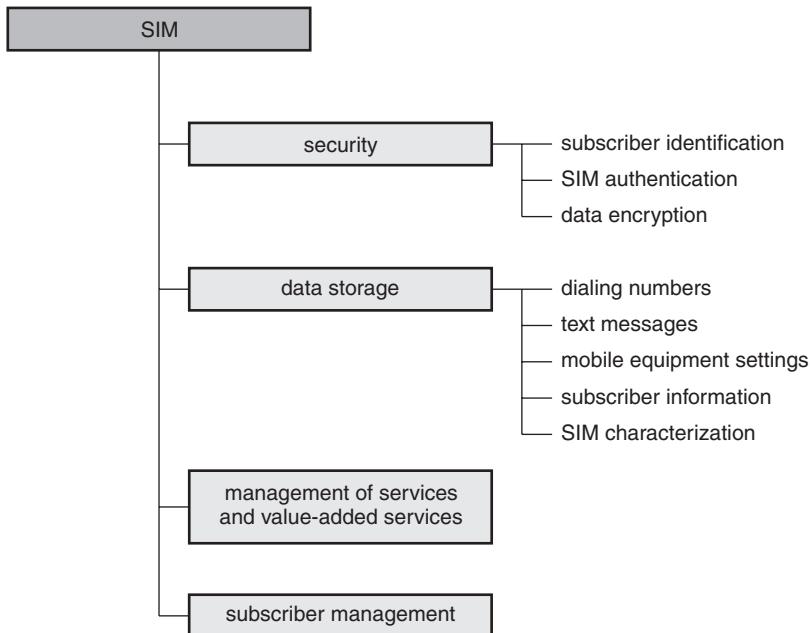
THE MSISDN is the dialing number of the mobile station. It is independent of the subscriber identity (IMSI).

#### **19.4.3.12 Temporary mobile subscriber identity (TMSI)**

The TMSI is a temporally and geographically constrained subscriber identity with a length of four bytes. It is used to protect the true subscriber identity. The TMSI is only unique in combination with the location area information (LAI). The TMSI is assigned by the VLR, where it is also stored.

### **19.4.4 The subscriber identity module (SIM)**

The SIM is a mandatory security module located in the mobile station of a GSM mobile telecommunication system as an exchangeable component. It is defined as follows in the EN 300 920 specification: ‘The SIM is an entity that contains the identity of the subscriber. The primary function of the SIM is to secure the authenticity of the mobile station with respect to the network,’ as shown in Figure 19.11.



**Figure 19.11** The basic functions of the SIM in the GSM system

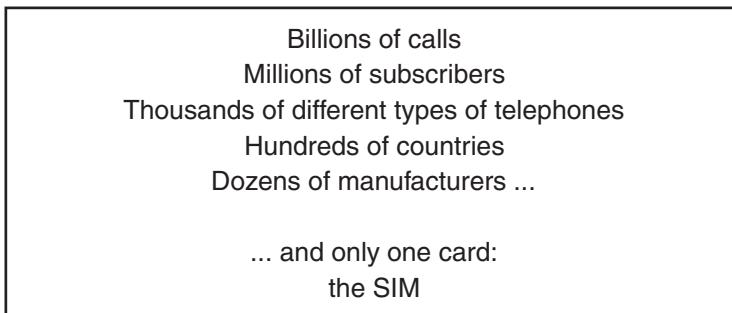
In addition to its primary functions as the bearer of the identity of the subscriber, which is realized using a PIN, and authentication of the mobile station with respect to the network, the SIM performs several other functions. It allows program execution to be protected against manipulation, and it enables the storage of data such as dialing numbers, text messages, and personal configuration settings for the mobile telephone. In addition, it is the bearer of secure value-added services related to mobile telecommunication.

SIMs are used in various form factors in GSM systems, although the ID-1 format is no longer used. The ID-1 format was originally based on the idea of a company telephone or family telephone with a separate card for each user. With modern mobile telephones, the SIM is only rarely exchanged, for which reason the ID-000 (plug-in SIM) format is used instead. SIMs in mini-UICC format are also used in some mobile telephones, although only rarely up to now.

Communication between the mobile equipment and the SIM uses the T = 0 protocol with the standard parameters specified in ISO/IEC 7816-3. The data transmission convention can be freely selected by the card via the ATR. PPS can be used, and this is often done in practice to increase the data transmission rate. The divisor value (clock rate conversion factor) typically used in many mobile telephones is 64, which yields a data transmission rate of 78 kbit/s with a 5-MHz clock rate. In isolated cases, values as low as 8 ( $\approx 156$  kbit/s with a 5-MHz clock rate) are used.

For historical reasons, the T = 0 communication command GET RESPONSE is incompatible with ISO/IEC in two regards.<sup>10</sup> If the first difference is that if data can be fetched

<sup>10</sup> Strictly speaking, in this case the ISO/IEC 7816-3 standard is effectively incompatible with TS 51.011 because the latter was chronologically first. However, an ISO/IEC standard has a higher rank than a TS specification, so formally the TS specification is incompatible



**Figure 19.12** The slogan of the SIM standardization group SMG9 on its tenth anniversary in 1998

from the terminal using GET RESPONSE, the SIM indicates this by putting '9F' in the SW1 byte, rather than '61' as specified in ISO/IEC 7816-3. The other special feature of GET RESPONSE is that according to TS 51.011 the data provided by the SIM can be fetched one byte at a time using GET RESPONSE, and the SIM maintains a pointer for this purpose in the transmit buffer. This is not possible according to ISO/IEC 7816-3, which specifies that the terminal can only use GET RESPONSE to request data to be fetched starting with the first byte or as an entire block. However, these two incompatibilities do not lead to any problems in practice.

On the occasion of the tenth anniversary of the SIM standards in 1998, the SMG9 published the slogan shown in Figure 19.12. It clearly indicates the significance and size already achieved by GSM system that time, as well as the extent of the pride in one of the essential components of the system: the subscriber identity module.

The SIM specifications form the basis for many other specifications for smart cards used in the mobile telecommunication sector. The most important of these specifications are briefly described below.

As part the development of the European (DECT) standards by ETSI (DECT stands for 'digital enhanced cordless telecommunication' and refers to cordless telephones using cellular technology and operating in the 1.9-GHz band), the first version of the specification for the DECT authentication module (DAM) was published in 1992. This specification was frozen in 1995 under ETSI number ETS 300 331. Unfortunately, the DAM was never implemented as an actual product, since it was specified as optional and fell victim to the cost reduction efforts of all manufacturers of cordless telephones.

The TETRA digital terrestrial trunked radio system [TETRA] also has provision for an optional smart card called the TETRA-SIM, whose specifications are based on the SIM for GSM mobile telephones. The EN 300 812 specification for the TETRA-SIM also allows it to be implemented as an application in the UICC if necessary. As the TETRA-SIM is optional, it can also take the form of a software implementation in the mobile station.

Another type of smart card whose specification is based on the SIM is the R-UIM (removable user identity module) in the 3G mobile telecommunication systems defined by the Third Generation Partnership Project 2 (3GPP2), such as the CDMA 2000 system. The R-UIM is specified as an optional component of the mobile station in such systems, and its functionality is similar to that of the SIM. It is specified in the TIA/EIA/IS-820 and TIA/EIA/IS-839

standards. A significant difference between the R-UIM and the SIM is that the former includes the CAVE (cellular authentication, voice privacy and encryption) cryptographic algorithm, which as its name suggests can be used in the R-UIM for a wide variety of cryptographically secured functions. A UIM Application Toolkit (UATK) based on the SIM Application Toolkit is also specified for the R-UIM.

In the satellite-based Inmarsat mobile telephone system [Inmarsat], which has been in operation since the early 1980s, modified GSM cards are also used as the basis for determining the subscriber identity. Another extension of the SIM, which features a few additional files and a special cryptographic algorithm, is the smart card for the international Iridium mobile telecommunication system [Iridium]. In its ultimate form, this system is intended to consist of 66 satellites orbiting at a height of 780 km that are functionally equivalent to GSM base stations. The air interface between the mobile stations and the satellites operates at a frequency of 1616 MHz.

#### 19.4.4.1 SIM commands

The TS 51.011 specification defines 22 operational commands for the SIM, which are identified by a class byte value of ‘A0’.<sup>11</sup> The commands can be classified into commands related to security, commands for file operations, and commands for the SIM Application Toolkit. They are summarized in Table 19.3 on the facing page.<sup>12</sup>

There is a special feature with regard to entering the four-digit PIN code, which is called the ‘cardholder verification’ (CHV) in GSM. User PIN queries can be disabled with the DISABLE CHV command together with the right PIN, thus making it unnecessary to enter the PIN code before logging in to a mobile telecommunication system. The disadvantage of this, which is that lost cards can be used illicitly for telephoning until they have been blocked by the network operator, falls under the responsibility of the user. The ENABLE CHV command can be used as necessary to again enable PIN queries.

A SIM normally has two CHVs. The idea behind this is to differentiate between the card user and the cardholder, in order to allow different sets of functions to be defined or allow only the cardholder to use certain functions. This can be briefly illustrated using the EF<sub>FDN</sub> file holding the fixed dialing numbers as an example. We assume that the card user only knows CHV 1, which is sufficient for dialing the numbers stored in EF<sub>FDN</sub>. The cardholder also knows CHV 2, which, due to the access condition specified for the UPDATE RECORD command (prior correct verification of CHV 2), enables the cardholder to change the entries in EF<sub>FDN</sub>. This feature can for example be used to restrict the numbers that children can dial with the mobile telephone to the numbers stored in EF<sub>FDN</sub>, since they only need to know CHV 1 in order to place calls to these numbers. Their parents, who also know CHV 2, can also edit the allowed numbers.

For compatibility reasons, all SIMs still support the SLEEP command, although it has been obsolete for many years. Its original function, which was to save power in terminals, has now been taken over by the hardware of the smart card microcontroller or the operating system.

The STATUS command is used for two purposes. The first is requesting information about the currently selected file, while the second is verifying that a SIM is present. The mobile

<sup>11</sup> See also Section 8.3.1, ‘Command APDU structure’, on page 221 for a description of the command structure

<sup>12</sup> See also Chapter 11, ‘Smart Card Commands’, on page 353 for descriptions of typical smart card commands

**Table 19.3** The commands specified for the SIM in TS 51.011

Command	Brief description
<i>Security commands</i>	
CHANGE CHV	Change the PIN
DISABLE CHV	Disable PIN queries
ENABLE CHV	Enable PIN queries
RUN GSM ALGORITHM	Execute the GSM-specific cryptographic algorithm
UNBLOCK CHV	Reset the PIN retry counter from its terminal count
VERIFY CHV	Verify the PIN
<i>File operation commands</i>	
INCREASE	Increment a counter in a file
INVALIDATE	Reversibly block a file
READ BINARY	Read from a file with a transparent structure
READ RECORD	Read from a file with a record-oriented structure
REHABILITATE	Unblock a file
SEEK	Search for a text string in a file with a record-oriented structure
SELECT	Select a file
STATUS	Read various data from the currently selected file
UPDATE BINARY	Write to a file with a transparent structure.
UPDATE RECORD	Write to a file with a record-oriented structure
<i>SIM Application Toolkit commands</i>	
ENVELOPE	Supply data to a SIM value-added service in the context of the SIM Application Toolkit
FETCH	Fetch a SIM Application Toolkit command from the SIM in the mobile equipment
TERMINAL PROFILE	List all the SIM Application Toolkit functions of the mobile equipment
TERMINAL RESPONSE	Supply the response of the mobile equipment to a previous SIM Application Toolkit command of the SIM
<i>Miscellaneous commands</i>	
GET RESPONSE	Command specific to T = 0 for requesting data from the smart card
SLEEP	Obsolete command for placing the smart card in a low-power state

equipment periodically sends a STATUS command to the SIM at intervals of approximately 30 s to confirm that the SIM is present. If the SIM does not send a response to the STATUS command within 5 s, the SIM is deactivated and the call is terminated. In addition, there is often some form of mechanical contact present to detect or prevent exchanging the SIM while the mobile telephone is in use.

The relevant GSM specifications do not specify any administrative commands for file management. Originally, such commands were not necessary, since for a long time smart card operating systems did not allow file creation or deletion because there was not enough memory space available. This situation has fundamentally changed, with the result that these file management functions, which in principle are very important, are now available. If sufficient free memory space for files is available, they can be used to download file-based applications into SIMs at any desired time, including by means of remote file management (RFM).

**Table 19.4** Smart card commands specified in TS 102 222 for managing applications in telecommunication smart cards

Command	Brief description
ACTIVATE FILE	Unblock a file
CREATE FILE	Create a new file
DEACTIVATE FILE	Reversibly block a file
DELETE FILE	Delete a file
TERMINATE CARD USAGE	Irreversibly block a smart card
TERMINATE DF	Irreversibly block a DF
TERMINATE EF	Irreversibly block an EF

The administrative commands are described in the TS 102 222 specification (see Table 19.4), which comes from the 3GPP environment and was originally intended for the USIM. However, there is no fundamental difference between SIMs and USIMs as far as the administrative commands for smart card file systems are concerned, so in practice this standard has become firmly established in the SIM environment as well.

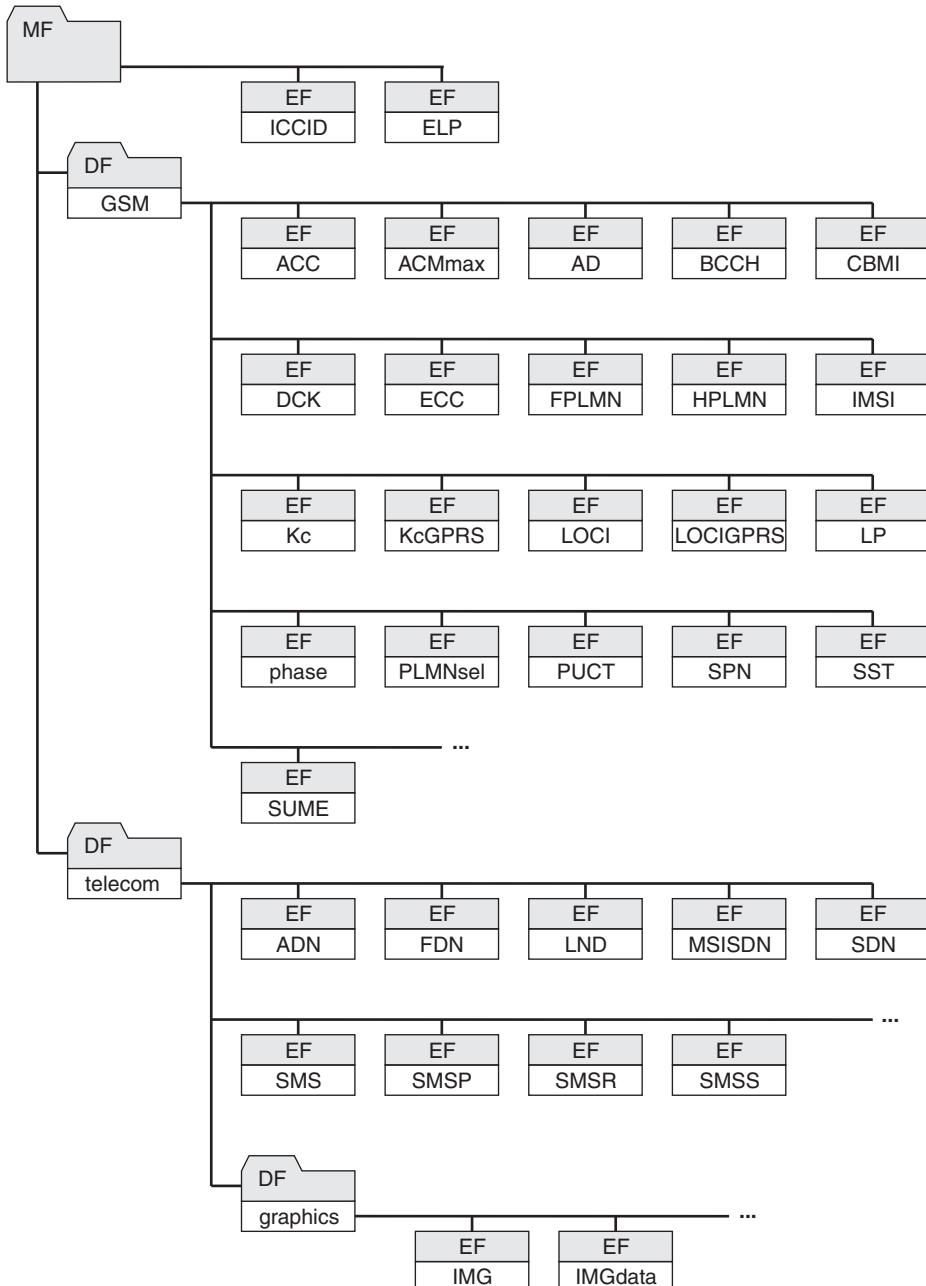
#### 19.4.4.2 SIM files

As illustrated in Figure 19.13, the SIM has a hierarchical file system with an MF and two DFs directly below the MF. EFs containing data for the application are located under the MF and in the DFs. The EFs may have transparent, linear fixed, or cyclic file structures. The files typically present in a SIM files are described in Table 19.5.

The file identifiers (FIDs) of the SIM files have a special feature, which is that the first byte of each DF under the MF always has the value ‘7F’, while DFs and EFs in the DF<sub>GSM</sub> have the value ‘5F’. EFs directly below the MF must have ‘2F’ as the first byte of their FID, while EFs in the DF<sub>TELEKOM</sub> directory must have the value ‘6F’. These conventions are a remnant of the early days of smart cards, and they have long since ceased to have any practical significance.

The access conditions for all files are state-oriented and are specified individually for each file for the four file access commands READ, UPDATE, INVALIDATE, and REHABILITATE. There are 16 states for file access, numbered from 0 to 15 in increasing order of security. State 0 as an access condition means always, or in other words that the file can always be accessed using the associated command. State 15 represents the opposite extreme, which is that the file can never be accessed using the associated command. State 1 means that access is allowed following successful verification of CHV 1, which means PIN 1. Similarly, State 2 requires successful verification of CHV 2 before the file can be accessed. State 3 is not presently used and is reserved for future use. States 4 to 14 are reserved for administrative use, which means that the network operator can access files using these access conditions by using special PIN codes or authentications.

All EFs containing general information about the smart card, such as the unique card serial number (ICCID), are located directly below the MF. All EFs relevant to the GSM system are located in the DF<sub>TELEKOM</sub> directory. A typical example of such EFs is the EF holding the abbreviated dialing numbers. DF<sub>GSM</sub>, by contrast, contains all EFs holding information specific to the network operator, such as the IMSI.



**Figure 19.13** Summary of the most important SIM files. The contents of these files are described in more detail in Table 19.5 on the following page

**Table 19.5** Typical SIM files according to TS 51.011, with the coding of the data elements and illustrative decoded examples

MF	Root directory
Description:	This is the root directory of the entire SIM
File:	FID = '3F00'
DF <sub>TELECOM</sub>	Telecom directory
Description:	This directory holds all files specific to these services
File:	FID = '7F10'
DF <sub>GSM</sub>	GSM directory
Description:	This directory holds all files specific to the GSM network
File:	FID = '7F20' or '7F21' (for compatibility with older-model GSM1800 mobile telephones)
DF <sub>GRAPHICS</sub>	Graphics directory
Description:	This directory holds all files containing graphics data
File:	FID = '5F50'
MF.EF <sub>ELP</sub>	Extended language preference (ELP)
Description:	This file holds an extended list of the preferred languages for the user interface
File:	FID = '2F05'; structure: transparent, file size: $2n$ bytes; accesses: READ: always; UPDATE: CHV 1
Coding:	Each country code consists of two alphanumeric characters according to ISO 639 using the TS 23.038t alphabet bytes 1 and 2: highest-priority language ... bytes ( $2n - 1$ ) and $2n$ : lowest-priority language
Example:	'64 65' → highest-priority language: German '65 6E' → second highest-priority language: English '66 72' → third highest-priority language: French '65 73' → lowest-priority language: Spanish
MF.EF <sub>ICCID</sub>	ICC identification (ICCID)
Description:	This file holds a unique smart card identification number.
File:	FID = '2FE2'; structure: transparent, file size: 10 bytes; accesses: READ: always; UPDATE: never
Coding:	Sequential number, BCD-coded, left-justified and right-padded with 'F' as necessary byte 1, bit 1 ... bit 4: digit 1 byte 1, bit 5 ... bit 8: digit 2 byte 2, bit 1 ... bit 4: digit 3 etc.
Example:	'98 94 20 00 00 10 81 85 39 11' → '89 49 02 00 00 01 18 58 93 11'
DF <sub>GSM</sub> .EF <sub>ACM</sub>	Accumulated call meter
Description:	This file holds the number of charge units accumulated since a particular starting time
File:	FID = '6F39'; structure: cyclic, $3n$ bytes; accesses: READ: CHV 1; UPDATE: CHV 2
Coding:	bytes 1 ... 3: accumulated number of charge units

**Table 19.5** (*continuation*) Typical SIM files according to TS 51.011

$DF_{GSM}.EF_{ACMmax}$	Accumulated call meter maximum (ACMmax) Description: This file holds the maximum allowable number of charge units File: FID = '6F37'; structure: transparent, 3 bytes; accesses: READ: CHV 1; UPDATE: CHV 2 Coding: bytes 1 . . . 3: maximum charge units
$DF_{GSM}.EF_{PLMN}$	Forbidden public land mobile network (FPLMN) Description: This file holds a list of forbidden network operators File: FID = '6F7B'; structure: transparent, 12 bytes; accesses: READ: CHV 1; UPDATE: CHV 1 Coding: bytes 1 . . . 3: forbidden PLMN 1 bytes 4 . . . 6: forbidden PLMN 2 etc. Example: See $EF_{PLMNsel}$ for the data structure and an example 'FF FF FF FF FF FF 62 F2 20' '62 F2' → MCC → '262' → Germany '10' → MNC → '01' → Germany T-Mobile D1
$DF_{GSM}.EF_{HPLMN}$	Home public land mobile network search period (HPLMN) Description: This file holds a time interval for searching for the home network File: FID = '6F31'; structure: transparent, 1 byte; accesses: READ: CHV 1; UPDATE: administrator Coding: Time interval (in minutes) for searching for the home network; coding according to GSM 02.11 Example: '05' → search for home network every 5 minutes
$DF_{GSM}.EF_{IMSI}$	International mobile subscriber identity (IMSI) Description: This file holds the international subscriber identity number. File: FID = '6F07'; structure: transparent; file size: 9 bytes; accesses: READ: CHV 1; UPDATE: administrator Coding: byte 1: IMSI length in bytes byte 2, bits 1 . . . 3: 100 byte 2, bit 4: IMSI parity; coding according to TS 04.08 byte 2, bits 5 . . . 8: IMSI digit 1 bytes 3 . . . 9: IMSI digits 2 . . . 10 IMSI = MCC    MNC    serial number of the network operator, BCD-coded and right-padded with 'F' as necessary Coding: see $EF_{PLMNsel}$ for MCC and MNC coding Example: '08 92 62 01 71 00 10 92 67' '08' → length → 8 bytes '9'    '2 62' → MCC → Germany '01' → MNC → Germany T-Mobile D1 '71 00 10 92 67' → serial number of the network operators
$DF_{GSM}.EF_{KC}$	Kc key Description: This file holds the key Kc for data encryption on the air interface File: FID = '6F20'; structure: transparent; file size: 9 bytes; accesses: READ: CHV 1; UPDATE: CHV 1 Coding: bytes 1 . . . 8: Kc key; byte 9, bits 1 . . . 3: serial number of the key

(Continued)

**Table 19.5** (*continuation*) Typical SIM files according to TS 51.011

DF <sub>GSM</sub> .EF <sub>LOCI</sub>	Location information (LOCI)
Description:	This file holds information about the current location of the mobile telephone
File:	FID = '6F7E'; structure: transparent; file size: 11 bytes; accesses: READ: CHV 1; UPDATE: CHV 1
Coding:	bytes 1 . . . 4: TMSI (temporary mobile subscriber identity) bytes 5 . . . 9: LAI (location area information) byte 10: TMSI TIME (not used from Phase 2 onwards) byte 11: location Update Status (b3 . . . b1 = 000: updated; b3 . . . b1 = 001: not updated; b3 . . . b1 = 010: forbidden PLMN; b3 . . . b1 = 011: forbidden location area Coding according to TS 04.08
Example:	'5F 40 96 46 62 F2 10 80 04 FF 00' '5F 40 96 46' → TMSI '62 F2 10 80 04' → LAI 'FF' → TMSI TIME → not used '00' → location update status → updated
DF <sub>GSM</sub> .EF <sub>LP</sub>	Language preference (LP)
Description:	This file holds a list of the preferred languages for the user interface
File:	FID = '6F05'; structure: transparent; file size: $n$ bytes, $n \geq 1$ ; accesses: READ: always; UPDATE: CHV 1
Coding:	according to TS 23.038 byte 1: highest-priority language ... byte $n$ : lowest-priority language
Example languages:	Valid for GSM alphabet according to TS 23.023 '00': German '01': English '02': Italian '03': French '04': Spanish '05': Dutch '06': Swedish '07': Danish '08': Portuguese '09': Finnish '0A': Norwegian '0B': Greek '0C': Turkish '0D': Hungarian '0E': Polish '0F': unspecified language
Example:	'00 01 03 05' '00' → highest priority language: German '01' → second highest priority language: English '03' → third highest priority language: French '05' → lowest priority language: Dutch
DF <sub>GSM</sub> .EF <sub>PHASE</sub>	Phase information
Description:	This file holds information about the phase supported by the SIM
File:	FID = '6FAE'; structure: transparent; file size: 1 byte; accesses: READ: always; UPDATE: administrator
Coding:	byte 1: 00 = Phase 1; 02 = Phase 2; 03 = Phase 2+ '02' → Phase 2
Example:	

**Table 19.5** (*continuation*) Typical SIM files according to TS 51.011

---

$DF_{GSM}.EF_{PLMNsel}$	Public land mobile network selector (PLMNsel)
Description:	This file holds a list of the preferred network operators
File:	FID = '6F30'; structure: transparent, $3n$ bytes ( $n = 8$ ); accesses: READ: CHV 1; UPDATE: CHV 1
Coding:	bytes 1 ... 3: PLMN with the highest selection priority bytes 4 ... 6: PLMN with the second-highest selection priority Data structure: 2-byte MCC (mobile country code)    1-byte MNC (mobile network code); BCD-coded according to TS 04.08 with high and low nibbles swapped; 'FF FF FF' → entry not used
Example	262: Germany
MCC codes:	208: France 234: Great Britain 222: Italy 232: Austria 310: USA
Example	01: Germany T-Mobile D11
MNC codes	02: Germany Vodafone D2
for Germany:	03: Germany E-plus 07: Germany O2
Example:	'62 F2 20 72 F0 10 32 F4 01 32 F2 30 32 F0 10 62 F2 10 62 F0 20 42 F0 10 22 F8 10', remainder 'FF' '62 F2' → MCC → '262' → Germany '20' → MNC → '02' → Germany Vodafone D2 etc.
$DF_{GSM}.EF_{SST}$	SIM service table (AAT)
Description:	This file holds a table of available and activated services in addition to voice service
File:	FID = '6F38'; structure: transparent; file size: = 2 bytes; accesses: READ: CHV 1; UPDATE: administrator
Coding:	byte 1, bits 1 and 2: service 1 byte 1, bits 3 and 4: service 2 byte 1, bits 5 and 6: service 3 byte 1, bits 7 and 8: service 4 byte 2, bits 1 and 2: service 5 etc. Bit coding: b1, b3, b5, b7 = 1/0 service available / not available b2, b4, b6, b8 = 1/0 service enabled / not enabled
Example	Service 1: disable CHV verification
services:	Service 2: abbreviated dialing numbers (ADN) Service 3: fixed dialing numbers (FDN) Service 4: short message service (SMS) (text messaging) Service 18: service dialing numbers (SDN) Service 35: status report for text messages Service 38: GPRS

---

(Continued)

**Table 19.5** (continuation) Typical SIM files according to TS 51.011

Example:	'DF 3F DF FF 03' = 1101 1111    0011 1111    1101 1111    1111 1111    0000 0011 11 → PIN verification disabling available and enabled 11 → abbreviated dialing numbers available and enabled 01 → fixed dialing numbers available but not enabled 11 → short message service available and enabled etc.
DF <sub>TELECOM</sub> .EF <sub>ADN</sub>	Abbreviated dialing numbers (AND) Description: This file holds the abbreviated dialing numbers. Each record holds a name and the associated dialing number File: FID = '6F3A'; structure: linear fixed, record size: $n + 14$ bytes; accesses: READ: CHV 1; UPDATE: CHV 1 Coding: bytes 1 . . . $n$ : name coded in characters according to TS 23.038 byte ( $n + 1$ ): length of the BCD-coded dialing number in bytes byte ( $n + 2$ ): type of dialing number, coded according to TS 04.08 z. B.: '81': unknown dialing number type, ISDN dialing number scheme z. B.: '91': international dialing number, ISDN dialing number scheme bytes ( $n + 3$ ) . . . ( $n + 1$ ): BCD-coded dialing number with upper and lower nibbles swapped bytewise bytes ( $n + 13$ ) and ( $n + 14$ ): pointer to supplementary data for this entry in EFCCP and EFEXT1, normally not used (set to 'FF') Unused bytes are set to 'FF'. Example 1: Record content: '57 4F 4C 46 47 41 4E 47 FF FF FF FF FF FF FF FF 07 91 94 98 69 35 24 46 FF FF FF FF FF FF' '57 4F 4C 46 47 41 4E 47' → 'Wolfgang' 'FF FF FF FF FF FF FF' → not used '07' → length of the dialing number (7 bytes) '91' → international dialing number, ISDN dialing number scheme '94 98 69 35 24 46' → dialing number 49 89 96 53 42 64 'FF FF FF FF' → not used 'FF FF' → EF <sub>CCP</sub> and EF <sub>EXT1</sub> not used Example 2: Record content: '57 4F 4C 46 47 41 4E 47 FF FF FF FF FF FF FF FF FF 07 81 80 99 56 43 62 F4 FF FF FF FF FF' '57 4F 4C 46 47 41 4E 47' → 'Wolfgang' 'FF FF FF FF FF FF FF' → not used '07' → length of the dialing number (7 bytes) '81' → unknown dialing number type, ISDN dialing number scheme '80 99 56 43 62 F4' → dialing number 089 96 53 42 64 'FF FF FF FF' → not used 'FF FF' → EF <sub>CCP</sub> and EF <sub>EXT1</sub> not used
DF <sub>TELECOM</sub> .EF <sub>FDN</sub>	Fixed dialing numbers (FDN) Description: Fixed dialing numbers can be stored in this file as necessary. These dialing numbers are used if the subscriber is only allowed to dial certain numbers File: FID = '6F3B'; structure: linear fixed, record size: $n + 14$ bytes; accesses: READ: CHV 1; UPDATE: CHV 2 Coding: Same as EF <sub>ADN</sub> Example: See EF <sub>ADN</sub>

**Table 19.5** (*continuation*) Typical SIM files according to TS 51.011

DF <sub>TELECOM</sub> .EF <sub>LND</sub>	Last number dialed (LND) The most recently dialed numbers are stored in this file (optional file) File: FID = '6F44'; structure: cyclic, record size: $n + 14$ bytes; accesses: READ: CHV 1; UPDATE: CHV 1 Coding: Same as EF <sub>ADN</sub>
DF <sub>TELECOM</sub> .EF <sub>MSISDN</sub>	Mobile station ISDN number (MSISDN) This file holds the dialing number of the mobile station File: FID = '6F40'; structure: linear fixed, record size: $n + 14$ bytes; accesses: READ: CHV 1; UPDATE: CHV 1 Coding: Same as EF <sub>ADN</sub>
DF <sub>TELECOM</sub> .EF <sub>SDN</sub>	Service dialing numbers (SDN) This file holds the service dialing numbers, such as dialing numbers for directory assistance or schedule information File: FID = '6F49'; structure: linear fixed, record size: $n + 14$ bytes; accesses: READ: CHV 1; UPDATE: administrator Coding: Same as EF <sub>ADN</sub>
DF <sub>TELECOM</sub> .EF <sub>SMS</sub>	Short message service (SMS) (text messages) This file belongs to the short message service. It holds the text messages sent to and received from the network File: FID = '6F3C'; structure: linear fixed, record size: 176 bytes; accesses: READ: CHV 1; UPDATE: CHV 1 Coding: byte 1: individual record status: '00' = free record '01' = message received from network and read '03' = message received from network and unread '05' = message sent to network '07' = message to be sent to network bytes 2 . . . 176: message coded according to GSM 03.40; unused bytes at the end of the message are set to 'FF'. <i>Coding of messages from the network to the mobile telephone:</i> byte 2: number of bytes in the SMSC dialing number, including the dialing number type next 2–12 bytes: SMSC dialing number: '81' = unknown type of dialing number (no '+'); '91' = international of dialing number (with '+'); data nibblewise swapped next byte: control data (usually '04') next byte: number of digits in the dialing number of the sender, excluding the dialing number type next 2–12 bytes: dialing number of the sender, with data nibblewise swapped next byte: protocol tag ('00' = text message) next byte: data coding ('00' = GSM standard alphabet) next 7 bytes: SMSC time stamp, with data nibblewise swapped: year    month    day    hours    minutes    seconds    time zone ('00' = GMT) next byte: number of characters in the message

(Continued)

**Table 19.5** (continuation) Typical SIM files according to TS 51.011

next 1–140 bytes: message (the text portion is compressed if the GSM standard alphabet is used, which means the 7-bit codes are continuously packed into bytes)

*Coding of messages from the network to the mobile telephone:*

byte 2: number of bytes in the SMSC dialing number

next 2–12 bytes: SMSC dialing number: ‘81’ = unknown type of dialing number (no ‘+’); ‘91’ = international of dialing number (with ‘+’); data nibblewise swapped

next byte: relative time of the mobile telephone (usually ‘FF’)

next byte: message reference

next 2–12 bytes: dialing number of the destination, with data nibblewise swapped

next byte: protocol tag (‘00’ = text message)

next byte: data coding (‘00’ = GSM standard alphabet)

next  $n$  bytes: term of validity of the message:

- 1 . . . 143:  $t = (n + 1) \times 5$  min
- 144 . . . 167:  $t = 12$  h +  $(n - 143) \times 30$  min
- 168 . . . 196:  $t = (n - 166) \times 1$  day
- 197 . . . 255:  $t = (n - 192) \times 1$  week

next byte: number of characters in the message

next 1–140 bytes: message (the text portion is compressed if the GSM standard alphabet is used, which means the 7-bit codes are continuously packed into bytes)

*Example text message from the network to a mobile telephone:*

‘01 07 91 94 71 01 67 05 00 04 0C 91 94 71 71 46 53 42 00 00 00 60 52 31 63  
 15 00 17 C8 A0 93 28 AC 0E 91 20 62 51 0A 1A 22 93 D0 65 50 4A 2D 3A  
 01’ || remainder of record is ‘FF’;

‘01’ → message received from network and read

‘07’ → number of bytes in the SMSC dialing number, including the dialing number type

‘91 94 71 01 67 05 00’ → +49 17 10 76 50 00 SMSC dialing number

‘04’ → no further messages

‘0C’ → 12 → number of digits in the dialing number of the sender, excluding the dialing number type, is 12

‘91 94 71 71 46 53 42’ → sender number = +49 17 17 64 35 24

‘00’ → text message

‘00’ → GSM standard alphabet

‘00 60 52 31 63 15 00’ → 00 06 25 13 36 51 00 SMSC time stamp  
 → 25.06.00 13:36:51 time zone 0 (GMT)

‘17’ → 23 → number of characters in the message = 23

‘C8 A0 93 28 AC 0E 91 20 62 51 0A 1A 22 93 D0 65 50 4A 2D 3A 01’  
 → message: ‘Handbuch der Chipkarten’

*Example text message from a mobile telephone to the network:*

‘07 02 81 F0 11 FF 00 81 00 00 00 08 D7 27 D3 78 0C 3A 8F FF’ || remainder of record is ‘FF’

‘07’ → message to be sent to network

‘02’ → number of bytes in the dialing number, including this length data †

‘81’ → unknown dialing number †

**Table 19.5** (continuation) Typical SIM files according to TS 51.011

---

	<p>'F0' → control data          '11' → relative time of mobile telephone          'FF' → message reference †          '00' → length of the destination dialing number: 0 †          '81' → unknown dialing number †          '00' → text message          '00' → GSM standard alphabet          '00' → term of validity †          '08' → number of characters in the message: 8          'D7 27 D3 78 0C 3A 8F FF' → message: 'WOLFGANG'</p> <p><i>Note 1: The record structure depends on the implementation in the actual mobile telephone and is not generally standardized</i></p> <p><i>Note 2: After this SMS record has been read from the SIM, the data elements marked with † above are expanded before being sent from the mobile telephone. After the message has been sent to the network, the first byte of this record is changed from '07' to '05'</i></p>
DF <sub>TELECOM</sub> .EF <sub>SMSP</sub>	<p>Short message service parameters (SMSP)</p> <p>Description: This file belongs to the short message service. It holds the settings for sending text messages</p> <p>File: FID = '6F42'; structure: linear fixed, record size: 28 + n byte; accesses: READ: CHV 1; UPDATE: CHV 1</p>
DF <sub>TELECOM</sub> .EF <sub>SMSS</sub>	<p>Short message service status (SMSS)</p> <p>Description: This file belongs to the short message service. It holds the status of the stored text messages</p> <p>File: FID = '6F43'; structure: linear fixed, record size: 2 + n bytes; accesses: READ: CHV 1; UPDATE: CHV 1</p> <p>Coding: byte 1: last used SMS message reference number according to GSM 03.40 byte 2: b1 = 0: no space available for the message in SIM memory b1 = 1: space available for the message in SIM memory b2 ... b7: RFU; set to 1</p> <p>Example: '70 FF'          '70' → last used SMS message reference number          'FF' → memory space available in the SIM</p>

---

In total, 70 different EFs are defined in the TS 51.011 specification, of which only 12 (with a total data content of approximately 110 bytes) are mandatory. The rest of the EFs are optional, so their presence in the file system of the SIM depends on the network operator and the available services. In addition to the files defined in the specification, the network operator can install its own files in the SIM file tree for maintenance or administrative purposes. In practice, intensive use is made of this possibility, with the result that typically around 40 files holding approximately 12 KB of user data are present in the SIM.

Some of the EFs in the file tree of the SIM must be written especially often. One example is the LOCI (location information) EF. This file stores the currently valid temporary mobile subscriber identity (TMSI) along with the supplementary location area information (LAI). The data in this file must be changed for each change in base station and each new call.

Consequently, a SIM operating system must support a special file attribute called ‘high update activity’. The technical implementation of this involves storing several file bodies under a single file header. If an error occurs in a file body, the operating system automatically switches to a replacement file body.

This file attribute dates from the time when EEPROM pages had only 10 000 guaranteed write/erase cycles. However, technical refinements have increased the number of cycles to around half a million, which means that this attribute has effectively become obsolete. Nevertheless, it is still present in the GSM specifications, although current smart card operating systems often do not treat files with this attribute any differently from files without it.

The original intention was to replace GSM smart cards every two years in order to avoid failures due to the limited number of EEPROM write/erase cycles. However, since practically no problems have arisen in this regard up to now, most network operators replace smart cards only in the event of actual failure. This yields considerable cost savings for the provider because it only has to deal with the logistics of replacing defective cards. The number of cards that must be replaced is also reduced considerably by the fact that the useful life of most cards is significantly longer than two years. This markedly decreases procurement costs, since it is only necessary to replace smart cards when they no longer work properly. Practical experience has shown that cards only need to be replaced every seven to ten years.

#### ***19.4.4.3 Example of a typical command sequence***

As illustrated in Sequence Diagram 19.1, reading dialing numbers from an EF with a record-oriented structure, such as EF<sub>ADN</sub>, is a good practical example of a typical command sequence. The first step is to select the appropriate file in the proper directory. As the number of records in the file is left to the discretion of the network operator, the file size must first be determined so that the number of records (entries) in the file can be determined using the record length. After this, the records containing a dialing numbers can be read using READ RECORD.

#### ***19.4.4.4 Authentication of the SIM***

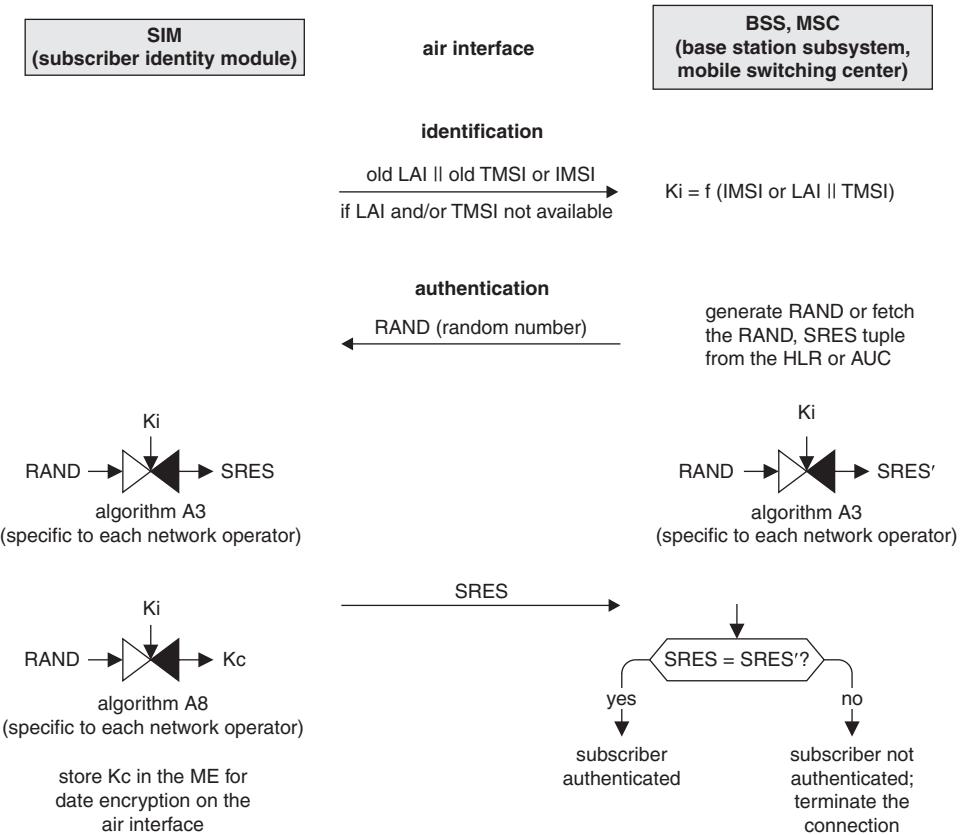
In addition to storing data, one of the primary functions of the SIM is authentication with respect to the GSM network. This consists of a unilateral authentication of the SIM by the background system, as illustrated in Figure 19.14. The background system verifies the authenticity of the SIM, but the SIM does not verify the authenticity of background system. If the authenticity of the SIM is confirmed, the network operator knows that the call can be billed to the owner of the mobile telephone. However, this unilateral authentication has the disadvantage that the user of the mobile telephone cannot be certain that he is connected to an authentic network instead of a counterfeit network. As a consequence, it is possible to eavesdrop on conversations by using a suitable device, called an IMSI catcher, even if the secret keys are not known.

An IMSI catcher operates on the principle of establishing its own radio cell and interposing itself in the air interface between a genuine base station and the mobile stations. With respect to the mobile stations, it acts like a base station, and with respect to the genuine base station, it acts like a mobile station. This form of attack would not be possible with mutual authentication accompanied by encryption of all voice data between the SIM and the background system, as illustrated in Figure 19.15. The roles of the various components of the GSM system in data encryption and decryption are depicted in Figure 19.16.

Terminal (IFD)		Smart card (ICC)
SELECT		
<i>Command</i> [DF <sub>Telecom</sub> ]	→	...
IF (return code = OK)	←	Return code := file selection result
THEN file selection successful		<i>Response</i> [return code]
ELSE abort		
SELECT		
<i>Command</i> [EF <sub>ADN</sub> ]	→	...
IF (return code = OK)	←	Return code := file selection result
THEN file selection successful		<i>Response</i> [return code]
ELSE abort		
GET RESPONSE		
<i>Command</i> []	→	Determine the file size <i>s</i> Determine the record length <i>m</i> <i>Response</i> [ <i>s</i>    <i>m</i>    return code]
IF (return code = OK)	←	
THEN command successfully executed		
ELSE abort		
Compute the number of records <i>n</i>	→	Test CHV
<i>n</i> := <i>s</i> ÷ <i>m</i>		
VERIFY CHV		
<i>Command</i> [CHV1]		return code := result of CHV test <i>Response</i> [return code]
IF (return code = OK)	←	
THEN CHV test successful		
ELSE abort		
FOR <i>x</i> := 1 TO <i>n</i> (	→	record data := content read from record <i>x</i>
READ RECORD		
<i>Command</i> [record <i>x</i> ]		
IF (return code = OK)	←	<i>Response</i> [record data    return code]
THEN record read successfully		
ELSE abort		
) // FOR		

**Sequence Diagram 19.1** A typical command sequence for reading abbreviated dialing numbers from EF<sub>ADN</sub>

The SIM is identified by a number that is unique within the entire GSM system. It has a maximum length of eight bytes and is called the international mobile subscriber identity (IMSI). The subscriber can be identified by the IMSI in all GSM networks everywhere in the world. In order to protect the confidentiality of the subscriber identity as much as possible in the network, whenever possible a temporary mobile subscriber identity (TMSI) is used instead of the IMSI. The TMSI is generated from the visitor location register (VLR) and is thus valid only within a portion of the GSM network concerned. However, the combination of the TMSI and the location area information (LAI) is unique within the entire GSM network. After a TMSI has been assigned, it is used exclusively for all further identification transactions. The

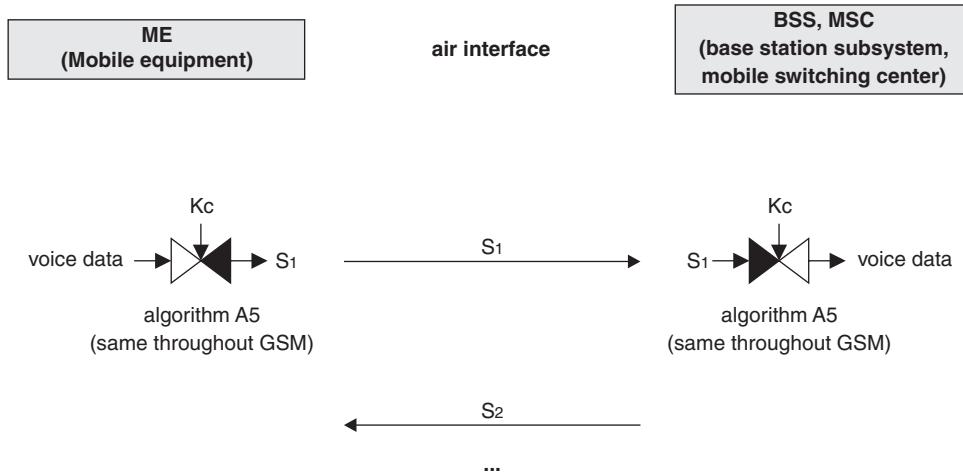


**Figure 19.14** Identification and subsequent authentication of the SIM by the GSM background system using the A3 and A8 cryptographic algorithms, which are specific to the network operator. Key  $Kc$  is subsequently used for encrypting the data transmitted between the mobile station and the base station over the air interface

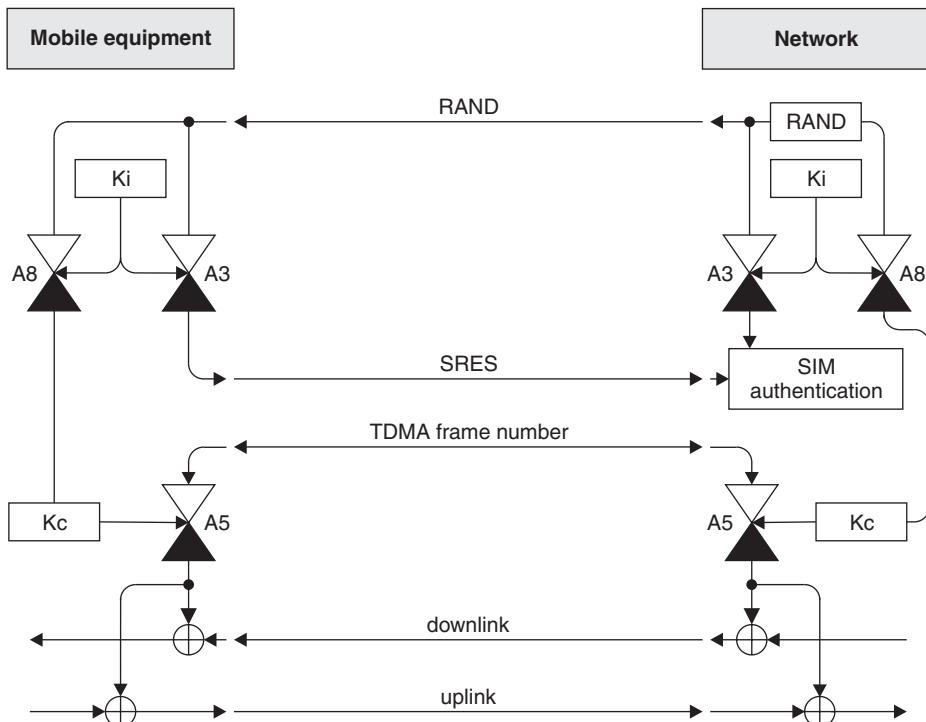
relationship between the IMSI and the TMSI is stored in the visitor location register (VLR) as long as the TMSI is in use. If the TMSI is not known in the VLR, then by way of exception the IMSI must be transmitted in plaintext over the air interface in order to identify the subscriber.

The card-specific keys for authentication and encrypting data on the air interface can be derived from the IMSI. However, the SIM cannot encrypt data for the air interface, since the data processing and transmission capacity of a smart card are not adequate for real-time encryption of voice data. Instead, the SIM computes a derived temporary key for transmission encryption and passes it to the mobile equipment. The mobile equipment has a high-performance encryption unit in the form of a signal processor, which can encrypt and decrypt voice data on the air interface in real time. The encrypted data on the air interface is usually decrypted back into plaintext by the base station controller (BSC).

If a subscriber wishes to make a call, the mobile station establishes a connection to the base station with the best reception and sends it the TMSI from the SIM memory along with the LAI, or in exceptional cases it sends the IMSI. If the subscriber is in the territory of the home network, a set of authentication and encryption data called the authentication triple is



**Figure 19.15** Encryption of the data transmitted between the mobile station and the base station over the air interface, using the A5 cryptographic algorithm and the secret key  $K_c$ . This must be preceded by the identification and authentication of the SIM by the GSM background system as shown in Figure 19.14 on the facing page



**Figure 19.16** Functional overview of the cryptographic functions of the SIM, the mobile equipment, and the background system in the GSM network

generated by the authentication center (AuC). This data set includes the cipher key ( $K_c$ ) for encrypting data on the air interface, a random number (RAND), and the associated signed response (SRES). The advantage of this method is that the secret individual key ( $K_i$ ) and the authentication algorithm, which is partly confidential, never need to leave the authentication center. The authentication triple is then sent to the home location register (HLR).

If the mobile telephone is logged in to its home network, the triple ( $K_c$ , RAND and SRES) is sent to the appropriate visitor location register (VLR). There the result of encrypting the random number (SRES) is requested from the SIM by the mobile switching center (MSC) and compared with the result received from the AuC (SRES'). If the two results match, the SIM has been authenticated and encryption of the data on the air interface using the A5 cryptographic algorithm and associated key ( $K_c$ ) can commence.

If the mobile telephone is instead logged in to a foreign network, the triple is passed to the foreign network concerned, where it can be used in the same way as in the home network. This clearly shows the cleverness of this authentication and encryption scheme, since the A3 and A5 cryptographic algorithms are specific to individual network operators and cannot be computed in a foreign network, even if the secret key is known. Only the A5 cryptographic algorithm, which is used to encrypt data on the air interface, is common throughout the GSM system, so this data can be given suitable cryptographic protection if the key  $K_c$  is known.

The cryptographic algorithms used in the GSM system are generally confidential, which is the only deviation from Kerckhoff's principle<sup>13</sup> in this system. All other information about the system is publicly accessible. Originally, an algorithm called COMP128 was often used for the operator-specific A3 and A8 cryptographic algorithms, but this algorithm was cracked in 1998 because its key was too short and it did not adequately churn the changes in input values. In retrospect, this illustrates the soundness of Kirchhoff's principle, since the inadequacy of this algorithm would have been apparent to cryptologists if it had been made public. The COMP128 cryptographic algorithm is now used only rarely. The alternative algorithms COMP128-2, COMP128-3, and COMP128-4 are normally used instead. The A5 cryptographic algorithm, which is the same throughout the GSM system, is a stream cipher formed by three linear feedback shift registers (LFSRs) with lengths of 19, 22, and 23 bits [Anderson 01], which are incremented by the TDMA frame number.

#### **19.4.4.5 Mobile telephone switch-on and switch-off processes**

When the mobile telephone is switched on, hardware self-tests are run and then the operating system, with a code size of several megabytes, is started up. To temper the impatience of the user during the several seconds taken by this process, more or less entertaining animations are often shown on the display. Once the operating system is fully up and running, one of the next steps is to initiate the activation sequence for the SIM. As described in Table 19.6, the activation sequence is followed by several activities intended to achieve the optimum configuration of the transmission parameters, such as analyzing the ATR and performing a PPS.<sup>14</sup> Following this, it is nowadays common practice to generate a virtual SIM in the memory of the mobile telephone. For this purpose, the mobile telephone reads a large volume of data from the files in the SIM, such as the abbreviated dialing numbers, and stores this data in appropriate data

<sup>13</sup> See also Chapter 7, 'Security Foundations', on page 133

<sup>14</sup> See also Section 8.1, 'Answer to Reset', on page 203 and Section 8.2, 'Protocol Parameter Selection', on page 217

**Table 19.6** Typical mobile telephone activities related to the SIM, shown in correct time sequence. The portrayed activities and command sequences correspond to a typical mobile telephone, although it should be borne in mind that the GSM specifications usually leave the associated details to the discretion of the manufacturer of the mobile telephone. In this example, the SIM used in the mobile telephone essentially has only the functionality necessary for making telephone calls, with the exception of the files for abbreviated dialing numbers and text messages. With a SIM or mobile telephone having a larger functional scope, the activities of the two parties would increase accordingly

Command	Brief description
<i>The user switches on the mobile telephone</i>	
Perform SIM activation sequence	Activate the SIM
Receive ATR	Determine whether a SIM is present and determine the transmission protocol parameters
Perform PPS sequence	Reconfigure the transmission protocol parameters as necessary
<i>The mobile telephone has now established a working communication link with the SIM</i>	
SELECT DF <sub>GSM</sub>	Select the GSM directory and retrieve information about the directory
GET RESPONSE	
SELECT EF <sub>PHASE</sub>	Select and read the EF holding the phase data
READ BINARY	
SELECT EF <sub>LP</sub>	Select the language preference EF, retrieve information about the file structure, and read the file
GET RESPONSE	
READ BINARY	
<i>The user enters a PIN code</i>	
VERIFY CHV STATUS	Test the PIN and then query the state of the retry counter
SELECT EF <sub>SST</sub>	Select the SIM service table EF, retrieve information about the file structure, and read the file
GET RESPONSE	
READ BINARY	
TERMINAL PROFILE	Send information about the properties of the mobile telephone to the SIM (important for SIM Application Toolkit applications)
SELECT MF	Select the root directory
SELECT EF <sub>ICCID</sub>	Select the ICC identification number EF, retrieve information about the file structure, and read the file
GET RESPONSE	
READ BINARY	
SELECT DFGSM	Select the GSM directory
SELECT EF <sub>IMSI</sub>	Select the international mobile subscriber identity EF, retrieve information about the file structure, and read the file
GET RESPONSE	
READ BINARY	
SELECT EF <sub>AD</sub>	Select the administrative data file EF <sub>AD</sub> (which holds the administrative data for the mobile station), retrieve information about the file structure, and read the file
GET RESPONSE	
READ BINARY	
SELECT EF <sub>LOCI</sub>	Select and read the location information EF
READ BINARY	
SELECT EF <sub>KC</sub>	Select and read the cipher key EF
READ BINARY	

(Continued)

**Table 19.6** (continuation) Typical mobile telephone activities related to the SIM

Command	Brief description
SELECT EF <sub>BCCH</sub>	Select and read the broadcast control channels file (EF <sub>BCCH</sub> ), which holds network-specific information
READ BINARY	
SELECT EF <sub>FPLMN</sub>	Select and read the forbidden PLMN EF
READ BINARY	
SELECT EF <sub>HPLMN</sub>	Select and read the HPLMN search period EF
READ BINARY	
SELECT DF <sub>TELECOM</sub>	Select the telecom directory
SELECT EF <sub>SMSS</sub>	Select the SMS status EF (which holds information about stored text messages), retrieve information about the file structure, and read the file
GET RESPONSE	
READ BINARY	
SELECT EF <sub>SMSP</sub>	Select the SMS parameters EF, retrieve information about the file structure, and read the file
GET RESPONSE	
READ BINARY	
SELECT EF <sub>SMS</sub>	Select the SMS EF, retrieve information about the file structure, and read all $n$ records of the file
GET RESPONSE	
$n$ times READ RECORD	
SELECT EF <sub>ADN</sub>	Select the SMS dialing numbers EF, retrieve information about the file structure, and read all $n$ records of the file
GET RESPONSE	
$n$ times READ RECORD	
<i>The mobile telephone is now ready to make a call or transmit data</i>	
<i>The user makes a call</i>	
SELECT DFGSM	Select the GSM directory
RUN GSM ALGORITHM	Authenticate the SIM with respect to the background system
GET RESPONSE	
SELECT EF <sub>KC</sub>	Select the cipher key (Kc) EF and write the updated cipher key to the EF
UPDATE BINARY	
SELECT EF <sub>LOCI</sub>	Select the location information EF and write the updated location information to the EF
UPDATE BINARY	
SELECT EF <sub>BCCH</sub>	Select the broadcast control channels EF ( EF <sub>BCCH</sub> )and write network-specific data to this EF
UPDATE BINARY	
<i>The user switches off the mobile telephone</i>	
SELECT EF <sub>LOCI</sub>	Select the location information EF and write the updated location information to the EF
UPDATE BINARY	
SELECT EF <sub>BCCH</sub>	Select the broadcast control channels EF ( EF <sub>BCCH</sub> )and write network-specific data to the EF.
UPDATE BINARY	

fields in the mobile telephone. The purpose of this is to ensure fast read and write access to the SIM data, which would otherwise not be possible due to the low data transmission rate between the mobile telephone and the SIM and the write cycle time of the EEPROM. Consequently, mobile telephones usually work primarily with a copy of the SIM data located in their own memories. Of course, this technique cannot be used with all the data in the SIM. For instance, activities such as PIN verification and authentication must always be performed together with the SIM because the data related to these functions is not allowed to leave the SIM.

One of the secondary benefits of using a virtual SIM is that it considerably increases the life expectancy of the actual SIM by dramatically reducing the required number of EEPROM write operations. This significantly reduces wear on certain files in the SIM that would otherwise occur as a result of frequent write accesses. A typical example is the EF<sub>LOCI</sub> file, which holds information about the current location of the mobile telephone. The portion of the EEPROM that holds this file is subject to especially heavy wear in mobile telephones that change GSM cells frequently, for which reason it is assigned the ‘high update activity’ attribute. If the data in this file is normally updated in the RAM of the mobile telephone instead, the problem of excessive write accesses to the EEPROM in the SIM is effectively eliminated.

After critical operations, the data in the virtual SIM located in the memory of the mobile telephone is written back to the SIM to update the contents of the SIM files. This is often performed asynchronously by the mobile station using a background task, so the user is not aware that it is happening. This file synchronization at critical points is also important because the SIM should always hold nearly current data in its EEPROM in the event of a sudden loss of power, such as will happen if the batteries are removed. For instance, it would be extremely annoying if removing the batteries resulted in the loss of all of the dialing numbers painstakingly entered into the telephone during the most recent session.

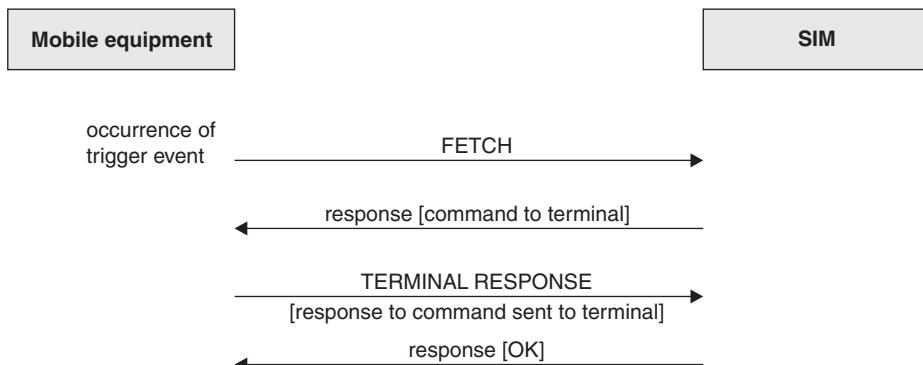
When the mobile telephone is switched off, the user usually sees only a brief sequence of animated characters on the display. However, all the data in the virtual SIM is written to the physical SIM while this is happening in order to bring it up to date. This is followed by a SIM deactivation sequence, and finally the operating system of the mobile telephone is shut down.

The procedures and mechanisms described above are not part of the GSM specification. Consequently, they are generally implemented in quite different ways in different models of mobile telephones. The example described here should be regarded as only a possible and technically effective implementation. Particularly with the GSM system, it should also be borne in mind that mobile telephones up to ten years old are still in use. It can confidently be assumed that older telephones do not have virtual SIMs, but instead perform all read and write operations directly in the SIM.

#### **19.4.4.6 SIM Application Toolkit (SAT)**

In the original specifications for the GSM system, the GSM card was simply regarded as a means to identify the user, using a PIN and an authentication token in the interest of billing security, that was independent of the mobile telephone. However, in the course of time the desire to utilize the GSM card for additional functions, particularly value-added services, became increasingly pronounced. For instance, a mobile telephone could also be used to check the balance of a bank account or to receive interesting news, such as football scores and daily horoscopes. However, the modest capabilities of the GSM were not sufficient to permit the technical implementation of these value-added services (VAS). The response to this was the development of the TS 51.014 GSM specification, ‘SIM Application Toolkit’ (SAT). The first version of this specification was published in 1996 by ESTI.

The SIM Application Toolkit enables the SIM to directly access functions of the mobile equipment, such as driving the display, polling the keypad, sending text messages, and other functions needed in connection with a value-added service. Ultimately, the SIM Application Toolkit is a construction kit that allows almost any desired application to be implemented in a SIM.



**Figure 19.17** The extended protocol process between the mobile equipment and the SIM for the proactive commands of the SIM Application Toolkit as specified in TS 51.014. The data part of the smart card's response to a command contains a command to the terminal. The terminal executes this command and returns the associated response to the smart card in the data part of another command. The process shown here is based on the sent APDUs and shows only successful results

A number of new commands had to be defined for the SIM Application Toolkit. A noteworthy feature of these commands is that they are sent to the mobile equipment by the SIM, which requires a certain change in mental attitude. The data part of these proactive commands is BER-TLV coded.<sup>15</sup> This makes it possible to easily achieve expansion capability while ensuring downward compatibility. However, the greatest advantage of this is the enormous flexibility obtained by using TLV-coded data.

With the SIM Application Toolkit, it was necessary to devise a way to circumvent the usual master–slave arrangement between the terminal and the smart card for the SIM, but for reasons of compatibility, modifying the transmission protocol was not allowed. The solution to this problem was relatively simple. In a process called polling, the mobile equipment periodically sends the query command STATUS to the SIM at a specifiable interval (such as every 30 s), and if necessary the SIM can indicate in its response that a command for the mobile equipment is ready to be sent and should be fetched from the SIM. In practice, the mobile equipment does not maintain the configured polling interval especially strictly, but this does not cause any problems. This circumvention of the master–slave principle using the process shown in Figure 19.17 is called ‘SIM proactivity’, and the associated commands, which are described in Table 19.7, are called proactive commands.

This technique effectively reverses the master–slave relationship between the mobile equipment and the SIM. This makes it possible for the card to take the initiative in polling the keypad, showing its data and menu structures on the display of the mobile telephone, and emitting a beep. The SMS mechanism can also be used to exchange data between the SIM and the GSM background system over the air interface. For example, this can be used to periodically poll a news server, with the results being presented on the display of the mobile telephone as an e-mail message or text message.

The commands that make this mode possible are FETCH, TERMINAL RESPONSE, and ENVELOPE. The mobile equipment uses FETCH to retrieve a command from the SIM. After

<sup>15</sup> See also Section 6.1, ‘Data Structures’, on page 109

**Table 19.7** Proactive SIM smart card commands specified for the SIM Application Toolkit in TS 51.014. Note that the commands listed here are sent to the terminal by the smart card, rather than from the terminal to the smart card as usual. Certain commands cannot be used if they are not supported by the configuration of the mobile equipment

Command	Brief description
<i>User interface</i>	
DISPLAY TEXT	Show a character string or icon passed with the command on the display of the mobile telephone
GET INKEY	Show a character string or icon passed with the command on the display of the mobile telephone, followed by requesting a character from the keypad
GET INPUT	Show a character string or icon passed with the command on the display of the mobile telephone, followed by requesting one or more characters from the keypad
LANGUAGE NOTIFICATION	Advise the mobile equipment of the language used by the SIM Application Toolkit in text fields
PLAY TONE	Request the mobile equipment to issue a tone
SELECT ITEM	Transfer a selection list to the mobile equipment with a request to have the user select an item
SET UP IDLE MODE TEXT	Show a character string or icon passed with the command on the display of the mobile equipment if it is switched on but not in use
SET UP MENU	Transfer a menu list to the mobile equipment with the request to integrate it into the menu structure of the mobile equipment
<i>Second card terminal</i>	
GET READER STATUS	Query the status of a supplementary card terminal in the mobile station
PERFORM CARD APDU	Send an APDU to the smart card in a supplementary card terminal in the mobile telephone
POWER OFF CARD	Deactivate the smart card in a supplementary card terminal in the mobile telephone
POWER ON CARD	Activate the smart card in a supplementary card terminal in the mobile telephone
<i>Network interface</i>	
CLOSE CHANNEL	Request the mobile equipment to close a data channel
GET CHANNEL STATUS	Request the mobile equipment to return the status of a data channel
OPEN CHANNEL	Request the mobile equipment to open a data channel
RECEIVE DATA	Request the mobile equipment to receive data via an open data channel
RUN AT COMMAND	Transfer an AT command to the mobile equipment with command execution in the mobile equipment, followed by sending the result to the SIM
SEND DATA	Request the mobile equipment to transmit data via an open data channel
SEND DTMF	Transmit a DTMF signal during a current voice connection
SEND SHORT MESSAGE	Send a text message

(Continued)

**Table 19.7** (continuation) Proactive SIM smart card commands

Command	Brief description
SEND SS	Send a supplementary application (SS) message, which is a control sequence for the network
SEND USSD	Send an unstructured supplementary applications data (USSD) message, which can be used to send any desired data
SET UP CALL	Establish a connection
<i>Miscellaneous</i>	
MORE TIME	Ask the mobile equipment to give the SAT application more time for processing
POLL INTERVAL	Start cyclic polling of the SIM with the specified interval
POLLING OFF	Stop cyclic polling of the SIM
PROVIDE LOCAL INFORMATION	Ask the mobile equipment to provide current location information to the SIM
REFRESH	Advise the mobile equipment that the data content of the SIM has changed, so it should read this data anew
SET UP EVENT LIST	Transfer an event list to the mobile equipment with the request to inform the SIM if one of these events occurs
TIMER MANAGEMENT	Start, stop or configure the eight possible timers in the mobile equipment that can generate an event
LAUNCH BROWSER	Start a microbrowser supported by the smart card operating system

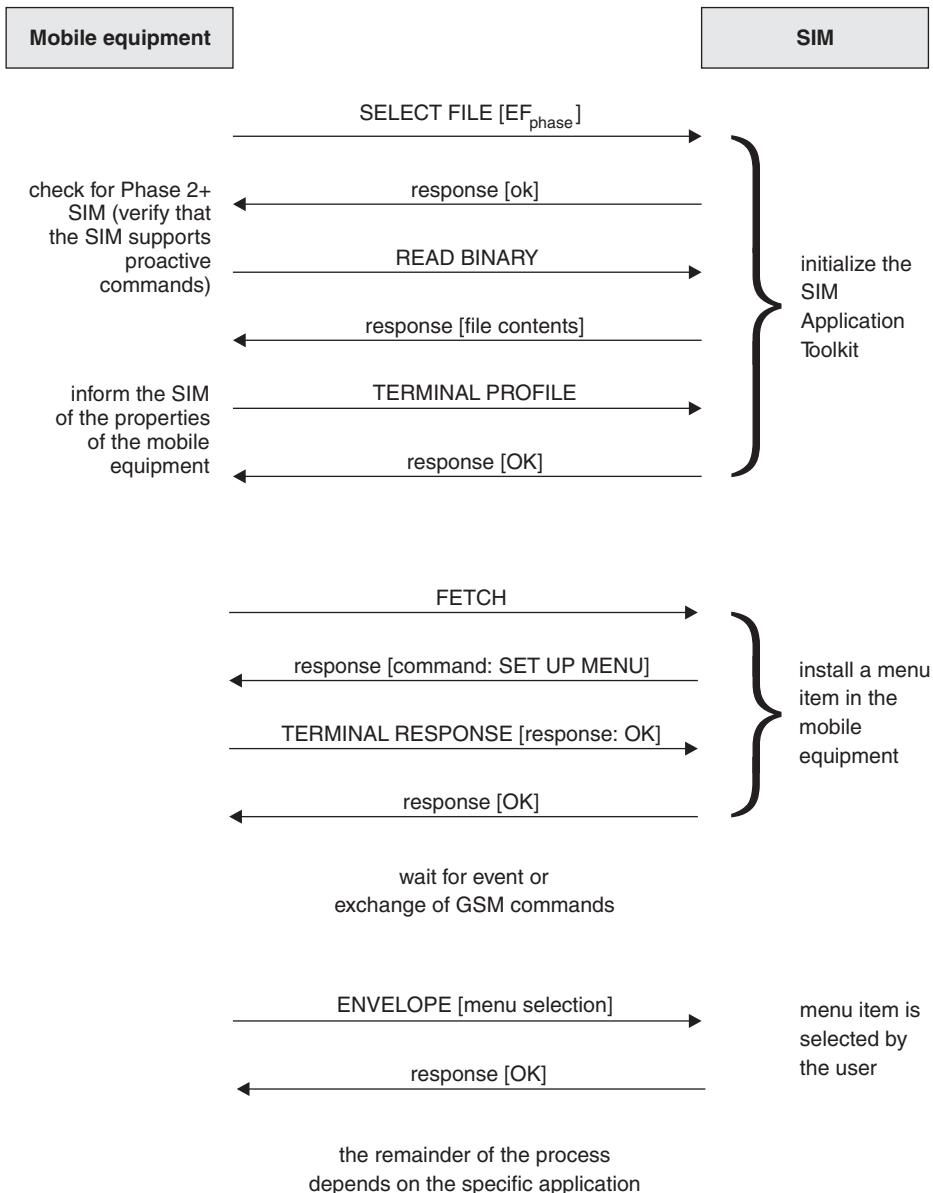
processing this command, the mobile equipment uses the TERMINAL RESPONSE command to return the associated result to the SIM. The ENVELOPE command allows data to be transferred to the SAT application of the mobile equipment.

In addition to this proactivity, the SIM can inform the mobile equipment of certain events for which the SIM must be explicitly notified if they occur.

There are several ways to launch a SAT-based supplementary application in the SIM. The simplest manner involves an action on the part of the user. For example, if the user selects a certain function from the mobile telephone menu and this function is based on a supplementary SIM application, the mobile equipment sends a corresponding command to the SIM. The further course of events is then determined by the supplementary application in the SIM. Certain events in the mobile telephone, such as call setup, call termination or changing network cells, can also be used to invoke a SAT-based application in the SIM. The simplest way to invoke a SAT application in the SIM is cyclic polling of the SIM by the mobile equipment. With these three basic invocation methods, all practically relevant SIM-based supplementary applications can be implemented at relatively moderate cost.

The actual functions for controlling supplementary applications in the SIM Application Toolkit are provided by executable program code, which can be written in any desired programming language, such as assembly language, C, or Java.

A typical sequence of activities with a SIM Application is shown in Figure 19.18. Toolkit application is as follows. After the SIM activation sequence, the mobile equipment reads a variety of data, including the EF<sub>Phase</sub> file, which indicates which GSM phase the SIM



**Figure 19.18** A typical example of using the SIM Application Toolkit to install a supplementary menu entry in the mobile equipment. The process shown here is based on the sent APDUs and shows only successful results

supports. If the code for Phase 2+ is stored in this file, the terminal concludes that the SIM Application Toolkit is fully supported. Following this, the terminal uses the TERMINAL PROFILE command to inform the SIM of its properties that are relevant to the SIM Application Toolkit. This completes the initialization, after which any other commands related to the GSM application that do not belong to the SIM Application Toolkit can be sent as necessary.

Typically, the next process is installing a selection menu in the mobile equipment. This is done by placing BER-TLV coded data for the menu in the response to a FETCH command requested by the SIM and sent by the mobile equipment. The mobile equipment then integrates the new selection menu in its menu structure and acknowledges this with a confirmation in the subsequent TERMINAL RESPONSE command. The selection menu is thus installed in the mobile equipment and enabled. After this, the usual GSM commands can again be exchanged and processed. When the user of the mobile telephone selects the menu entry, the ENVELOPE command is sent to the SIM with information about the selected menu entry. The SIM confirms receipt of the command and can then initiate a wide variety of processes, depending on the application and user selection. For example, the application could query a share price on the securities exchange in response to the user selection.

This function could be implemented in a wide variety of ways. A simple solution would be to send a text message to a server of the network operator with a request for the current share price of a specific company. If all goes well, the server will send a text message reply to inform the application in SIM of the share price, and the application can then use the DISPLAY TEXT command to inform the mobile telephone user of the current share price.

This is a very simple example of what can be done with the SIM Application Toolkit, but it clearly shows that the toolkit is a very powerful resource for generating supplementary applications in the SIM and that these applications can be implemented relatively easily. Difficulties can arise in certain situations, such as when supplementary applications must be implemented using functions of the mobile equipment that are not supported by the SIM Application Toolkit. Other well-known obstacles to SAT-based supplementary applications are the large variety of display formats and fundamental incompatibilities or implementation errors in the mobile equipment. However, they can all be overcome with a certain amount of effort and suitable experience. In summary, it can be said that the SIM Application Toolkit still provides the most technologically mature and secure means to implement supplementary applications in mobile telephones.

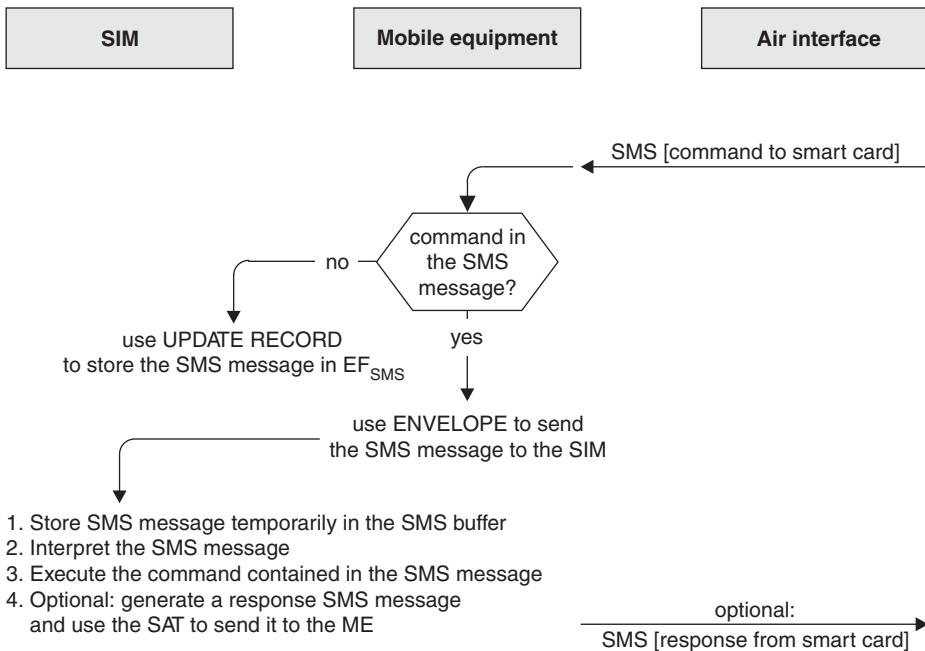
The SIM Application Toolkit forms a very powerful interface for supplementary applications in the SIM, and it can be integrated into the existing system without any modifications.

Based on the SIM Application Toolkit, the ETSI Project Smart Card Platform (EP SCP) expert group defined a generic basis for all application toolkits for smart cards in the mobile telecommunication sector. This toolkit is called the Card Application Toolkit (CAT), and it forms the basis for the SIM Application Toolkit (SAT), the USIM Application Toolkit (USAT), and the UIM Application Toolkit (UATK).

#### **19.4.4.7 Over-the-air (OTA) communication**

It is sometimes necessary to establish a direct connection between the background system and a SIM that has already been issued. This communication option is essential for managing existing applications and generating new supplementary applications in the SIM. Consequently, mechanisms are defined in the TS 43.048 specification to allow a secure end-to-end communication link to be established between the background system and the SIM over the air interface.

As this requirement is not covered by the original GSM specifications and it is nearly impossible to change a system of this size, a special technique is used to implement end-to-end communication with the GSM card. As illustrated in Figure 19.19, this consists of using the text messages supported by the system as containers for messages to and from the



**Figure 19.19** Data exchange using text messages passed between the background system and the GSM card. This method is commonly called over-the-air (OTA) communication

SIM. This only requires making certain adjustments to the background system and issuing new smart cards, with all intermediate portions of the system remaining unchanged. However, OTA with text messages is now only one of several possible bearer services, although it is the most widely used option.

OTA communication offers a relatively wide range of protection options for the transmitted data. For instance, the simplest security level consists of using a CRC checksum to protect the data against transmission errors. In the realm of cryptographic protection, a send sequence counter can be added to the data and the data can be encrypted using DES or triple DES (with two or three keys). If necessary, a MAC or a digital signature of the data to be transmitted can be computed.

The operating principle of using SMS as a bearer service is as follows. If the background system wants to send a command (such as READ BINARY) to a particular SIM, it generates a text message to the card concerned and embeds the command in the message, using suitable cryptographic protection mechanisms. When the mobile telephone containing the SIM in question logs in to the system, the text message is sent via the signaling channel. This does not require establishing a voice connection via a traffic channel. From the coding of the text message as specified in GSM 03.40, the mobile equipment recognizes that it contains data for the SIM and uses the SIM Application Toolkit ENVELOPE command to send it to the SIM. In this case, the mobile equipment does not use the UPDATE RECORD command to store the received text message in the EF<sub>SMS</sub> file, which is otherwise done automatically with incoming text messages. The SIM places the text message received with the ENVELOPE command in a separate buffer. If the text message is chained with other text messages, the next task of the

SIM is to restore the correct message sequence, since SMS does not guarantee that messages are received in the order that they are sent.

The SIM then interprets the received message, extracts the command or commands from it, and processes the command or commands. The SIM can optionally generate another text message in response. This message is sent to the mobile equipment in the response to a FETCH command requested by the SIM, and the mobile equipment forwards it to the background system in the usual manner via the service channel.

Using this artifice, it is possible to establish a bidirectional end-to-end link between the background system and the SIM that is fully transparent to all of the intermediate system components. This allows the SIM to be accessed in the same way as a SIM located in a terminal linked to a PC. This communication channel can be used for file management tasks such as modifying existing data in files. A common use for OTA communication is updating the service dialing numbers stored in the SIM. It can also be used to perform significantly more complex tasks. For instance, with SIMS based on Java Card it can be used to download executable program code for supplementary applications in the form of applets. The possibilities that OTA communication offers to network operators are immense. Unfortunately, many parts of TS 43.048 do not provide precise bit-level specifications, but instead have the nature of a standard that defines a wide range of more or less loosely defined options that individual card producers can use as they see fit in their SIMs. This naturally has a detrimental effect on the mutual compatibility of SIMs from different producers, with the consequence that network operators must maintain libraries for individual smart card producers in their background systems in order to achieve operational compatibility.

#### **19.4.4.8 Remote file management (RFM)**

The mechanisms provided by OTA enable direct end-to-end communication between the background system and the SIM. This provides the data transmission basis for remote management of the files in the SIM, which is called RFM (remote file management) in GSM terminology. This bearer-independent basic functionality is specified in TS 43.048, which in turn is based on the requirements of TS 42.048.

Only certain SIM commands can be used for remote file management (see Table 19.8), but a considerable functional scope can be achieved with these commands. They are divided into input commands, which send data to the SIM, and output commands, which request data from the SIM. In addition to individual commands, the background system can send a list of commands to the SIM in an OTA message. However, this is subject to the restriction that only the final command in the list can request data from the SIM. The main reason for this restriction, which does not cause any difficulties in practice, is that it significantly simplifies the remote file management software in the SIM. Due to this restriction, several OTA messages must be sent to the SIM if several files or records must be read.

The program flow for remote file management is shown in Figure 19.20, and the operating principle of remote file management using SMS as a bearer service can be briefly illustrated using a typical practical example. If the background system wants to modify an abbreviated dialing number stored in the EF<sub>ADN</sub> file, it can proceed as follows. In the first OTA message, which may consist of a series of text messages, it selects EF<sub>ADN</sub> by sending a SELECT command with the path to the file in DF<sub>Telecom</sub>. The final command in this OTA message is READ RECORD with a record number known to the background system, which causes the service number to be read from the file and returned via OTA. If the service number is not

**Table 19.8** Smart card commands that can be sent to the SIM for remote file management, according to TS 43.048

Input commands	Output commands
SELECT	READ BINARY
UPDATE BINARY	READ RECORD
UPDATE RECORD	GET RESPONSE
SEEK	
INCREASE	
VERIFY CHV	
CHANGE CHV	
DISABLE CHV	
ENABLE CHV	
UNBLOCK CHV	
INVALIDATE	
REHABILITATE	

current, an UPDATE RECORD command is sent to the SIM in another OTA message, and the appropriate record is updated with the new number.

Naturally, caution must be exercised in using remote file management to modify files that are used during an open session. A typical example of such files is EF<sub>SST</sub>, which holds the SIM service table. This table lists all the available SIM services, regardless of whether they are actually enabled. Under certain conditions, modifying the content of this file can cause the mobile telephone to behave unpredictably, and in the worst case it can render the SIM unusable.

The SIM also contains two EFs for which modification is simply forbidden. These are EF<sub>ICCID</sub>, which holds the identification number of the smart card (ICCID), and EF<sub>Kc</sub>, which holds the key for encrypting data transmitted between the mobile station and the base station over the air interface. There is no logical reason to modify either of these files. The ICCID is the unique identification number of the smart card and plays no role in normal operational activities, while the Kc key is computed by the SIM individually for each session, so it would be pointless to modify it via RFM. If it were modified during an open session, the connection to the network might be broken because the mobile equipment would be using an incorrect key for encrypting the data on the air interface.

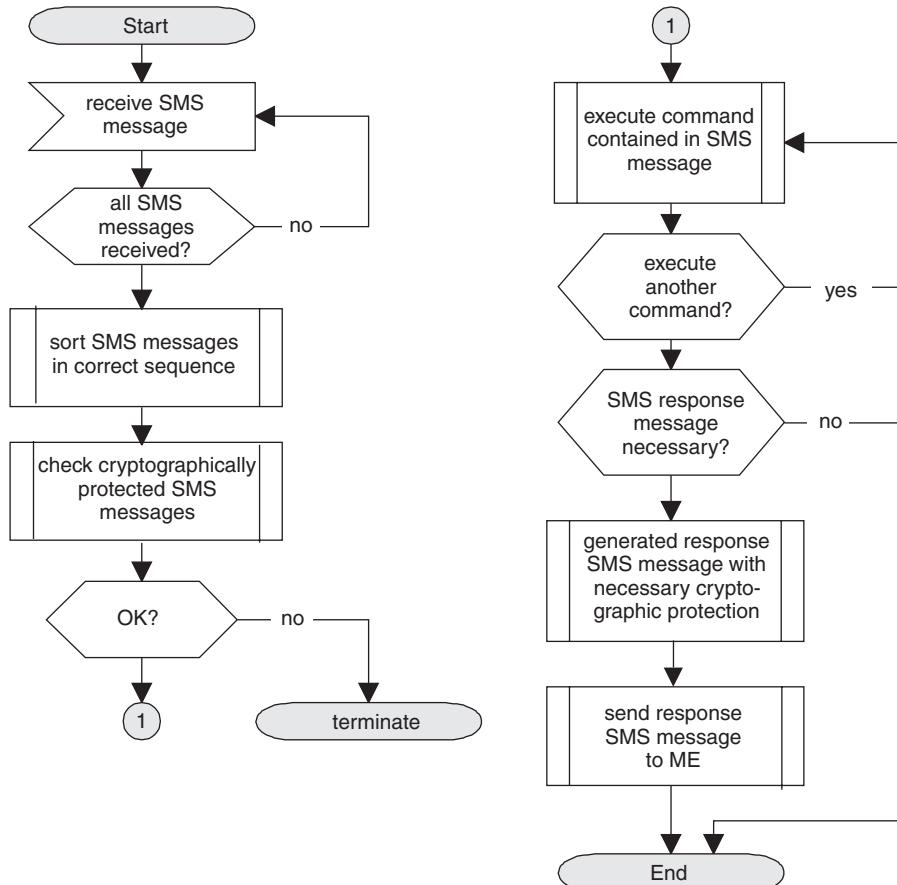
#### 19.4.4.9 Remote applet management (RAM)

The TS 43.048 specification includes a section on remote applet management, which is similar to remote file management. With this functionality, applications based on Java Card can be managed using a direct end-to-end link between the background system and the SIM.

A general prerequisite for this is that the smart card concerned is a SIM compliant with TS 43.019, which is essentially based on the Java Card 2.x specifications.<sup>16</sup> All management commands for applets and packages are based on the Global Platform specification, which is effectively the industry standard for these mechanisms.<sup>17</sup>

<sup>16</sup> See also Section 13.16.2, ‘Java Card’, on page 499

<sup>17</sup> See also Section 13.13, ‘Application Management with Global Platform’, on page 485



**Figure 19.20** Flow chart of the program sequence for remote file management (RFM) via the air interface using OTA as specified in TS 43.048. Remote file management is essentially performed by the processes in the upper right branch of the flow chart; the rest of the processes serve to establish secure communication in accordance with TS 43.048

The application management functions include loading, installing, deleting, locking, and unlocking Java applets in the SIM and retrieving parameter data from these Java applets. Similar mechanisms are defined for loading packages into the SIM and deleting packages from the SIM. The commands available for remote applet management are listed in Table 19.9.

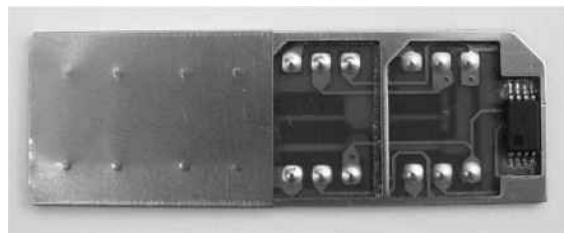
All of the processes and mechanisms are basically independent of any particular bearer service, but SMS is presently the most commonly used medium for OTA management of applets and packages in SIMs. If SMS is used as the bearer service, data transmission to and from the SIM takes place in exactly the same manner as described above for OTA remote file management.

#### 19.4.4.10 Dual IMSI

In the business world, the charges for using a company mobile telephone are usually paid by the company. However, if the user of the mobile telephone also uses it for private calls, the

**Table 19.9** Smart card commands that can be sent to the SIM for remote applet management over the air interface, as specified by TS 43.048. These commands correspond to the Global Platform specification in terms of functionality and coding

Input commands	Output commands
DELETE	GET STATUS
SET STATUS	GET DATA
INSTALL	
LOAD	
PUT KEY	



**Figure 19.21** An opened plug-in adapter with room for two SIMs. This is the simplest hardware implementation of dual-ISIM capability. The user changes from one SIM to the other one by switching the mobile telephone off and back on; this process is controlled by the microcontroller shown at the right. The SIMs must be trimmed to the right size before they are inserted

associated charges must later be reimbursed to the company, which is only possible with an itemized telephone bill. In practice, the user will probably make private calls at the expense of the company, as otherwise the user would have to carry two telephones for the two types of calls. Needless to say, this is too much to expect of anyone. Dual-IMSI mobile telephones are intended to provide a solution to this problem. A dual-IMSI SIM has a supplementary file that holds two IMSIs and sometimes two Ki keys, depending on the implementation. This file is not specified in any standard. A supplementary SAT-based application in the SIM generates a new menu entry in the mobile equipment that allows the user to select either business mode or private mode. Depending on the user's selection, a particular IMSI is copied to EF<sub>IMSI</sub>, and if necessary another associated Ki key is used. This gives the mobile telephone two separate identities in terms of the IMSI, and the network operator can generate two separate bills. This makes it possible to separate the charges for private calls from the charges for business calls.

Dual IMSI functionality can also be implemented in hardware by using an adapter that can hold two SIMs, such as the example shown in Figure 19.21.

This functionality can be further extended, for example by enabling the fixed dialing numbers for the business IMSI in order to restrict the possible destinations of business calls. With this arrangement, the mobile telephone user cannot charge private calls to the company because only the numbers of importance to the company can be dialed. If the second IMSI is instead enabled by using the supplementary application in the SIM, the fixed dialing numbers are disabled and the company is not charged for the calls. In this way, a single mobile telephone

can be used to make both business and private calls without creating problems in the allocation of the associated charges.

There is yet another use for multiple IMSIs, although it is not especially elegant in a technical sense. If IMSIs for several different network operators are stored in a SIM and they can be individually selected by the user via a menu, the mobile telephone can be used for manual roaming. The user first selects the IMSI corresponding to the network in whose territory she happens to be located and then logs into this network using the selected IMSI. She subsequently receives separate telephone bills from the individual network operators. This form of roaming with multiple IMSIs is used in practice in many regions of India, for example, due to the lack of regular roaming agreements between some of the network operators.

#### **19.4.4.11 Implementing a home zone**

Some network operators offer special personal rates for a restricted local area. This area is usually a roughly circular zone centered on the place of residence of the subscriber, within which calls are charged at the landline rate instead of the more expensive mobile rate. For the user, the main advantage of this sort of location-specific service is that it eliminates the need for a connection to the regular telephone network and the need to synchronize the abbreviated dialing numbers in the regular telephone and the mobile telephone.

A suitable way to implement this service is to use a SIM-based application generated with the SIM Application Toolkit functions. This also ensures that all information about the home zone is stored in the SIM (which is specific to the person concerned) instead of the mobile telephone or somewhere else.

In the GSM system, each base station continuously transmits a unique identifier over the air interface. This identifier consists of the location area information (LAI) and a cell identity (CI). One approach to implementing home zone capability is to have the mobile telephone send this information to the SIM, where it can be compared with stored data. This data is preferably stored in a file so it can be modified or updated at any desired time by remote file management. A drawback of this approach is the relatively large amount of memory needed in the SIM, since it is necessary to accommodate areas with a high density of base stations, which can result in large LAI and CI lists.

A similar approach would be to utilize the advance timing information on the air interface. With this information, in areas with high cell density the current location of the mobile station could be determined by triangulation with an error of significantly less than 10 meters, which is more than adequate for implementing a home zone.

However, a different solution is often preferred in practice.<sup>18</sup> With this solution, all the base stations belonging to a particular network operator use the cell broadcast service to periodically transmit their location coordinates on the signaling channel. The SIM has an EF containing reference data that is read by the mobile equipment and compared with the received location coordinates. If the mobile equipment determines that the mobile telephone is located within the home zone, a suitable symbol (such as a small house icon) is shown on the display. The background system knows the location of the mobile telephone, so it can switch to a more favorable rate for incoming and outgoing calls. The home zone coordinate data is stored in an

<sup>18</sup> See [Bögeholz 99]

EF in the SIM, so it can easily be modified using remote file management. This also enables convenient home zone setup and relocation using remote maintenance. A drawback of this approach is that it is implemented using special software in the mobile equipment instead of a supplementary application in the SIM generated with the SIM Application Toolkit.

#### 19.4.4.12 Operating principle of SIM lock

‘SIM lock’ is the name given to a technique for binding the mobile equipment to a particular SIM or set of SIMs. The SIM lock function is used by network operators to tie a mobile telephone subsidized by the network operator to a particular SIM and its payment mode for a certain length of time. This function is based on the TS 101 624 specification. The operating principle of the SIM lock is always based on specific data that is stored in both the SIM and the mobile equipment and compared by one of the two components each time the mobile telephone is switched on, with the telephone only being enabled for use if the two data sets match.

Two different implementations of the SIM lock function are used in practice. With the commonly used first method, the mobile telephone reads certain data from the SIM and compares it with data stored in the mobile telephone. This usually consists of the group identifiers, which are stored in the EF<sub>GID1</sub> (group identifier level 1) and EF<sub>GID2</sub> (group identifier level 2) files. These group identifiers can be used to specify classes of SIMs, which can then be used to specify class-based pairings of particular SIMs with particular (subsidized) mobile equipment units. The advantage of this approach is that the SIM and the mobile equipment do not have to be individually married. As an alternative to the group identifiers, the IMSI in EF<sub>IMSI</sub> or other static SIM-specific data stored in EFs is sometimes used as the reference data for pairing the devices.

The second option, which is rarely used in practice, requires the SIM to use the SIM Application Toolkit to read unique data from the mobile equipment and compare it with stored data. If the two data sets match, the SIM enables the mobile telephone for normal use.

It is usually possible to disable the SIM lock, either over the air interface or by entering a secret key in the mobile telephone, so that other SIMs can be used in the mobile equipment previously protected by the SIM lock. The reference data for this is most often the individual mobile equipment identity (IMEI) of the mobile equipment unit.

#### 19.4.4.13 Operating principle of prepaid systems

The proportion of prepaid SIMs ranges from around 30 % to as much as 80 %, depending on the country. The main reasons for using prepaid SIMs are that they provide better cost control and they avoid the need to pay subscriber fees.

Systems designed to work with prepaid SIMs usually operate as follows. A card-shaped voucher, which often has the dimensions of an ID-1 card but is not as thick, has a 13-digit number printed in a scratch field with a protective seal layer.<sup>19</sup> If a user wants to reload his mobile telephone, he purchases a voucher, whose integrity can be seen from the fact that the scratch field is still intact. After scratching off the top layer to reveal the number, the user selects a specific menu and enters the number in the mobile telephone. This reference value

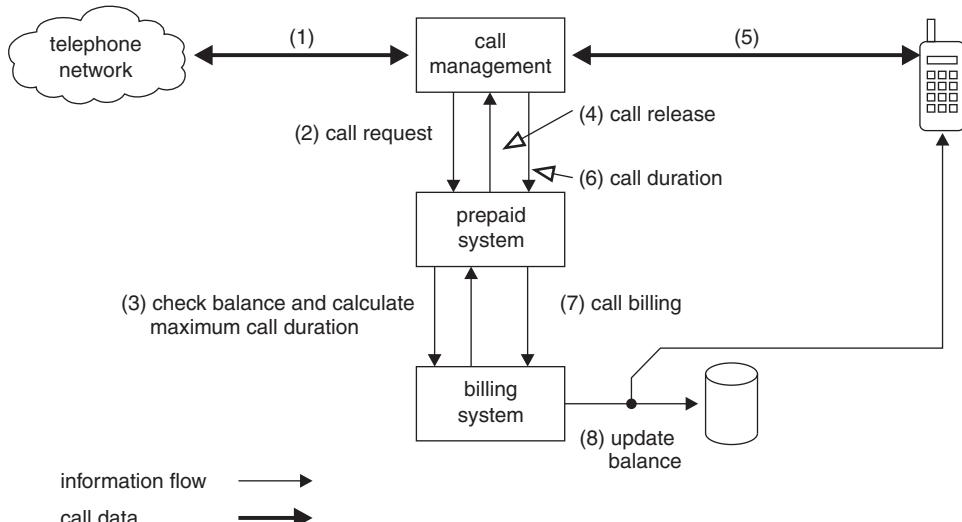
<sup>19</sup> See also Section 3.5.11, ‘Scratch field’, on page 47

is sent directly to the background system, where it is compared with a database value for the issued voucher. If the two values match and the voucher has not been used before, the load amount associated with the reference value is credited to the SIM in the mobile telephone. At this point, the possible implementations diverge.

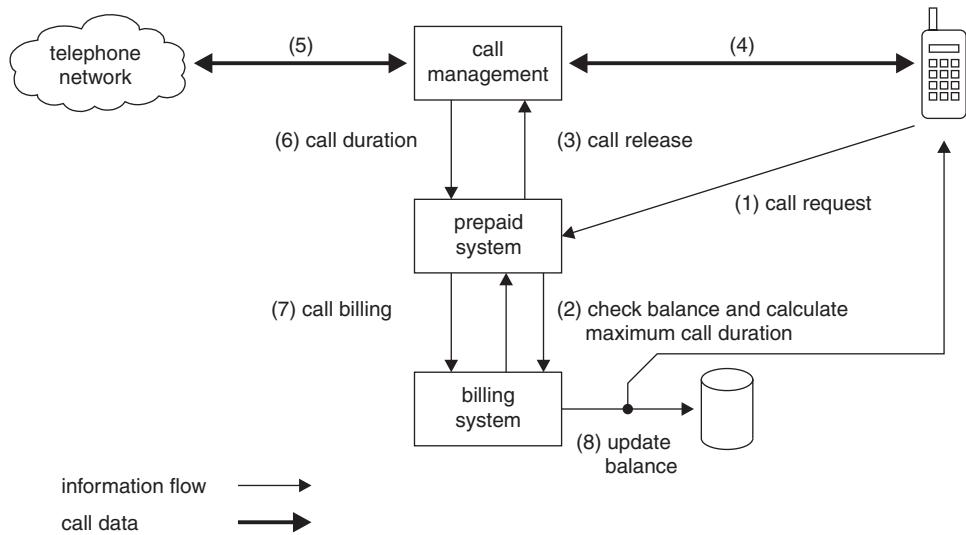
The solution originally envisaged in the GSM specifications consists of a units counter in a file (the accumulated call meter file  $EF_{ACM}$ ) in the SIM. Using the advice of charge data provided by the mobile equipment, the SIM continuously updates this counter and compares it with the maximum value stored in the accumulated call meter maximum value file ( $EF_{ACMmax}$ ). If the counter value reaches the maximum value, the mobile telephone prohibits further calls until the counter value is reset, which can be done by remote file management or some other means. Although this solution is technically feasible, it is not used in practice because communication between the mobile equipment and the SIM is not secure and could be manipulated relatively easily.

In practice, prepaid SIMs are managed by a centralized system, with two different approaches being used. The first approach makes no use of supplementary data in the SIM and operates entirely in the background system. A drawback of this approach is that the background system computer must have real-time capability, which makes it more expensive. With the second approach, a suitable supplementary application must be present in the SIM, but there is no need for general real-time capability in the background system. A disadvantage of the latter approach is that when the prepaid amount has been used up, there may be a delay before the connection is broken.

A typical background system for prepaid SIMs that does not necessarily need to have real-time capability can be described using the components shown in Figures 19.22 and 19.23 on the facing page as examples. A GSM system that supports prepaid SIMs must have a number of



**Figure 19.22** Basic architecture of a system for prepaid SIMs using the SICAP Prepaid Roaming solution as an example. This diagram shows the progress of a call routed to a mobile telephone, with the numbers in parentheses indicating the sequence of events. Call management is part of the GSM background system, and may for example be a component of the mobile switching center (MSC)



**Figure 19.23** Basic architecture of a system for prepaid SIMs using the SICAP Prepaid Roaming solution as an example. The diagram shows the progress of a call originating from the mobile telephone, with the numbers in parentheses indicating the sequence of events. Call management is part of the GSM background system and may for example be a component of the mobile switching center (MSC)

supplementary integrated functions. In this example, these functions include call management as a supplementary component of the mobile switching center (MSC), which maintains an interface to the prepaid system and can route or block calls in real time. The prepaid system is the core element of the system; its function is to coordinate call management and the billing system. The billing system uses a database to manage the individual SIM accounts holding the credit balances. With this arrangement, the SIM contains only a few special commands and corresponding data.

When a call to a mobile telephone arrives at the background system, the call management subsystem first advises the prepaid system that a call is waiting to be routed to a specific mobile telephone with a specific SIM. The prepaid system asks the billing system to calculate the maximum possible call duration based on the outstanding credit balance of the SIM and then sends this information to the call management subsystem. If the credit balance is sufficient, the call management subsystem routes the call, and it terminates the call if the maximum call duration is reached. After completion of the call, the call management subsystem informs the prepaid system of the call duration, and the prepaid system sends this information to the billing system in order to update the credit balance of the account. The updated balance can then be shown on the display of the mobile telephone.

A similar process occurs when a call is made from a mobile telephone. First, a USSD query is issued to determine the maximum call duration with the aid of the prepaid system and the billing system, and this information is sent to the call management subsystem. If the maximum duration is reached during the call, the call is terminated. Otherwise, the account balance is updated after the call is completed and stored in the database accordingly. Naturally, the remaining credit can optionally be displayed on the mobile telephone.

### 19.4.5 Future developments

The GSM application paved the way for the international success of smart cards, and it is still the most important standard for smart cards and smart card operating systems. Although some of the commands and mechanisms in the GSM realm may appear outdated compared with the latest developments in the smart card world, GSM was and still is the pioneer for large-scale and international smart card applications. Ultimately, all subsequent applications can only learn and benefit from the experience gained and problems encountered with this application. In many respects, GSM in the form of the TS 51.011 and TS 51.014 specifications forms the foundation for all later and more elaborate smart card applications.

Even an established system such as GSM must be further developed in order to meet new requirements and satisfy customer desires for additional functions. Using an approach that has proven its worth over many years, this has been done in small steps, leading to changes and extensions such as proactive SIM commands, OTA, WIM, microbrowsers, and RAM. Nevertheless, at a certain point it became necessary to take a major evolutionary step in order to transform all of these incremental extensions, changes and special cases into a new system that was once again self-contained. This step was the specification of the UICC and the USIM as components of the Universal Mobile Telecommunication System (UMTS), which is a logical evolution of the SIM and all of its functions.

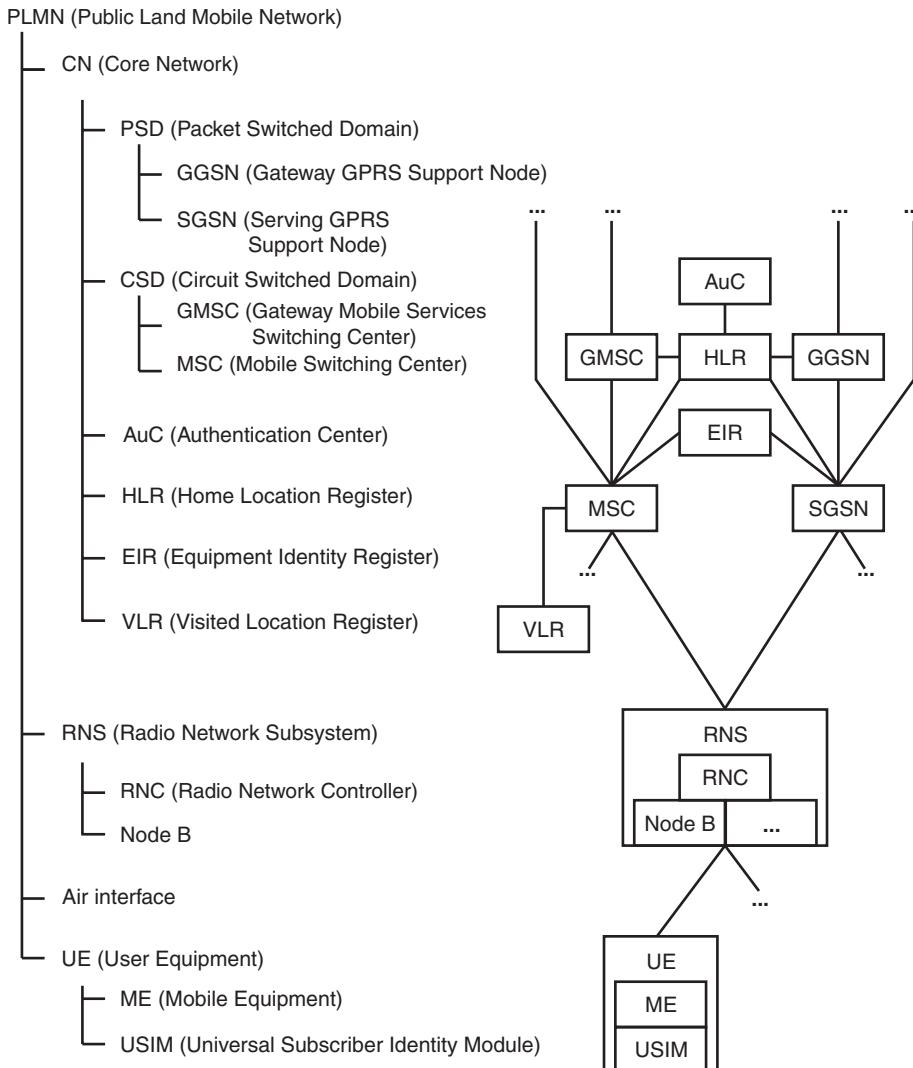
## 19.5 THE UMTS SYSTEM

In 1998, a group of five standardization organizations, consisting of ANSI T1 (USA), ARIB (Japan), ETSI (Europe), TTA (Korea) and TTC (Japan), initiated the Third Generation Partnership Project (3GPP), whose task was to specify a successor to the GSM system in the form of an international IMT-2000-compliant mobile telecommunication system based on the GSM specifications. This mobile telecommunication system, which is generally designated worldwide as the 3G (third-generation) system, is generally known in Europe as the Universal Mobile Telecommunication System (UMTS).<sup>20</sup> This system initially aroused more interest among the general public due to the enormous license fees paid for the necessary frequency spectrum, rather than the new functions it could offer. In Germany alone, network operators paid approximately €50 billion for the UMTS frequency spectrum in a competitive auction, and the total amount paid in Europe was €112 billion. Setting up the network infrastructure absorbed an additional €30 billion in Germany alone.

A relatively short time later, the first UMTS network went into service in Japan in late 2001. The development of UMTS was mainly driven by a few countries, such as Japan, in which subscriber density was already so high that further extension of the existing mobile telecommunication systems was no longer reasonable.

Some of the essential differences between UMTS and GSM are described below. For the air interface, which is called the UMTS radio access network (UTRAN), UMTS uses code division multiple access (CDMA) for communication between the base stations and the mobile stations. In accordance with IMT-2000, the transmission frequency is in the 2000-MHz band

<sup>20</sup> The term UMTS is used consistently in this book instead of 3G because it unambiguously describes a specific mobile telecommunication system that is only one of several 3G systems. For example, the CDMA 2000 mobile telecommunication system is also a 3G system, although it specifies an optional smart card called the removable user identity module (R-UIM) that differs from the USIM for UMTS in many respects



**Figure 19.24** Basic architecture and designations of the major components of a typical mobile telecommunication system according to the TS 123.002 UMTS standard

(wavelength approximately 15 cm). As illustrated in Figure 19.24, the architecture is very similar to that of GSM, with the main difference being that the components of the UMTS system are linked by an IP-based network.

From the smart card perspective, the greatest difference between GSM and UMTS is that UMTS uses a completely redefined security module called the universal subscriber identity module (USIM), which is based on the ISO/IEC 7816 family of standards. For this first time in the world of mobile telecommunication smart cards, this ensures a high level of compatibility with other smart cards that also comply with these standards, such as EMV-2000-compliant smart cards used in payment systems.

**Table 19.10** The most important standards for the USIM system. A more general listing of all 3GPP and ETSI standards in the USIM environment is provided in the directory of standards in the appendix. All of these standards can be obtained free of charge from the ETSI web server [ETSI]

TS 21.111	USIM and IC card requirements
TS 31.102	Characteristics of the Universal Subscriber Identity Module (USIM) application
TS 31.110	Numbering system for telecommunication IC card applications
TS 31.111	Universal Subscriber Identity Module (USIM) Application Toolkit (USAT)
TS 31.120	UICC-terminal interface; physical, electrical and logical test specification
TS 31.121	UICC-terminal interface; Universal Subscriber Identity Module (USIM) application test specification
TS 31.122	Universal Subscriber Identity Module (USIM) conformance test specification
TS 102 221	Smart Cards; UICC-Terminal interface; physical and logical characteristics
TS 102 222	Integrated Circuit Cards (ICC); administrative commands for telecommunication applications
TS 102 223	Smart Cards; Card Application Toolkit (CAT)
TS 102 241	Smart Cards; UICC Application Programming Interface (UICC API) for Java Card
TS 102 268	Smart Cards; Test specification for the UICC Application Programming Interface (API) for Java Card
TS 102 412	Smart Cards; Smart Card Platform Requirements Stage 1
TS 102 588	Smart Cards; Application invocation Application Programming Interface (API) by a UICC web server for Java Card
TS 102 600	Smart Cards; UICC-Terminal interface; Characteristics of the USB interface
TS 102 613	Smart Cards; UICC-Contactless Front-end (CLF) Interface; Part 1: physical and data link layer characteristics
TR 102 151	Smart Cards; Measurement of Electromagnetic Emission of SIM Cards

At this point, we recommend that in order to better understand the functionality of the USIM, you first read the description of the SIM in Section 19.4.4, ‘TMSI’, on page 811, since the SIM and USIM smart cards are very similar and the following description focuses primarily on the differences. From this, you can also clearly see that the UMTS system is an evolutionary successor to the GSM system that incorporates many of the proven principles and mechanisms of the GSM system.

The smart card application in the UMTS system, which resides in a UICC, is normally called the USIM. However, in practice the term USIM is used not only for the application but also for the UMTS smart card, although this is not entirely correct. The USIM is primarily the bearer of the subscriber identity, and its principal function is to ensure the authenticity of the mobile station with respect to the network and vice versa.

The most important standards for the USIM system are listed in Table 19.10, and the basic properties of the USIM are described in Table 19.11. The USIM specification is based on the TS 102 221 specification, which is the fundamental specification for telecommunication smart cards and characterizes the physical and logical parameters of a universal integrated circuit card (UICC). Based on this specification for a general-purpose smart card for telecommunication applications, requirements specification TS 21.111 defines the basis for the USIM application, which in turn is described in detail in TS 31.102. These specifications are complemented by TS 31.111, which contains an extensive description of the USIM Application Toolkit (USAT). The commands for managing applications in a UICC are defined in TS 102 222, which is now also used as a quasi-standard in the SIM environment. Finally, TS 31.122 contains specifications for conformity tests.

**Table 19.11** The characteristic physical, electrical and logical properties of a USIM based on the UICC specification

Property	Remarks
Card format	ID-1, ID-000, or mini-UICC
Supply voltage	UICC: 1.8 and/or 3 and/or 5 V
Transmission protocol	In practice, USIMs typically support the 1.8 and 3 V voltage classes PPS (mandatory) $T = 0$ (mandatory) $T = 0$ (optional)
Logical channels	Up to four
Commands	See the command summary in Table 19.12 on page 853
Access conditions for files	As specified in ISO/IEC 7816-9.

Operating systems for UICCs must comply with the ISO/IEC 78126 family of standards in all of their essential properties, which means they inherently support multiple applications. This applies in particular to the commands, file management, and file access conditions, which are rule-based in accordance with ISO/IEC 7816-9, with access being controlled by an access rule reference EF (EF<sub>ARR</sub>). However, there are many similarities to the SIM with regard to commands and file management.

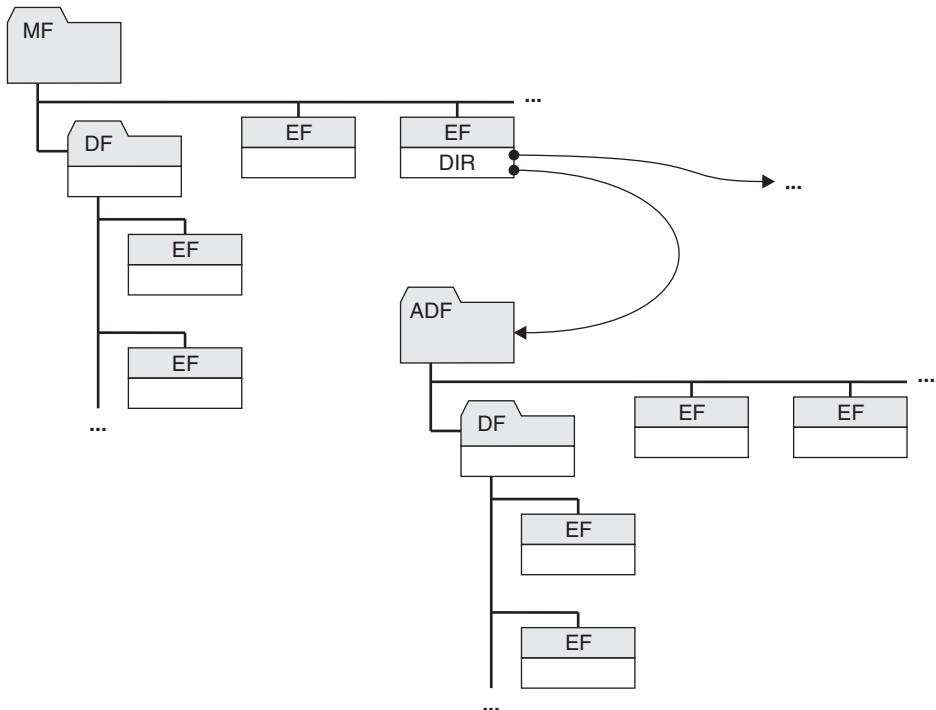
With regard to file types, USIM has a special feature in the form of application dedicated files (ADF). As illustrated in Figure 19.25, an ADF is a special type of DF that contains all of the DFs and EFs of a particular application, but that does not have the MF as its root directory. In this regard, an ADF can be regarded as similar to the MF because it does not have any higher-level file type. An ADF is selected using an AID stored in EF<sub>DIR</sub>.

In the UICC, the USIM is simply one of several applications located in individual ADFs. Due to the multiapplication capability of the UICC, it is not particularly difficult to implement a SIM in addition to the USIM and thereby create a smart card that can be used in both UMTS and GSM systems. This will probably be the standard configuration in the foreseeable future, since the logistics expenses of network operators can be reduced by using a single smart card for both mobile telecommunication technologies.

Most of the files of the USIM application can also be found in the same or similar form in the SIM. The only significant change in this area relates to the storage of dialing numbers. A relatively obscure concept with optional and mandatory files linked by pointers is specified for the USIM. A wide variety of data related to dialing numbers and subscribers can be stored in these files.

A USIM has two PIN codes called PIN 1 and PIN 2, which are fully analogous to CHV 1 and CHV 2 in a SIM. The access conditions for the files are defined such that the card user knows PIN 1 and the cardholder knows PIN 2. This enables flexible management of most of the functions of the mobile equipment.

A set of cryptographic functions named f1, f2, f3, f4, and f5 ('function 1' ... 'function 5') are used for USIM authentication in the UMTS system. In addition to the authentication of the network and the USIM, they are used to provide cryptographically secured communication over the air interface. The kernel of these security functions is a symmetrical cryptographic algorithm that can be parameterized with supplementary linked input values. The USIM specification suggests the Milenage algorithm, whose kernel is based on AES, as the example algorithm for f1 to f5.



**Figure 19.25** The relationships between the MF, DFs and ADFs in the UICC or USIM. An ADF, which holds all the files of an application, can be selected using an AID stored in  $EF_{DIR}$

Just as the SIM has the SIM Application Toolkit, an application toolkit called the USIM Application Toolkit (USAT) is specified for the USIM. Like the SAT, it is used for implementing value-added services in the USIM. ETSI also specified a microbrowser called USAT Interpreter for the USIM, but it was unable to make any inroads on the established WIB and ST browser technologies.<sup>21</sup> Table 19.12 lists the smart card commands defined for use with the USIM.

### 19.5.1 Future developments

With the broadly oriented UICC platform for telecommunication smart cards and the USIM application adapted to the UMTS mobile telecommunication system, a large and important step has been taken toward a modular architecture for smart cards and their applications. In keeping with the evolutionary method proven over more than a decade, these specifications are also undergoing continual refinement and extension. The following examples present a few of the important new functions resulting from this evolution, which will doubtless also become visible to end users in the coming years.

Although the functionality of the SIM Application Toolkit as an application interface is sufficient for nearly all imaginable applications, its text-based user interface no longer

<sup>21</sup> See also Section 19.7, ‘Microbrowsers’, on page 857

**Table 19.12** Smart card commands for the USIM as specified in TS 102 221

Command	Brief description
<i>Security commands</i>	
CHANGE PIN	Change the PIN
DISABLE PIN	Disable PIN queries
ENABLE PIN	Enable PIN queries
UNBLOCK PIN	Reset the PIN retry counter from its terminal count
VERIFY PIN	Verify the PIN
AUTHENTICATE	Authentication of the USIM by the outside world
<i>Commands for operations on files</i>	
INCREASE	Increment a counter in a file
READ BINARY	Read from a file with a transparent structure
READ RECORD	Read from a file with a record-oriented structure
SEARCH RECORD	Search for a character string in a file with a record-oriented structure
SELECT	Select a file
STATUS	Read various data from the currently selected file
UPDATE BINARY	Write to a file with a transparent structure.
UPDATE RECORD	Write to a file with a record-oriented structure
DEACTIVATE FILE	Reversibly block a file
ACTIVATE FILE	Unblock a file
<i>Miscellaneous commands</i>	
GET RESPONSE	Command specific to T = 0 for requesting data from the smart card
MANAGE CHANNEL	Manage logical channels

corresponds to the state of the art. This shortcoming is recognized, but all of the alternatives up to now are still in the prototype stage. In all likelihood, the new user interface will be based on HTML and utilize the browser in the mobile telephone. The browser will receive data from a web server<sup>22</sup> in the USIM and from other entities in the network. The trend in this area is clearly heading in the direction of integrating current Internet technologies in the mobile telecommunication system. The USIM will thus become a device that is fully integrated in the (mobile) Internet.

This can open up a new application area for mobile telephones as a medium for paying for goods and services. The original idea of implementing this capability using the IrDA infrared interface never got past the paper stage, but wireless NFC technology<sup>23</sup> could prove to be the key to success here. This requires a secure element in the telephone that is either permanently installed or implemented as an additional function of the USIM. Communication between the USIM and the contactless front end (CLF) will be implemented using the Single-Wire protocol,<sup>24</sup> which will thus establish a presence in the smart card world. If this entirely new function for mobile telephones is commercially successful, it can lead to many interesting and technically sophisticated extensions of the functionality of the USIM.

<sup>22</sup> See also Section 23.3, ‘Smart Card Web Server (SCWS)’, on page 912

<sup>23</sup> See also Section 10.11, ‘Near Field Communication (NFC)’, on page 348

<sup>24</sup> See also Section 9.6, ‘Single-Wire Protocol’, on page 278

Another development trend is the use of multi-megabyte cards in the telecommunication sector. The application ideas include music with DRM protection, video clips, and configuration (branding) of mobile telephones for specific networks or network operators. However, these large memory volumes require a significantly faster interface between the USIM and the mobile telephone than what is currently available. The USB interface<sup>25</sup> already fulfills all of the technical requirements, but mobile telephones that support this interface are not yet available. Here there is a possibility that an alternative interpretation of the abbreviation ‘GSM’ that became popular in the early 1990s, when a shortage of handsets prevailed for several months, could again become relevant: God sends mobiles.

## 19.6 THE WIRELESS IDENTIFICATION MODULE (WIM)

In 2000, WAP was widely extolled in the mass media as the long-awaited new Internet technology for mobile telephones. For a variety of reasons, including a lack of content, a lack of suitable mobile telephones, low data transmission rates, and disappointing content display on the mobile telephones (some of which were still text-based and monochrome), WAP never established a market presence. WAP is now often used as a synonym for expensive, senseless investment in technology that misses the mark with consumers.

WAP (Wireless Application Protocol) actually refers to a set of specifications for the implementation of links between mobile terminals (mobile telephones, PDAs, etc.) and servers for direct data exchange, using a wireless network such as GSM, CDMA or TDMA. The most common application for WAP nowadays is implementing Internet services on mobile telephones in manner that is largely independent of the mobile telecommunication standard used in the system. Along with the technology referred to in the ordinary usage of the term, WAP also refers to the protocol for communication between the terminal and the background system.

The international standardization body for WAP is the WAP Forum [WAP], which was founded in June 1997 by Phone.com, Ericsson, Motorola, and Nokia and is presently composed of representatives of more than 350 companies. Since June 2000, the specifications for WAP include a security module called the wireless identification module (WIM) for use in the mobile equipment. Only with such a module is it possible to establish secure communication with applications in the background system in a reasonable manner. Two possible versions of the WIM have been specified: as an application in a dedicated smart card, or as a supplementary application in a SIM, USIM or UICC. As dual-slot mobile telephones have not achieved a significant market presence, the second version with the WIM implemented as a supplementary application in an existing smart card will doubtless prevail.

The WIM specification is based on the PKCS #15 specification [RSA] for digital signature applications in smart cards, which are now also specified in a compatible form by ISO/IEC 7816-15. In order to understand the following description properly, you should have at least a cursory knowledge of PKCS #15 signature applications. If necessary, consult Section 23.1.7, ‘Signature cards’, on page 906 for more information.

The primary task of the WIM is to provide cryptographic functions for secure WAP data transmission at the transport level in the form of wireless transport layer security (WTLS).

<sup>25</sup> See also Section 9.4, ‘USB Transmission Protocol’, on page 272

**Table 19.13** The primitives defined in the WIM specification and the associated smart card commands

Primitive	Smart card command
<i>Device control</i>	
Open Service	MANAGE CHANNEL
Close Service	MANAGE CHANNEL
<i>Verification</i>	
Perform Verification	VERIFY
Disable Verification Requirement	DISABLE VERIFICATION REQUIREMENT
Enable Verification Requirement	ENABLE VERIFICATION REQUIREMENT
Change Reference Data	CHANGE REFERENCE DATA
Unblock Reference Data	RESET RETRY COUNTER
<i>Data access</i>	
Open File	SELECT
Close File	—
Read Binary	READ BINARY
Update Binary	UPDATE BINARY
<i>Cryptography</i>	
Compute Digital Signature	MANAGE SECURITY ENVIRONMENT PERFORM SECURITY OPERATION
Verify Signature	MANAGE SECURITY ENVIRONMENT PERFORM SECURITY OPERATION
Get Random	ASK RANDOM
Key Transport	MANAGE SECURITY ENVIRONMENT PERFORM SECURITY OPERATION
Key Agreement	MANAGE SECURITY ENVIRONMENT PERFORM SECURITY OPERATION
Derive Master Secret	MANAGE SECURITY ENVIRONMENT
PHash	MANAGE SECURITY ENVIRONMENT PERFORM SECURITY OPERATION
Decipher	MANAGE SECURITY ENVIRONMENT PERFORM SECURITY OPERATION
<i>Exception</i>	
Exception	This can relate to any command

Interestingly enough, these WIM functions can be used not only in the WAP context, but also in the SIM Application Toolkit environment. This makes the WIM a general-purpose signature application for telecommunication cards, and for this reason it has become a standard component in many SIMs and USIMs.

The WIM specification includes a number of primitives in abstract form that describe basic functions. Table 19.13 summarizes these generic functions, which are used in the WTLS protocol. In a subsequent stage, these primitives are linked to suitable commands or command sequences.

WIM files generally have a transparent structure in order to simplify access and avoid creating unnecessary overhead. This does not make access any more difficult in terms of

**Table 19.14** Summary of the smart card commands defined in the WIM specification

Command	Brief description
<i>Logical channels</i>	
MANAGE CHANNEL	Manage logical channels. Optional command specified in ISO/IEC 7816-4
<i>Verification</i>	
VERIFY	Verify supplied PIN data. Command specified in ISO/IEC 7816-4
ENABLE VERIFICATION REQUIREMENT	Enable PIN queries. Optional command specified in ISO/IEC 7816-8
DISABLE VERIFICATION REQUIREMENT	Disable PIN queries. Optional command specified in ISO/IEC 7816-8
CHANGE REFERENCE DATA	Change PIN code. Command specified in ISO/IEC 7816-8
RESET RETRY COUNTER	Reset a retry counter. Command specified in ISO/IEC 7816-8
<i>Data storage</i>	
SELECT	Select a file. Command specified in ISO/IEC 7816-4
READ BINARY	Read from a file with a transparent structure. Command specified in ISO/IEC 7816-4
UPDATE BINARY	Write to a file with a transparent structure. Command specified in ISO/IEC 7816-4
<i>Cryptographic operations</i>	
MANAGE SECURITY ENVIRONMENT	Modify the parameters for using cryptographic algorithms in the smart card. Command specified in ISO/IEC 7816-8.
PERFORM SECURITY OPERATION	Execute a cryptographic algorithm in the smart card. Command specified in ISO/IEC 7816-8
ASK RANDOM	Request a random number from the smart card. Command specified in ISO/IEC 7816-4
<i>Miscellaneous</i>	
GET RESPONSE	Request data from the smart card; used with the T = 0 transmission protocol. Command specified in ISO/IEC 7816-4

software engineering, since the underlying pointer structure of PKCS #15 enables relatively straightforward and extensible access to the desired data objects. All data is described in ASN.1 format and coded using the Distinguished Encoding Rules (DER).<sup>26</sup>

Along with the usual PKCS #15 files and data objects, the WIM has a peers data object and a sessions data object. These two data objects are used to store specific link data to allow an abbreviated authentication handshake to be used in a subsequent session.

<sup>26</sup> See also Section 6.1, ‘Data Structures’, on page 109

**Table 19.15** Basic properties of a WIM

Property	Remarks
Card format: ID-000	optional
Card format: ID-1	optional
Commands	See list of commands in Table 19.14 on the preceding page
Logical channels	optional
T = 0 transmission protocol	mandatory
T = 1 transmission protocol	optional
Supply voltage: 3 V	mandatory
Supply voltage: 5 V	optional

With regard to its physical properties, the WIM must comply with the relevant GSM specifications, such as TS 51.011. The WIM must support the T = 0 transmission protocol; support for the T = 1 protocol is optional. The terminal can use PPS to select the protocol as necessary. Table 19.15 provides a summary of the basic properties of a WIM.

The commands necessary for executing the functions of the WIM application are fully compliant with ISO/IEC 7816-4 and 7816-8. This makes it relatively easy to implement a WIM in a smart card with a normal multiapplication operating system, such as a UICC.

In order to use the WIM application, a new logical channel to the smart card is opened if this is supported by the operating system. Along with this, the application is first selected using its AID. Here the RID has the value ‘A0 00 00 00 63’ and the associated PIX is the ASCII encoding of ‘WAP-WIM’, which has the hexadecimal value ‘57 41 50 2D 57 49 4D’. Application selection is followed by the usual command sequences for WLTS or the signature application.

## 19.7 MICROBROWSERS

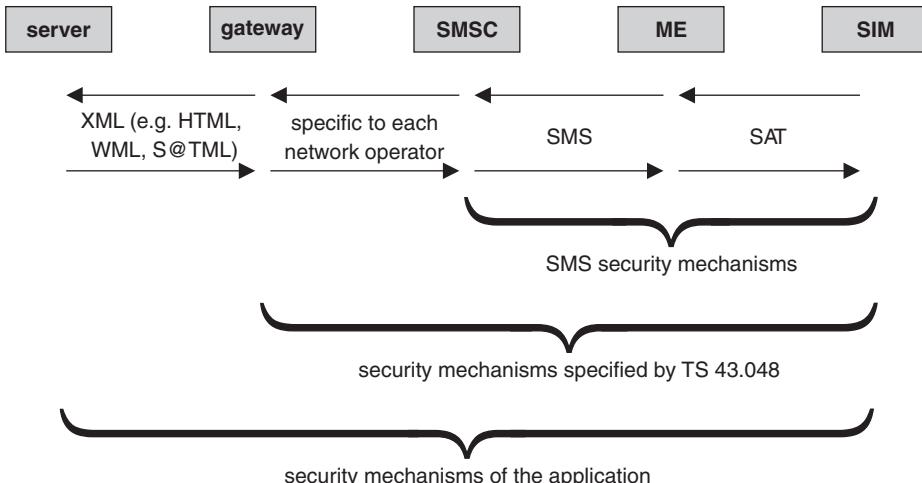
The World Wide Web undoubtedly owes a significant portion of its success to the web browsers. They made it possible to view content in the form of hypertext documents stored on remote servers, navigate among these documents, and run program code embedded in these hypertext documents without undesirable side effects in the client computer, all without requiring the installation of computer-specific software.

Browsers with simple structures that need only a small amount of memory and processing power are often called microweb browsers. Generally speaking, these browsers cannot run program code embedded in hypertext documents, and they are adapted to a specific target system with regard to memory usage and the necessary processing power. A microweb browser is an extension of the associated smart card operating system.

SIM-based microweb browsers were first announced for the GSM environment in 1998 by Across Wireless, which is now called Smarttrust [Smarttrust]. These microweb browsers can interpret a WML dialect optimized for SIMs and USIMs, and they support the addition of supplementary functions, such as generating digital signatures using the RSA algorithm, by means of downloadable plug-ins. The program code size of a typical microweb browser lies in the range of 30 to 50 KB, although plug-ins for various supplementary functions can easily add another 20 KB of program code. The size of the working buffer in RAM is typically 1 to 2 KB. Table 19.16 shows an example of a simple WML application.

**Table 19.16** An example of a simple WML application. The WML document, which is located on a server, is converted into a 20-byte WML bytecode that is packaged in a text message and sent to the microbrowser in the SIM or USIM. The microbrowser interprets the WML bytecode and uses the SIM Application Toolkit DISPLAY TEXT command to show the message ‘Hello World’ on the display of the mobile equipment

<?xml version="1.0" encoding="ISO-8859-1"?>	WML version and character encoding specification
<wml>	Start of the WML document
<card>	Start of a new card
Hello World	Message to be displayed
</card>	End of the card
</wml>	End of the WML document
'00 12 02 0C 48 65 6C 6C 6F 20' '57 6F 72 6C 64 20 06 00 06 00'	WML bytecode equivalent of the WML document



**Figure 19.26** The basic data transmission chain and hierarchically structured security mechanisms used with a microbrowser application with SMS as the bearer service

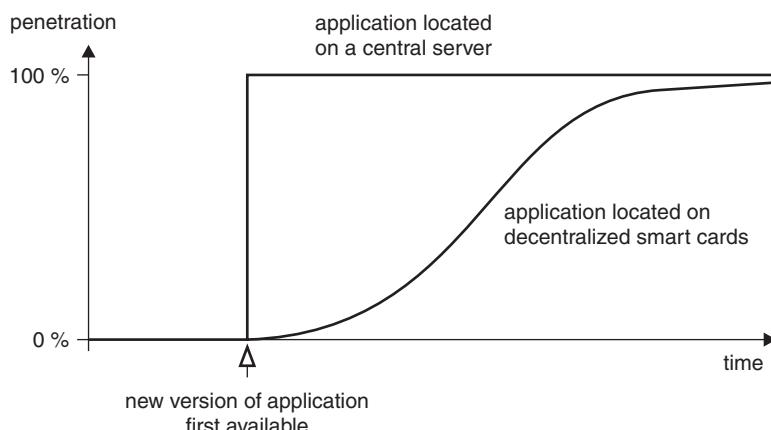
In 1999, Gemplus, Giesecke & Devrient, ORGA and Schlumberger founded the SIM Alliance [SIM Alliance] consortium with the objective of enabling mobile telephones without WAP capability to also use the services developed for WAP. At that time, this was a very attractive option because the market penetration of WAP-enabled mobile telephones was low. To provide the capability, the SIM or USIM must have a microbrowser called the S@T browser that supports the SIM Alliance specification, while the mobile telephone only needs to support GSM Phase 2+. This gives the SIM or USIM enough control over the mobile equipment via the SIM Application Toolkit to allow part of the WML content and certain WML functions to be reproduced.

Security is an important consideration with microbrowsers, and it involves various parts of the overall system as depicted in Figure 19.26. Compared with microbrowsers that run directly in the mobile equipment, microbrowsers that run in a SIM or USIM have the major advantage

of operating in a secure environment. It is entirely conceivable that many of the functions of a browser that runs in the mobile equipment could be manipulated in a manner that would make both the processor and the memory accessible for manipulation. By contrast, a microwebbrowser that runs in a SIM or USIM is fully integrated into the smart card and can take advantage of all protective mechanisms available to the smart card. This makes it possible to establish a fully secure end-to-end link at the application level between an application on the application server and the SIM or USIM. This is not possible with a browser that runs in the mobile equipment.

The WIB browser and the S@T browser are both proprietary browsers that do not originate from a standardization body. They are also mutually incompatible. ETSI published the first version of the specification for the USIM Application Toolkit Interpreter (USAT Interpreter) at the end of 2000. However, the USAT Interpreter was unable to establish a market presence with network operators in competition with the WIB and S@T browsers because it did not offer any significant new functionality and the other two browsers had already achieved a high level of market penetration.

It is generally necessary to have an online system available, such as GSM or UMTS, in order to use a microwebbrowser in a smart card. The contents to be displayed or executed are located on an application server belonging to the content provider. This server is linked to a gateway server by the usual Internet protocols and cryptographic mechanisms for secure communication, such as SSL and TSL, which ensure suitable communication security for accessing the application server. In case of communication with the SIM or USIM via the SMS channel, the gateway server is linked to a short message service center (SMSC). The data to be sent to the SIM or USIM is reformatted into suitable TS 43.048-data packets, and the necessary cryptographic features for secure communication are added to the data. The data packets are then sent transparently to the SIM or USIM via the GSM system in the form of text messages. The SIM or USIM recognizes that these messages contain TS 43.048 data packets, and it decodes the data accordingly and reconstructs the original message from several text messages as necessary. If no errors occur in this process, the result is sent to the microwebbrowser in the



**Figure 19.27** Penetration versus time of an application update with centralized storage on a server versus decentralized storage in SIMs or USIMs. Depending on the mobile telecommunication system and the application concerned, several weeks may be required to achieve 90 % penetration with decentralized storage

SIM or USIM, which interprets the message and executes the corresponding SIM Application Toolkit commands. After this, a response may optionally be generated and returned to the application server using the reverse sequence.

In some cases the microbrowser does not receive its messages in the form of online content from an application server connected via the network, but instead in the form of XML bytecode stored in a file in the SIM or USIM. This offline content provides an elegant means to store frequently used applications locally for rapid access, while at the same time reducing the network load. However, a drawback of this approach is that these locally stored applications must be explicitly updated by the background system. As illustrated in Figure 19.27, the main advantage of applications located on a central server is that it is not necessary to update applications on individual cards.

# 20

## Smart Cards in Health Care Systems

The financial and administrative costs of good health care systems are enormous, which leads to all sorts of efforts to reduce them. The vast majority of the data acquired in health care systems is highly sensitive personal data that is of considerable interest to many parties, and for understandable reasons those responsible for this data refuse to grant them access to this data.

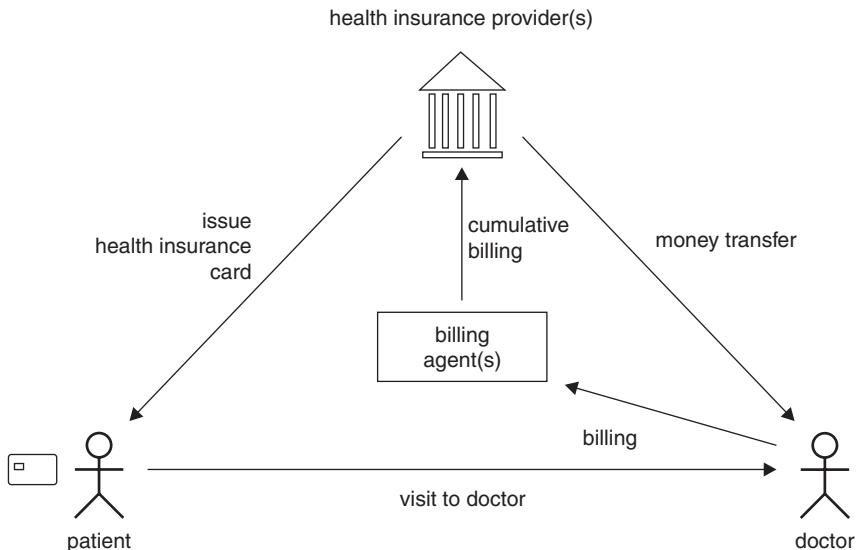
These are all compelling arguments against any system in which all health care data is stored centrally on a server where it can be retrieved and used by many processing entities as needed. Given this situation, smart cards are an ideal medium for managing diverse authorizations and assuming at least part of the burden of storing confidential data. Many countries have already introduced smart cards in the health care sector, such as the Taiwan Health Care Card (THC) and the e-card health care card in Austria, which can serve as good examples of such cards while at the same time illustrating the relatively broad range of options in this area.

In this chapter, we describe health care cards based on the two different types of smart cards used in the German health care system. This has the advantage that we can directly compare two different generations of cards: a simple memory card and a processor card with a large range of applications. Furthermore, the German health care system is highly complex, substantial efforts are made to protect personal data, and the specifications are public – all of which are factors that influenced our decision to consider only one country here. There is probably no other country in the world with a more complex spectrum of smart cards in the health care sector, which means that here readers can easily recognize the upper limits of what is possible.

### 20.1 HEALTH INSURANCE CARDS IN GERMANY

In Germany, every member of a public health insurance plan was issued a health insurance card (*Krankenversichertenkarte*) (KVK) by the end of 1994. In 1996, the private health insurance plans started issuing their own smart cards, which are compatible with the public KVK cards. The basic architecture of the German health insurance card system is shown in Figure 20.1.

Originally, it was only planned to introduce a magnetic-stripe card, but in consideration of possible future developments it was decided to use smart cards instead. The least costly solution was to use memory cards, since at that time microcontroller cards were much more expensive. However, the system is designed such that memory cards can be replaced by ‘real’



**Figure 20.1** Basic architecture of the German health insurance card system

smart cards in evolutionary stages over the course of several years. The result is a nationwide system that can form the basis for a future health care smart card if this becomes necessary.

The health insurance card has two basic functions for the insured person. Its first function is to identify the person to the doctor who treats the person, in which role it replaces the paper health insurance card. Its second function is to act as a machine-readable data storage medium for the computer in the doctor's clinic. The terminal is usually connected to a PC in the clinic, which also controls the terminal. The card can be read using the terminal, and the billing data obtained in this manner can be further processed automatically. If the doctor manages his or her practice using traditional methods (which means without a computer), the terminal can transfer the data directly from the card to a printer and thereby to a printed form.

Three different parties can access the health insurance card. The first is the doctor's clinic, where data can only be read from the card. Here there is no intention of allowing data to be written to the card, and the terminal software prevents such access. The second party is the health insurance organization, which again can only read the data in the card. Here the insured person can read and check the personal information stored in the card. The health insurance organization also has special terminals that allow data to be written to the card. This can for example be necessary if the insured person moves to a new address.

However, many insurers simply issue a new card to the insured person instead of modifying the data in the existing card, and they request the cardholder to destroy the old one. This is significantly simpler in terms of logistics and thus less costly.

In the initial phase of the KVK project, consideration was given to storing a wide variety of patient data in the card. Some people wanted to include all possible data in the card, ranging from the blood type to allergies, so it would be a sort of emergency card. However, after all objections regarding the protection of personal data had been resolved, only the personal information listed in Table 20.1 on the facing page was left to be stored in the chip in the card.

The data stored in the card is essentially the same as the information printed on the card body, so the insured person knows exactly what data is in the card, at least to the extent that it

**Table 20.1** Data elements of German health insurance cards and their TLV coding, as defined in the technical specifications for German health insurance cards (1993)

Data element	Length	Tag
Federal state	1–3 bytes	'8A'
Surname of the insured person	2–28 bytes	'87'
Date of birth of the insured person (DDMMYYYY)	8 bytes	'88'
Expiry date of the card (MMYY)	4 bytes	'8D'
Name of the insurer	2–28 bytes	'80'
Name extensions of the insured person	1–15 bytes	'86'
Number of the insurer	7 bytes	'81'
Number of the insured person	6–12 bytes	'82'
Name of the city or town	2–22 bytes	'8C'
Postal code	4–7 bytes	'8B'
Checksum of the entire template (XOR)	1 bytes	'8E'
Legal system (east/west)	1 bytes	'90'
Status of the insured person	4 bytes	'83'
Street name and house number	2–28 bytes	'89'
Title of the insured person	2–15 bytes	'84'
Health insurance data template	70–212 bytes	'60'
Health insurance card number	5 bytes	'8F'
Given name of the insured person	1–28 bytes	'85'

is fixed and personal. The address is stored only in the memory of the card, so in principle it is not necessary to generate a new card if the insured person moves to a new address.

The requirement that the information in the card be generally known was also one of the prerequisites for the approval of the overall system. No information that is secret or not known to the insured person is allowed to be present in the card. It must also not be possible to write additional data to the card at a later date without authorization. To exclude the possibility of writing data to the card, neither doctors nor the insurers receive terminals that have this capability. Only a few administrative terminals located in the insurance organizations can write data to the cards. However, a special key authentication key is not required for this operation, so data can easily be written to the card using any suitably equipped terminal.

From a purely external perspective, a health insurance card behaves as though it contains a single transparent file (EF). Data can be freely read from this file using offset and length parameters. Certain administrative terminals can also write data to the memory, but for reasons of personal data privacy this only occurs as an exception.

When the arrangement of the data elements was being specified, it was very important to be able to make extensions or changes in the future without creating compatibility problems. Consequently, all personal data in the health insurance card is structured using the ANSI data description language and stored in the card's memory in a TLV structure. This makes it possible to add other data objects in the future or modify the coding of existing data objects. The tags to be used are defined by a specification to ensure that the data elements of all health insurance cards are structured on the same basis.

The health insurance card is not a processor card. It is a memory card, with hardware similar to what has been used for years in telephone cards. The EEPROM in the card must have a capacity of at least 256 bytes, which is equal to the volume of necessary data in the

health insurance card. If the EEPROM is exactly this large, the necessary data fits precisely in the memory and it is physically impossible to write any additional data to the card in violation of data privacy legislation.

Various synchronous data transmission protocols are used, depending on the type of chip that is used. Each terminal must therefore be able to fully execute all possible protocols. The card body can be produced using injection molding or in multilayer technology. The service life of the health insurance card is specified as six years. After this time, the insured person automatically receives a new card. This means that around 15 million new cards must be issued every year.

If an (optional) computer is available, all of the terminals present are controlled by the computer using the  $T = 1$  transmission protocol according to ISO/IEC 7816-3. There is one restriction in this regard, which is that data chaining may not be used in the protocol. Data chaining would increase the memory requirement of the terminal without enhancing the functionality of the application. Incidentally, this is a typical example of the fact that real applications often employ only the necessary parts of a standard, and it is rare for all of the functions specified by a standard to be implemented in practice.

There are only three possible commands that the terminal can execute. The first command sends a reset signal to the health insurance card, which is followed by reception or reading of the ATR. This command is always used at the start of a session to activate the health insurance card. The second command is READ BINARY with ISO coding, which can be used to read selected data or all of the data via the terminal. The third command is WRITE BINARY, also in accordance with ISO/IEC 7816-4, although this command is only available in administrative terminals. It is blocked in all other types of terminals. The health insurance cards of the private insurers have write protection implemented using PINs that are known only to the insurance organization. The cards for the public health insurance plans can be freely written if the necessary commands are known.

If the terminal is connected directly to a computer, its only real role is to convert between the  $T = 1$  protocol and the hardware-dependent synchronous protocol of the health insurance card. Nevertheless, here it can clearly be seen that a conversion to processor cards would be possible without undue effort or expense. With processor cards, the only function of the terminal would be to transparently relay the commands received from the clinic computer. The response from the card could also be returned transparently to the control computer without any processing by the terminal.

## 20.2 ELECTRONIC HEALTH CARE CARDS IN GERMANY

In 2002, the decision was taken to introduce new health care cards in Germany that would replace the health insurance cards issued throughout Germany since 1994. For this purpose, a project team was created in 2003, followed in 2005 by a company whose tasks are to issue the specifications and to install and operate the telematics equipment of the German health care system. The company is called *gematik*, which is a contraction of *Gesellschaft für Telematikanwendungen im Gesundheitswesen*.

The German health care system is complex, with a large number of players and special interest groups. The following figures can serve to give an initial impression: there are 700 million paper prescriptions per year, 80 million insured persons, 123 000 doctors, 65 000 dentists, 21 000 pharmacies, 2200 hospitals, and 260 health insurance providers. The health care cards,

together with their extensive infrastructure, are intended to be the binding link between all of these parties.

To provide a legal framework, the German social security statutes governing health insurance were amended with new provisions addressing the electronic health care card system, which define the essential aspects of the system and its functionality. This legislation stipulates that health care cards must be able to record medical prescriptions in a form suitable for use with electronic and automated equipment, and that they must serve as proof of entitlement to utilize the services of insurance providers. The data must also be available with network access. Furthermore, it is explicitly stated that the German Federal Data Protection Act is applicable. In addition, the cards must be able to support all of the supplementary functions listed below:

- medical data for emergency treatment
- findings, diagnoses, therapy recommendations, and treatment reports for interdisciplinary, case-specific cooperative treatment (electronic physician's letter)
- data for verifying the therapeutic safety of medications
- findings, diagnoses, therapeutic measures, treatment reports, and inoculation data for patient documentation (electronic patient file)
- data regarding the insured person and data made available to the insured person
- data regarding utilized services

The individual applications and functions of the health care card can be derived from the statutory provisions. The card has a set of mandatory applications, as well as supplementary voluntary applications. The mandatory applications, which are the same for all insured persons, are the personal data of the insured person, electronic prescriptions, and the European Health Insurance Card (EHIC). The voluntary applications, which can be chosen as desired by the insured person, are significantly more extensive. They include the emergency medical data, which among other things provides information about allergies, chronic illnesses, operations and medication allergies, as well as the electronic physician's letter for conveying medical data, the patient folder for storing health data related to the patient or other persons, and the electronic patient file for storing all accrued medical data.

Clearly, the combination of these extensive requirements and the sensitivity of the data to be processed leads to a highly complex overall system. In the meantime, the specifications have achieved a scope that is distinctly larger than the scope of this book, and the system is one of the most complex systems ever developed anywhere in the world. In addition to the fact that the technical requirements are only described in rudimentary form, some of the parties involved or affected are strongly opposed to the new system. Consequently, it is not especially surprising that the deadline specified by law for the operational launch of the system (1 January 2006) could not be achieved. It can reasonably be expected that several more years will pass before the new health cards are introduced throughout Germany.

### 20.2.1 Card types

Two types of cards are specified for the system: the electronic health care card (eGK) and the electronic health care professional card (HPC).

The eGK is used by the insured person, while the HPC is used as proof of access authorization and as a signature card by people who are professionally active in the health care system. The health care card application on the eGK is called HCA (health care application).

Both types of cards use a smart card operating system compatible with ISO/IEC 7816 with small extensions, which has the advantage that it allows the SECCOS<sup>1</sup> operating system, which is widely used in Germany in the Eurocheque and GeldKarte systems, to be used here as well. The transmission protocol is T = 1 with support for the extended length option, which has been used only rarely up to now. The mandatory cryptographic algorithms are triple DES with two or three different keys, AES up to 256 bits, and 2048-bit RSA.

As of now, the eGK and the HPC are defined in a set of three extensive and detailed specifications. As with all specifications for this system, they are freely available and can be downloaded at no charge from the website of the German Federal Ministry of Health [BMG]. Part 1 specifies the operating system, although its title indicates that it specifies the electrical interface. Part 2 describes the mandatory and voluntary card applications, while Part 3 describes the external form of the cards.

## 20.2.2 Applications in electronic health care cards

The eGK has several mandatory applications that must be present in every card. Under the MF there is an EF<sub>ARR</sub> for managing access rules, an EF<sub>ATR</sub> for extending the ATR, an EF<sub>DIR</sub> that lists the applications present in the card, and an EF<sub>GDO</sub> that holds the serial number of the card (ICCSN). EF<sub>StatusPIN</sub> holds the transport protection application, and EF<sub>Version</sub> holds the version number of the specification applicable to the card. There are also two files (EF<sub>CVC.CA\_eGK.CS</sub> and EF<sub>CVC.eGK.AUT</sub>) with certificates, which among other things are used for mutual authentication of the eGK and the HPC. Figure 20.2 on the next page shows the associated file tree. A variety of additional files are possible according to the eGK specification, but they depend on the choice of operating system and are thus not mandatory in the file system.

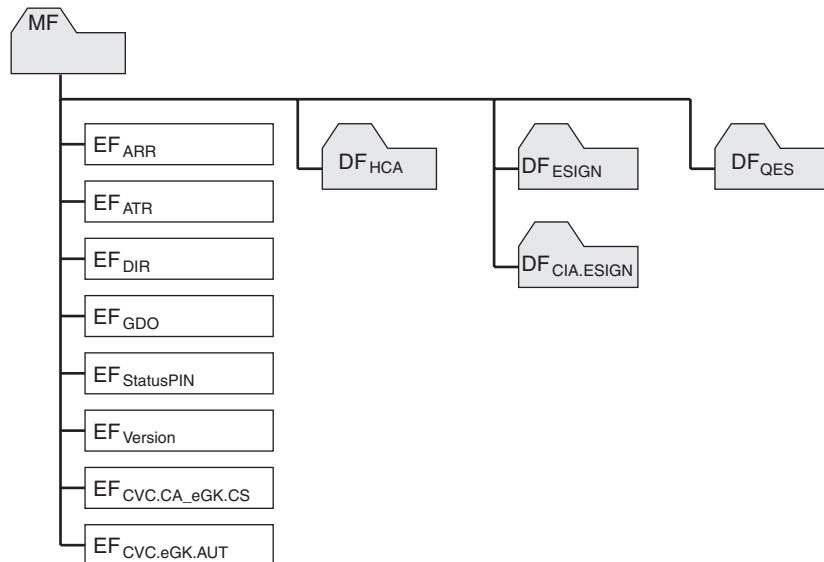
Along with the EFs, there are several DFs that hold the applications in the eGK. The most important application is the health care application (HCA), which holds the health care card data. There are also two DFs that hold the data of the ESIGN application, which is compliant with CWA 14890 and intended to be used for authentication, data encryption, and data decryption. In addition, there is an optional DF<sub>QES</sub> that holds the data for qualified electronic signatures. If necessary, this DF can be downloaded online by a card management system.

The HCA application in DF<sub>HCA</sub> is composed of the files shown in Figure 20.3 on the facing page. EF<sub>ARR</sub> holds the access rules for files in the directory. EF<sub>DM</sub> holds a message that can be displayed on a terminal after authentication so the user can confirm that the communication link is secure.<sup>2</sup> EF<sub>PD</sub> and EF<sub>VD</sub> hold the personal master data and the insured person data, which are important for administrative purposes. EF<sub>GVD</sub> holds the sensitive data of the insured person. This file can only be read after authentication. EF<sub>StatusVD</sub> has a transparent structure and holds status information regarding the EF<sub>PD</sub>, EF<sub>VD</sub>, and EF<sub>GVD</sub> files.

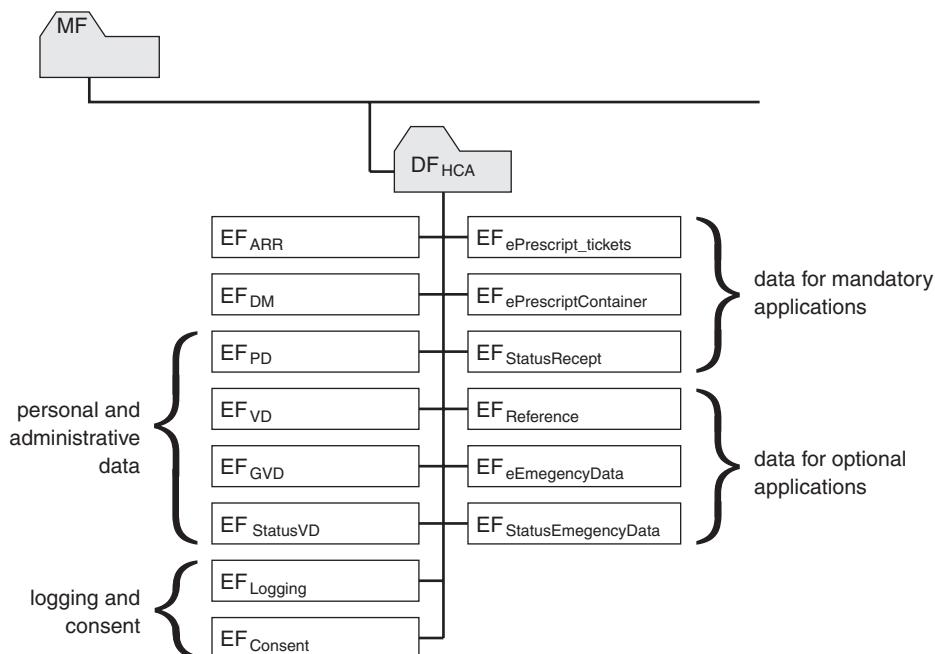
EF<sub>Logging</sub> is a log file with linear fixed structure that holds data on the last fifty accesses to the eGK. Data on the use of the voluntary applications is held in EF<sub>Einwilligung</sub>, and data regarding the electronic prescriptions currently present in the card is held in EF<sub>eRezept\_Tickets</sub>, which has a linear fixed structure. This includes EF<sub>eRezept\_Container</sub>, which can hold electronic prescriptions, and EF<sub>StatusRezept</sub>, in which associated status information can be stored.

<sup>1</sup> See also Section 18.6, ‘The Eurocheque System in Germany’, on page 783

<sup>2</sup> The underlying operating principle is described in Section 7.8.5, ‘Verifying that a terminal is genuine’, on page 192



**Figure 20.2** Basic file tree of the German electronic health care card (eGK) with the health care application (HCA), ESIGN, and qualified electronic signature (QES)



**Figure 20.3** File tree of the health care card application (HCA) in the German health care card

EF<sub>Verweis</sub> has a linear fixed structure with ten records and holds the IDs of the voluntary applications present in the card. One of the voluntary applications is the emergency medical data, which is located in EF<sub>eNotfalldaten</sub>. The status of this file is stored in EF<sub>StatusNotfalldaten</sub>.

### 20.2.3 Electronic prescriptions

The electronic prescription application is one of the mandatory applications in the eGK. It is intended to replace the paper prescriptions commonly used at present and to eliminate the current media discontinuity with prescriptions. This discontinuity results from the fact that doctors presently write prescriptions on paper and patients use these paper prescriptions to obtain medication in pharmacies, but in the end the prescriptions must be acquired in electronic form for settlement with the health insurance providers, which entails significant effort and expense.

The electronic health care card eliminates this media discontinuity by enabling the processing of prescriptions in electronic form along the entire chain. In the future, the process will be as follows. The physician uses her computer to generate a prescription and signs it digitally with her HPC card. The prescription is then stored on the patient's health care card or sent to an electronic prescription server via a secure connection. The patient then takes his health care card to a pharmacy, where the prescription is either read from the card or downloaded from an electronic prescription server. In order to access the prescription, the pharmacist must first use her HPC card to obtain suitable access permission. If everything goes well, the patient receives his prescription and the pharmacist deletes the prescription from the location where it was stored.

This example illustrates the process in the simplest case, which is doubtless the usual case in practice. However, there are a large number of special cases that must be taken into account, and they considerably increase the complexity of the entire system. Some examples include having a prescription filled online by an Internet pharmacy, having only certain items on the prescription filled, having different items on the prescription filled by different pharmacies, and changing a medication to another medication with the same active ingredient. The processes must be able to handle all these cases, since they commonly occur in practice. Naturally, the basic prerequisite for all these processes is a network and server infrastructure with full national coverage, high availability, and sufficient bandwidth and processing power.

### 20.2.4 Summary and prospects

In all likelihood, nationwide implementation of the German electronic health card system will take a while yet. This is in part due to the highly complex technical requirements, but the primary cause is a lack of consensus on the health care card system among the parties involved and affected.

From a technical perspective, it is a very interesting project that in future stages of development is intended to include applications such as transfers, admissions, physician reports, physician letters, medication and treatment histories, and electronic patient files. However, the main challenge is to convince the general population that their highly sensitive health data can only be accessed by parties that need it for medical purposes and cannot end up in the medical systems or databases of outsiders, even if they can demonstrate a well-founded interest in this data.

# 21

## Smart Cards in Transportation Systems

The use of smart cards in transportation systems is primarily characterized by many localized solutions, each with its own requirements and specifications. General-purpose, interoperable applications have not yet appeared in this sector. However, this application area with its stringent requirement for fast transactions in local public transport services has become the main driver for contactless technology.

### 21.1 ELECTRONIC TICKETS

Along with ski areas, the local public transport sector was one of the first application areas where contactless smart cards achieved a significant market presence. The main reasons for this are the distinctly shorter processing time and simplified logistics compared with paper tickets and tickets based on magnetic-stripe cards.

Another factor that encourages the use of contactless smart cards is that local public transport systems are usually organized at the regional level, so the necessary coordination effort is significantly less than with interregional projects having a large number of participants. Consequently, the largest applications in this sector are found with the local public transport companies in large cities. Some typical examples of this sort of smart card application are the Octopus card in Hong Kong [Octopus], the Oyster card in London [Oyster], the EZ-link card in Singapore [EZ-link], and the Super Urban Intelligent Card (Suica) in Tokyo [Suica].

The smart card application with the largest penetration in the population is the Octopus card. Its functionality now extends far beyond local public transport, and among other things it can be used as an electronic purse in many businesses. The Octopus card was launched in 1997 as the successor to a magnetic-stripe card called Common Stored Value Ticket. The number of cards issued to date is 14 million, and 95% of Hong Kong's 7 million inhabitants have an Octopus card. More than 10 million transactions are booked each day, with a total annual value of €2.8 billion. Approximately one-quarter of the transactions are not related to the transport services of the transportation association, which means they are payment

transactions for goods and services. There are presently around 50 000 businesses that accept the Octopus card in the Hong Kong district.

The system described below is based on the structure of the Octopus card system. The Octopus specifications are not public, so there may be some minor differences between the system described here and the actual Octopus system. However, the basic aspects of the system described here are independent of any detailed specification, so it provides a good example of the basic principles of a contactless smart card system for local public transport.

### 21.1.1 System architecture

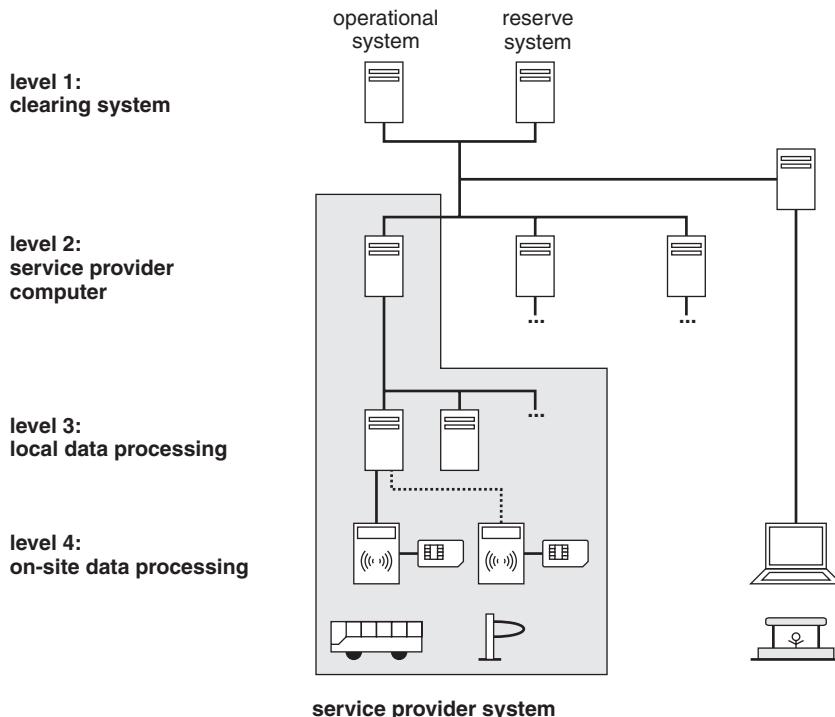
The Octopus system is based on an architecture with a central clearing house, to which the components of the various service providers are connected via several levels. The architecture also takes into account components that have only intermittent online contact with the background system. An example of such components is access terminals in buses, which must operate offline at some stops or even over entire routes. This offline capability of certain front-end components considerably enhances the robustness of the system, with the result that it can continue to operate, even if the connection to the background system is lost temporarily. Uninterrupted operation is indispensable for a local public transport system in a large city.

To enable the terminals to achieve the required offline capability, they must be able to verify the authenticity of the smart cards of travelers. This is done by mutual authentication of the terminal and the smart card, based on a shared secret consisting of a secret key. The terminal has a security module called the secure application module (SAM) that holds the secret key. In accordance with common practice in most payment systems for many years, smart cards in ID-000 format<sup>1</sup> are used for the SAMs. The terminals typically run the Linux operating system.

Figure 21.1 on the facing page shows the basic components of the system architecture, including the four levels of the overall system. The clearing house system is the central entity that receives all of the system data. It is implemented redundantly so that the backup system can take over the functions of the main system without any delay if the main system fails. The clearing house system, which performs a wide variety of functions, can be split into two subsystems. The clearing operator subsystem is responsible for system operation, settling accounts with service providers, and managing the components of the system (such as the terminals). The issuer subsystem concentrates on the smart cards used in the system and is responsible for key generation and key management, blacklist management, and in the case of personal smart cards, managing automatic reloading when the card balance drops below a certain threshold value.

The level below the clearing house system contains the central computers of the service providers. These computers also act as concentrators for the local data processors at the next lower level which from time to time establish contact with the terminals in the buses and other vehicles, some of which operate offline part of the time. The access gates may have permanent connections to the higher levels of the system, or they may connect to the higher levels only from time to time by a radio link or other type of connection. This depends on the configuration of the infrastructure at each gate site. An online connection is preferable, but for economic reasons this is not possible at every gate.

<sup>1</sup> See also Section 3.1, ‘Card Formats’, on page 29



**Figure 21.1** Basic architecture of a distributed system with contactless smart cards for local public transport, using the Octopus system as an example. The solid lines represent permanent connections, while the dotted lines represent temporary connections

By contrast, the loading terminals for the smart cards are connected online to the clearing house system via the intermediate levels. A similar arrangement exists with all businesses that accept the Octopus card as a means of payment. This makes it possible to block smart cards on the blacklist in real time.

## 21.1.2 Octopus card

The Octopus card has the standard ID-1 format.<sup>2</sup> As an alternative, the same functionality is integrated in a variety of wristwatches and a case for certain models of Nokia mobile telephones.

The data transmission channel operates at 13.56 MHz with a data rate of 212 kbit/s. The implementation of contactless communication is not compliant with ISO/IEC 14443, but instead compliant with the FeliCa protocol [FeliCa] developed by Sony. The smart card is a processor card<sup>3</sup> with a Sony microcontroller that has an 8-bit CPU, 4 KB of ROM, 4 KB of EEPROM, 512 bytes of RAM, a DES accelerator, and a pseudorandom number generator (PRNG).

<sup>2</sup> See also Section 3.1, ‘Card Formats’, on page 29

<sup>3</sup> See also Chapter 5, ‘Smart Card Microcontrollers’, on page 73

**Table 21.1** Basic data objects of an electronic ticket and their access conditions. For simplicity, additional data objects for extended-term tickets and special functions are not shown here

Data	Electronic ticket		
Access conditions			
READ:	mutual authentication of the smart card and the terminal		
WRITE:	mutual authentication of the smart card and the terminal		
Data object	Length	Format	Description
UID	8 bytes	...	Card serial number (unique identifier)
Balance	4 bytes	decimal number	Balance of the electronic purse
Entry	2 bytes	decimal number	Entry point
Exit	2 bytes	decimal number	Exit point
Status	1 bit	binary value	Trip status (started or completed)
K <sub>PICC</sub>	112 bits	...	Smart card key for authentication (proximity integrated chip card key)
K <sub>MAC</sub>	112 bits	...	Smart card key for authentic-mode data transmission (message authentication code key)

The operating system in mask-programmed ROM, which is specifically designed for use in electronic ticket applications, is evaluated at EAL 4<sup>4</sup> in accordance with the Common Criteria.

As with most contactless smart card systems, data transmission with the FeliCa protocol is cryptographically protected against interception and manipulation. Session keys generated at the start of each session after successful mutual authentication are used to defend against replay attacks. This protection is sufficient to secure not only the basic public transport application, but also the use of the smart card as a means of payment.

Access to the various data objects in the smart card (see Table 21.1) is governed by corresponding keys, which are used for mutual authentication in order to obtain access permission. With this arrangement, it is also possible to run different applications on a single smart card because access to the individual memory areas can be protected by different keys.

Particularly in contactless applications, short transaction times are very important for customer acceptance. The transaction time with the FeliCa system is around 100 ms, which is short enough to allow a complete transaction to be carried out when the card is swept past the terminal with a normal hand motion. If the transaction time were too long, the user would have to explicitly hold the card next to the terminal. This would result in congestion at the access gates, especially during the morning and evening rush hours, and would certainly lead to acceptance problems with travelers.

Smart cards that operate according to the FeliCa system support three types of data objects. The first type is called ‘random’ and contains block-structured data. These objects can be read or written in their entirety and thus correspond to the transparent file structure in smart cards compliant with ISO/IEC 7816-4.<sup>5</sup> The second type of data object is called ‘cyclic’ and has the

<sup>4</sup> See also Section 15.5.1, ‘Common Criteria’, on page 661

<sup>5</sup> See also Section 12.7.1, ‘Transparent file structure’, on page 432

same characteristics as the cyclic structure in an ISO file system. It is used to hold log data, such as the last ten payment transactions performed with the electronic purse. The third type of data object is called ‘purse’, and it holds the current balance of the electronic purse. All three types of data objects can be read, and if the corresponding access permission is obtained, they can also be written. The purse balance includes a debiting function.

FeliCa uses methods for contactless data transmission and data management that do not correspond to current ISO/IEC standards. This results from the fact that they were developed relatively early and the standards did not cover these topics at that time. Changing to smart cards that comply with the current international standards would entail considerable expense with a system of this magnitude, and in this case it would not provide any perceptible advantage for travelers.

Atomic operations<sup>6</sup> must be used for write access to nonvolatile memory in all contactless smart cards. This is necessary because it is possible for the user to move the card past the terminal too quickly. If this happens, the smart card will leave the working range of the terminal while the write operation to nonvolatile memory is taking place, with the result that the data cannot be written correctly. This could have the undesirable result for the traveler that the purse balance is set to zero instead of simply being debited by the fare for the trip. This is specifically prevented by using an atomic write operation so that the write operation is performed either completely or not at all. The terminal generates a visible or audible signal to indicate to the traveler that the transaction has been performed correctly and completely.

The Octopus system is designed in an especially favorable manner in terms of personal privacy. The smart cards used in the system are not associated with an individual person, so they can be used anonymously. This is a clear advantage relative to systems such as the Oyster system in London, which operates exclusively with personal cards. The fact that this can arouse the interest of police forces and other security agencies can be seen from the fact that in the period from August 2004 to March 2006 alone, the database of the Oyster system was queried 409 times.<sup>7</sup>

A drawback of anonymous cards is that the cardholder must use a loading terminal to reload the card. Reloading can be made more convenient for travelers by using personal cards. With such cards, it is only necessary to trigger an automatic add value service (AAVS), also known as ‘auto top-up’, when the purse balance drops below a certain threshold. However, this function requires a reference to a bank account or a credit card, which means that anonymity is not possible. The Octopus system supports both options – anonymous and personal – although the user must sacrifice anonymity for the convenience of automatic loading.

### 21.1.3 Trip registration

A wide variety of methods are used for trip registration with conventional tickets. In the simplest case, the traveler buys a ticket at the counter or from a ticket dispenser, goes to an entry point, travels over the desired route, and leaves the vehicle when it arrives at the destination point. Depending on the type of vehicle, the tickets may be checked by a conductor or access may be controlled by a turnstile or similar mechanism that checks machine-readable tickets. In certain rare cases, travelers must also have their tickets checked by a machine

<sup>6</sup> See also Section 13.10, ‘Atomic Operations’, on page 480

<sup>7</sup> source: [en.wikipedia.org/wiki/Oyster\\_card](http://en.wikipedia.org/wiki/Oyster_card), consulted on 28 April 2008

when they leave the vehicle or the station. In addition to tickets for specific routes, there are numerous other options, depending on the specific transportation company, such as tickets valid for a certain period ranging from a few hours to a year. The main reason for this nearly indescribable variety of fare schemes arises from the regional organization of transportation companies. With regard to checking tickets at the end of the trip, the degree of trust in the travelers and the fraud rate are the most significant considerations.

In systems that use electronic tickets, attempts are often made to reproduce the fare schemes used with conventional tickets in order to facilitate the use of both systems in parallel and avoid further increasing the complexity of the fare scheme in the eyes of the travelers. Nevertheless, in a relatively large number of cases the check-in/check-out (CICO) model has now proven to be a good solution. With this model, the traveler must present the electronic ticket when entering the system and again at the end of the trip when leaving the system. With this information, the trip can be determined and the corresponding fare can be calculated and debited from the card balance. An alternative designation for this method is ‘touch-in/touch-out’.

A similar approach is the be-in/be-out (BIBO) model, in which the electronic ticket is detected automatically when the traveler enters and leaves the vehicle and does not have to be presented explicitly to a terminal. Although this is distinctly more convenient for travelers, it requires contactless cards that can be detected and read over larger distances than the typical range (10 cm) of the FeliCa system and systems compliant with ISO/IEC 14443.

Be-in/be-out systems also eliminate one of the problems with check-in/check-out systems, which is that travelers may forget to check out when leaving the system at their destination. Although turnstiles can be placed at the exit to compel travelers to check out, this results in disproportionately high costs, especially in the outer reaches of extensive local public transport systems, due to the necessary infrastructure investments.

This problem is often solved in practice by taking the following approach. When the traveler checks in, the terminal records the ID of the starting point in a data field in the smart card. If the traveler forgets to check out after reaching the destination, the maximum possible fare is debited from the card balance the next time the traveler checks in to the local public transport system. The idea behind this is that the high fare will encourage the traveler to remember to check out at the end of the next trip.

With some systems, people have learned from experience that the balance of electronic ticket cards can go negative by a certain relatively small amount. If this is less than the amount of the deposit charged for the card, the system operator cannot suffer any damage as a result. There is an advantage to this form of goodwill with respect to travelers, which arises from the fact that it is easily possible for the fare to be higher than the current card balance when the traveler checks in to a CICO system. In addition, the fare cannot be determined reliably in advance with a CICO system because the system does not know where the traveler will go when the traveler checks in. Tolerating a negative balance is an elegant way to avoid situations in which travelers are prevented from passing through the exit gate after arriving at their destination.

#### 21.1.4 Typical transactions

The most important transactions with electronic tickets are the identification and authentication of the card and the check-in and check-out transactions. The terminal performs identification and authentication to check whether the ticket is genuine, and it usually generates the session key for subsequent secure communication at the same time. This forms the basis for all

Terminal (PCD)	Smart card (PICC)
<i>Identify the smart card</i>	
READ BLOCK [tag for UID]	————→ ←———— <i>Response [UID]</i>
<i>Authenticate the terminal and the smart card</i>	
$K_{PCD} = f(K_{MPCD}, \text{UID})$ READ BLOCK [tag for RND <sub>PICC</sub> ]	————→ ←———— <i>Generate RND<sub>PICC</sub></i> <i>Response [RND<sub>PICC</sub>]</i>
Generate RND <sub>PCD</sub> $X_{PCD} = E(RND_{PCD} \parallel RND_{PICC}, K_{PCD})$ AUTHENTICATE [ $X_{PCD}$ ]	————→ ————→ <i>RND'_{PCD} \parallel RND'_{PICC} = D(X_{PCD}, K_{PICC})</i> IF ( $RND_{PICC} = RND'_{PICC}$ ) THEN terminal authenticated ELSE terminal not authenticated $X_{PICK} = E(RND_{PICK} \parallel RND'_{PCD}, K_{PICK})$ <i>Response [X<sub>PICK</sub>]</i>
RND'_{PICK} \parallel RND'_{PCD} = D(X_{PICK}, K_{PCD}) IF $RND_{PCD} = RND'_{PCD}$ THEN smart card authenticated SE smart card not authenticated	————←————
<i>Generate a key for authentic-mode communication</i>	
$K_{MAC} = f(K_{MMAC}, \text{UID})$ $KS_{MAC} = f(RND_{PICK}, K_{MAC})$	$KS_{MAC} = f(RND_{PICK}, K_{MAC})$

**Sequence Diagram 21.1** Transaction process for mutual authentication of the smart card and the terminal. The random numbers generated in this process are used together with a card-specific key to generate session keys for data transmission in authentic mode. The following data is used in this process: the UID is the unique identifier of the smart card and  $K_{MPCD}$  is the master key used to derive the card-specific keys  $K_{PCD}$  (terminal) and  $K_{PICK}$  (smart card) for authentication, while  $RND_{PCD}$  and  $RND_{PICK}$  are the random numbers generated by the terminal and the smart card, respectively

following transactions. If authentication is successful, it is typically followed by a check-in or check-out transaction.

As it is very important in local public transport systems to check the tickets of a large number of people in a short time, the transaction duration is an essential acceptance criterion. In practice, an interval of up to 0.3 s has been found to be acceptable to travelers. This transaction time is short enough to allow a terminal to carry out a complete transaction while the traveler waves the smart card past the terminal. As the maximum allowable distance between the terminal and the smart card is 10 cm, it is not necessary to make direct contact between the card and the terminal in order to perform the transaction.

#### 21.1.4.1 Identification and authentication

Sequence Diagram 21.1 shows the process of identification followed by authentication and generation of a session key. The first step is to read the unique identifier (UID) of the electronic ticket, which is the card serial number. In the next step, the UID is used to derive a

**Table 21.2** The basic steps of a check-in transaction with an electronic ticket as seen by the terminal

Step	Activity
1	Read the UID from the electronic ticket
2	Authenticate the electronic ticket and generate the session key for subsequent authentic-mode communication
3	Read the status of the last trip from the electronic ticket. If the status is 'Started', debit the maximum fare from the electronic ticket and set the status to 'Completed'
4	Write the starting point ID to the electronic ticket and set the status to 'Started'

card-specific key for the subsequent mutual authentication using the challenge–response method. The cryptographic implementation of this step in this case is relatively sophisticated due to the use of a card-specific key. This could certainly be dispensed with if a simpler method were used, but it would have the disadvantage that the entire system would be compromised if the authentication key of a single smart card became known.

After successful mutual authentication of the smart card and the terminal, each of the parties generates a session key for authentic-mode communication, which is used to secure the subsequent data transmission.

#### **21.1.4.2 Check-in transaction**

The check-in transaction is a direct extension of the identification and authentication transaction shown in Sequence Diagram 21.1 on the preceding page. The essential steps on the terminal side are listed in Table 21.2. They also include the business process for billing trips that were not completed correctly, which means trips where the traveler forgot to check out. In such cases, the usual practice is to charge the maximum fare of the transportation system. This is recognized by means of a data object in the smart card that indicates to the terminal whether a trip was completed correctly.

#### **21.1.4.3 Check-out transaction**

Like the check-in transaction, the check-out transaction utilizes the identification and authentication process shown in Sequence Diagram 21.1 on the preceding page. The steps that take place in the terminal are listed in Table 21.3 on the next page.

### **21.1.5 Value-added services**

After electronic tickets have achieved widespread use in a region, they become very attractive as a medium for value-added services. This can be seen quite clearly with the Octopus card. It is now accepted as an electronic purse by many shops and service providers in Hong Kong. Some examples include grocery shops, vending machines, photocopiers, public telephones, and parking meters.

With this arrangement, the electronic ticket can also be used for many low-value purchases in the region concerned. As the fare payment function must have sufficient protection against

**Table 21.3** The basic steps of a check-in transaction with an electronic ticket as seen by the terminal

Step	Activity
1	Read the UID from the electronic ticket
2	Authenticate the electronic ticket and generate the session key for subsequent authentic-mode communication
3	Read the starting point ID from the electronic ticket
4	Calculate the fare for the trip
5	Debit the trip fare from the balance of the electronic ticket
6	Set the status of the trip the electronic ticket to 'Completed'
7	Write the endpoint ID of the trip to the electronic ticket

attacks, there is no objection in this regard to enlarging the scope of use. Normal credit cards can be used for larger purchase amounts.

Another value-added service that is available with the Octopus card is access control. The unique card ID (UID) is used as the reference data for this purpose. In this application, the terminal reads the UID from the smart card and compares it with a list of UIDs for which access is allowed. If the UID of the card is in the list, the user is granted access. The advantage of this simple method is that it does not require the terminal to be equipped with a security module to hold a secret authentication key. From a security perspective, this method is relatively open to attack because it does not employ any authentication. However, the level of security is adequate for some scenarios.

### 21.1.6 Evolution of electronic tickets

The size of the investment necessary for converting from paper tickets to electronic tickets acts as a deterrent for some operators. Another consideration is that many transportation companies do not generate enough earnings to finance the conversion to contactless smart cards. In addition, the savings resulting from the use of electronic tickets are not large enough to have a positive impact on earnings in the short term. As a result, the market penetration of electronic tickets is proceeding slowly, unlike the avalanche-like growth in the use of contactless ski passes.

However, existing localized systems are becoming increasingly attractive due to the potential for value-added services. These systems form the core for further expansion, especially in regions with high population density. In addition, standardization of contactless smart card technology and the consequent mutual compatibility of the various systems has now advanced to a point that, with a cooperative attitude and relatively small adaptations, it would certainly be possible to use existing electronic tickets for transport in systems operated by different entities. This would be a real boon for travelers, who for example would be able to use a single electronic ticket in all public transport systems of a particular country.

A difficulty here with regard to personal privacy is that in some countries electronic tickets will be used to monitor travelers, which would not be possible with conventional paper tickets. Here it is primarily up to citizens, who hopefully are willing to stand up for their rights, to determine whether this will be regarded as socially acceptable or this sort of monitoring will be curtailed by political means.

## 21.2 SKI PASSES

The first ski lift was built in Eisenach (Germany), located in the Black Forest highlands, in 1908 by a farmer and hotel owner named Robert Winterhalder. The lift, with a length of 280 m, was put into operation the same year. Since then the size and complexity of ski lifts has increased enormously, with the most striking modern-day examples being found in ski complexes such as Ski Amadé, Dolomiti Superski, and Salzburg Superski. Ski Amadé [Skiamade] is a conglomerate of five ski areas with 270 lifts and 860 kilometers of runs. Dolomiti Superski [Dolomiti] combines five ski areas with 450 lifts and 1200 kilometers of runs. Ski complexes such as Dolomiti Superski sell around 9 million ski passes each year, which are used for approximately 130 million trips. Salzburg Superski is an association of 23 ski areas with 580 lifts and a total of 2200 kilometers of runs.

It is understandable that manually punched cards are the simplest and least expensive solution with some lifts. However, large ski areas require automated systems for economical operation. This has led to an evolution in ski passes, starting with printed cards manually stamped or punched at the base stations, progressing to paper cards with printed bar codes, and then to contactless smart cards. Contactless smart cards have now become established on a broad front, although many older systems are still in use. The characteristics of contactless smart cards give them especially strong advantages in this application. With a contactless smart card, it is no longer necessary for skiers wearing gloves to awkwardly fish their ski passes out of a jacket pocket, and operators benefit from lower costs and greater security with contactless smart cards. Consequently, it is not surprising that lift systems started using contact smart cards as early as 1987.

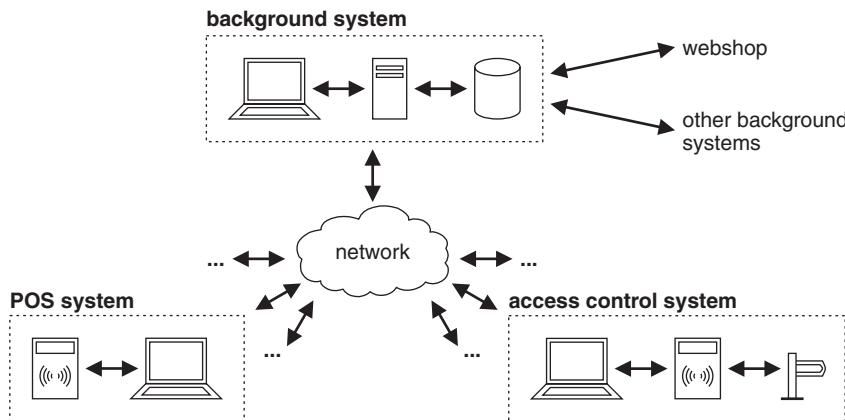
An enormous variety of ski passes have become established over the years, and they are also represented in electronic form. For instance, there are point cards good for a certain number of lift trips, day cards good for one day, multi-day cards, and even season cards. There are also ski passes with one of the previously mentioned properties that are valid in only one particular ski area or that can be used in several related ski areas, which are also called ski pools. All types of cards are also available in versions for adults, youths, children, and seniors, with various fee structures.

There are also many different ways to buy ski passes. The simplest option is to buy a pass directly at the lift. A logically more complex option is to buy the ski pass in a hotel in the ski area or from a webshop. With both of these options, there is a distinction between disposable cards and reloadable cards, with strong regional differences in this regard. Cards are usually provided against a deposit in Europe, while in the USA they are typically sold.

There is no international standard or industry standard for access control systems of this sort. However, some basic system architecture principles have become established in practice, and nearly all systems comply with these principles. The following description is thus based on the principles generally used for systems of this sort, rather than any actual system. It is thus entirely possible for actual systems to differ occasionally from what is described here, although their architecture is always very similar to that described here.

### 21.2.1 System architecture

The architecture of a ski pass system is centralized. There is a background system, and all POS and access systems are connected to it. Figure 21.2 on the facing page shows the basic features



**Figure 21.2** Basic architecture of a ski pass system in a single ski area. The background system is responsible for clearing processes, system monitoring, and communication with other systems. The POS system is used to display information or modify ski passes, and the access system controls access to the lifts

of the system architecture. In addition to the directly connected operational components, the background system provides several interfaces to other systems. If the ski area is part of a group of combined ski areas, these other systems may be the background systems of the other ski areas. However, there may also be an interface to a webshop, which is a feature offered by more and more systems. This enables skiers to buy their ski passes in advance via the Internet. In this case, the ski pass is activated when the user passes the first access point of the system, which is called the initial booking terminal. All of the related data is also loaded directly into the ski pass at this point.

The system and all of its components must have high availability, and the failure of individual components must not adversely effect other components. For example, users would find it fully unacceptable if they could not use a lift to leave a side valley because the communication link between the access system and the background system had failed. For this reason, all essential ski pass data is stored in the card as well as in a central location. This enables the system to continue operating for a certain length of time in the event of failure of the connection to the background system.

The connections to the individual components are implemented in a wide variety of ways. Systems (including large systems) can be networked reliably and economically using Ethernet and the TCP/IP protocol stack. However, small systems may use serial data transmission compliant with RS 232 or RS 422, although in mountainous regions it may be impossible to install cables due to environmental restrictions or steep terrain. In such cases, directional radio systems or data transmission techniques such as GPRS with GSM<sup>8</sup> are possible alternatives to wire-based communication. Here it must be borne in mind that radio links are not always possible due to terrain conditions, and that weather conditions that can temporarily disrupt radio communication are not uncommon in mountainous regions. This is another reason why the access systems (at least) must also be able to operate offline.

<sup>8</sup> See also Section 19.3.4, 'Bearer services', on page 802

In the simplest form, an access system consists of a terminal (usually called a reader in this connection), a motorized turnstile, and a control computer with an online connection to the background system. The combination of a terminal and a turnstile is called a gate. The terminal is responsible for contactless communication with the ski card, while the computer grants access if the user can be authorized for access. Gates can easily have a passage frequency of up to 600 people per hour.

In more elaborate forms, the computer may have a connected monitor that automatically displays a photo of the user (stored in the background system) after contact with the ski pass has been established. This enables the lift personnel to check (perhaps on a random sample basis) for unauthorized loaning of passes to other users. Due to the head coverings worn by skiers, automatic biometric recognition of facial features<sup>9</sup> is unlikely to be possible in the foreseeable future. The only practical form of automatic checking is measuring the user's height, which is actually done in some ski areas. This at least allows adults and children to be distinguished reliably.

High availability of the overall system is one of the essential requirements for trouble-free operation of a ski area. This also includes analyzing the visitor flows in real time. Among other things, this makes it possible to indicate that another lift should be put into operation if necessary. Another important function is real-time communication of equipment status to the control room. This makes it possible to recognize the failure of an infrastructure component without delay and to initiate suitable measures. This sort of infrastructure data is called operational data. It is supplemented by user data, which contains information about the people at the gates.

### 21.2.2 Ski cards

Smart cards in ID-1 format<sup>10</sup> are used for ski passes, which are often called ski cards. However, these cards are usually intelligent memory cards<sup>11</sup> instead of processor cards. This is because memory cards are less expensive than processor cards and the fact that ISO/IEC 15639 communication is used, which does not transfer enough power to operate currently available microcontrollers. The functionality of intelligent memory cards is fully comparable to that of simple processor cards, although they are slightly less versatile. This does not lead to any perceptible restrictions in this sort of application. In addition to the standard ID-1 form factor, another version has been available since 1995 in the form of a Swatch wristwatch with a built-in memory chip and antenna. Along with showing the time, this watch can act as a reloadable ski pass. This capability is achieved by integrating a suitable IC and an antenna in the watch.

Contactless data transmission is typically implemented compliant with ISO/IEC 15693<sup>12</sup> for vicinity cards and operates in the 13.56-MHz band. The data rate is around 26 kbit/s. The advantage of this data transmission scheme is that it enables reliable communication between the terminal and the smart card, even at a distance of up to 1.5 m. Naturally, the usual mechanisms are supported, such as error detection for data transmission and collision avoidance for contactless smart cards. In most cases the transferred power level is not enough

<sup>9</sup> See also Section 7.8, 'Identification of Persons', on page 187

<sup>10</sup> See also Section 3.1, 'Card Formats', on page 29

<sup>11</sup> See also Section 2.3.1, 'Memory cards', on page 20

<sup>12</sup> See also Section 10.10, 'Vicinity Integrated Circuit Cards', on page 344

**Table 21.4** Typical data elements of a ski pass and their access conditions

Data	Ski pass		
<i>Access conditions</i>			
READ:	mutual authentication of the smart card and the terminal		
WRITE:	mutual authentication of the smart card and the terminal		
Data object	Length	Format	Description
UID	8 bytes	...	Card serial number (unique identifier)
$K_{VICC}$	112 bits	...	Smart card key for authentic-mode data transmission (vicinity integrated chip card key)
$K_{MAC}$	112 bits	...	Smart card key for authentic-mode data transmission (message authentication code key)
Pass type	1 byte	...	Point card / Time card
Points	2 bytes	decimal number	Used only with point cards
Start date	6 bytes	DD.MM.YYYY	Ski pass validity start date
Start time	2 bytes	hh:mm	Ski pass validity start time
End date	6 bytes	DD.MM.YYYY	Ski pass validity end date
End time	2 bytes	hh:mm	Ski pass validity end time
Ski areas	2 bytes	bit field	Identifies ski areas where the pass is valid

to operate a microcontroller, but instead only a contactless memory chip, which nevertheless provides all the cryptographic security functions necessary for the application.

There are some systems that use data transmission compliant with ISO/IEC 14443, although reliable communication with this arrangement is limited to a maximum distance of approximately 10 cm between the terminal and the smart card. There are also a variety of older systems that use contactless smart cards operating in the 125-kHz band with proprietary protocols, but the trend is clearly heading toward data transmission compliant with ISO/IEC 15693 due to the large distance between the card and the terminal.

The most important data element in ski passes commonly used now is the 64-bit UID (unique identifier), which is a unique serial number assigned by the semiconductor manufacturer. It is stored in the smart card in unalterable form. The access system uses the UID to identify the associated ski pass and as a reference to all other data in the system. Except for the authentication key, no other data in the ski pass is necessary for online operation. However, it must also be possible to use ski passes offline in certain situations, such as when the connection to the background system is down. Accordingly, all data necessary for lift access in this situation is also stored in the ski pass.

Depending on specific needs, these data objects can be stored in memory regions of the intelligent memory card with read and/or write protection. Table 21.4 lists the typical data elements of a ski pass and their access conditions. One of these data objects is the ski pass type, which indicates whether it is a point card or a time card. With a point card, the number of points is stored, while with a time card, the start and end dates are stored. This value of the latter data object can range from a few hours with a half-day card to several months with a

season card. Depending on the region, the ski pass may also have a data element that identifies the ski areas where it is valid. Individual bits in a string are used for this purpose due to the relatively simple structure of memory cards. With a group of ski areas, this can be used to indicate the regions where the card can be used. Large data volumes, such as a photo of the user, are not stored in the ski pass due to a lack of storage space, since the card has only one to two thousand bits of EEPROM memory.

In addition to the ski pass data, the smart card contains a key for authentication and protection of data transmission. If triple DES is used, it has a length of 112 bits and is derived from a master key and the UID. The random number necessary for authentication is generated by a random number generator in the ski pass and stored there for the duration of the authentication process.

With contactless smart cards of this sort, the possible commands at the application level are essentially read commands, write commands, and authentication commands. The data generated during successful authentication can be used with subsequent commands for cryptographically secured communication. Authentic-mode communication is typically used, which means that a MAC is computed from the data to be transferred.<sup>13</sup> Triple DES<sup>14</sup> with two 56-bit keys is often used as the cryptographic algorithm.

The life cycle of a ski pass can be divided into two phases: administration and use. With disposable cards, the transition to the use phase is irreversible, while with reloadable cards a reverse transition is possible if the appropriate key is known.

### 21.2.3 Typical transactions

The number of different transactions with ski passes is relatively small. They essentially consist of identification of the ski pass with subsequent authentication, as well as communication in authentic mode for reading and writing data. All other activities can be performed in the terminal or the background system, independently of the ski pass.

#### 21.2.3.1 Identification and authentication

Sequence Diagram 21.2 on the facing page shows the usual command sequence for checking access authorization in the operational phase. The first step consists of determining the identity of the ski pass. To do so, the terminal reads the unique ID (UID) from the ski pass. As a result, the terminal (and the background system if necessary) knows the ski pass and all of the data it contains.

Following this, mutual authentication<sup>15</sup> of the terminal and the smart card is initiated. In this process, each party verifies the authenticity of the other party. If authentication is successful, the smart card knows that the terminal is genuine, and it allows access to its stored access data. In the other direction, the terminal is assured that the smart card is genuine. The illustrated sequence is generic, which means it can be followed by reading data from the ski pass and/or writing data to the ski pass.

<sup>13</sup> See also Section 7.1.4, ‘Message authentication code and cryptographic checksum’, on page 155

<sup>14</sup> See also Section 7.1.1.1, ‘DES algorithm’, on page 138

<sup>15</sup> See also Section 7.4, ‘Authentication’, on page 166

Terminal (VCD)	Smart card (VICC)
<i>Identify the smart card</i>	
READ BLOCK [tag for UID]	→ ← Response [UID]
<i>Authenticate the terminal and the smart card</i>	
K <sub>VCD</sub> = f(K <sub>MVCD</sub> , UID)	
READ BLOCK [tag for RND <sub>VICC</sub> ]	→ ← Generate RND <sub>VICC</sub> Response [RND <sub>VICC</sub> ]
Generate RND <sub>VCD</sub>	
X <sub>VCD</sub> = E(RND <sub>VCD</sub>    RND <sub>VICC</sub> , K <sub>VCD</sub> )	
AUTHENTICATE [X <sub>VCD</sub> ]	→ RND' <sub>VCD</sub>    RND' <sub>VICC</sub> = D(X <sub>VCD</sub> , K <sub>VICC</sub> ) IF (RND <sub>VICC</sub> = RND' <sub>VICC</sub> ) THEN terminal authenticated ELSE terminal not authenticated X <sub>VICC</sub> = E(RND <sub>VICC</sub>    RND' <sub>VCD</sub> , K <sub>VICC</sub> ) Response [X <sub>VICC</sub> ]
RND' <sub>VICC</sub>    RND' <sub>VCD</sub> = D(X <sub>VICC</sub> , K <sub>VCD</sub> ) IF (RND <sub>VCD</sub> = RND' <sub>VCD</sub> ) THEN smart card authenticated ELSE smart card not authenticated	←

**Sequence Diagram 21.2** Transaction process for mutual authentication of the smart card and the terminal. The random numbers generated in this process can be used with a card-specific key to generate session keys for data transmission in authentic mode. The following data is used in this process: the UID is the unique identifier of the smart card, K<sub>MPCD</sub> is a master key that can be used to derive the card-specific keys K<sub>PCD</sub> (terminal) and K<sub>PICC</sub> (smart card), RND<sub>PCD</sub> is the random number generated by the terminal, and RND<sub>PICC</sub> is the random number generated by the smart card

If an online connection between the terminal and the background system is available at this point, it is not necessary to read data from the ski pass because all the necessary data is provided via the online connection. Table 21.5 on page 885 shows the usual steps in this process.

Mutual authentication requires the exchange of two random numbers. These random numbers can be used to generate a session key for authentic-mode communication<sup>16</sup> following authentication, as illustrated in Sequence Diagram 21.3. This involves appending a message authentication code (MAC) to the data to be transferred, which the recipient of the data can check by simply recomputing it. This security method provides protection against manipulation during data transmission. It does not enable detection of deleted or inserted data packets during a session, but this capability is not essential for applications of this sort.

### 21.2.3.2 Reading data

A prerequisite for read access to the ski pass data is mutual authentication of the terminal and the smart card. If this is performed successfully, the ski pass allows read access to the corresponding user data. This use access simply consists of a read access to the user data and does not alter any of the data in the ski pass.

<sup>16</sup> See also Section 8.4, ‘Secure Data Transmission’, on page 225

Terminal (VCD)	Smart card (VICC)	
<i>Generate key for authentic-mode communication</i>		
$K_{MAC} = f(K_{MMAC}, \text{UID})$		
$KS_{MAC} = f(RND, K_{MAC})$	$KS_{MAC} = f(RND, K_{MAC})$	
<i>Read data</i>		
READ BLOCK [tag for DOx]	→	Read DOx from the memory $MAC_{VICC} = M(DOx, KS_{MAC})$
	←	<i>Response</i> [DOx    MAC <sub>VICC</sub> ]
$MAC_{VCD} = M(DOx, KS_{MAC})$		
IF ( $MAC_{VCD} = MAC_{VICC}$ )		
THEN DOx is authentic		
ELSE DOx is not authentic		

**Sequence Diagram 21.3** Generating a session key for communication secured with a MAC and reading one or more data objects from the smart card in authentic mode. Sequence Diagram 21.2 on the preceding page shows the prerequisite for this sequence. The following data is used in this process:  $K_{MMAC}$  is a master key used to compute the key  $K_{MAC}$  (used by the terminal and the smart card for authentic-mode data transmission) from the UID of the smart card,  $KS_{MAC}$  is a session key shared by the terminal and the smart card, RND is a random number known to both the terminal and the smart card (e.g.  $RND_{VICC}$ ), DOx is a data object with number x,  $MAC_{VICC}$  is the MAC computed by the smart card, and  $MAC_{VCD}$  is the MAC computed by the terminal

To prevent manipulation of the data during transmission or substitution of other ski pass data for presentation to the terminal after authentication has been performed with a valid ski pass, the transmitted data must arrive at the terminal with the assurance that it is authentic. For this purpose, a session key is generated using one of the random numbers exchanged during authentication and another key. This simplest way to do this is to perform triple-DES encryption on the random number using a key that is the same throughout the system. Another method that provides better security but is somewhat more complex is to use a card-specific key, which for example can be derived from the UID.

As the ski pass is usually an intelligent memory card, the data in the smart card is accessed directly by its memory address. However, logical addressing using the tags of the individual data objects is assumed in the following description for the sake of simplicity. This means that the terminal requests data objects from the smart card by supplying their tags. If corresponding data objects are stored in the smart card, they are returned to the terminal in the response; otherwise the return code indicates that the data objects could not be found. The terminal checks the authenticity of the received data by recomputing the MAC appended to the transmitted data. If the received data is authentic, it can be used as shown in Table 21.5 on the next page to check whether the pass holder is authorized to use the ski lift.

### 21.2.3.3 Writing data

Writing data to the ski pass uses essentially the same process as reading data. User access at the gates requires only read access and does not involve write access, which is used only during administrative access in order to modify user data.

Permission to write data to the ski pass requires successful mutual authentication of the terminal and the smart card, as shown in Sequence Diagram 21.2 on the preceding page. After

**Table 21.5** Basic steps for checking access authorization

Step	Activity
1	Read the UID from the ski pass
2	Authenticate the ski pass
3	If the UID is registered in a central or local instance of the database, read the access data and photo from the database.
4	If no database is available, read the access data from the ski pass in authentic mode
5	If the access data is within the allowed range, grant access

Terminal (VCD)	Smart card (VICC)
<i>Generate key for authentic-mode communication</i>	
$K_{MAC} = f(K_{MMAC}, UID)$	
$KS_{MAC} = f(RND, K_{MAC})$	$KS_{MAC} = f(RND, K_{MAC})$
<i>Write data</i>	
$MAC_{VCD} = M(DOx, KS_{MAC})$	
WRITE BLOCK [DOx    MAC <sub>VCD</sub> ] →	$MAC_{VICC} = M(DOx, KS_{MAC})$ IF ( $MAC_{VICC} = MAC_{VCD}$ ) THEN DOx is authentic ELSE DOx is not authentic Write DOx to memory
← Response [OK]	

**Sequence Diagram 21.4** Generating a session key for communication secured by a MAC and writing one or more data objects to the smart card in authentic mode. Sequence Diagram 21.2 on page 883 shows the prerequisite for this sequence. The following data is used in this process:  $K_{MMAC}$  is a master key used to compute the key  $K_{MAC}$  (used by the terminal and the smart card for authentic-mode data transmission) from the UID of the smart card,  $KS_{MAC}$  is a session key shared by the terminal and the smart card, RND is a random number known to both the terminal and the smart card (e.g.  $RND_{VICC}$ ), DOx is a data object with number x,  $MAC_{VICC}$  is the MAC computed by the smart card, and  $MAC_{VCD}$  is the MAC computed by the terminal

this, the terminal uses the UID of the ski pass to derive a card-specific key for communication in authentic mode. Subsequent data transmission is protected by a MAC so that manipulation of the data during transmission can be detected. The data is written blockwise to memory regions identified by associated memory addresses. Sequence Diagram 21.4 shows a typical process for writing user data to the smart card.

From a security perspective, it is better to use a card-specific key for write operations, but for the sake of simplicity the authentication key is sometimes used for write operations as well.

### 21.2.4 Future developments

Ski pass systems still have considerable latent development potential, and particularly in the area of expanded user functions they offer several attractive options for future evolutionary development. One of these options is booking ski passes directly from the user's place of residence or while traveling. This has been common practice on the Internet for several years, but it would also be possible to use mobile telephones for this purpose. In the future,

some mobile telephones will have an interface for contactless communication compliant with ISO/IEC 14443. This would allow the ski pass data to be requested via the mobile telephone and transferred over the air interface for immediate use. The mobile telephone could then be used in the same way as a conventional ski pass, thanks to its contactless interface. A disadvantage of this scenario is that ISO/IEC 14443 communication is designed for a maximum distance of 10 cm, so the telephone would have to be held relatively close to the terminal antenna at the gate.

This problem is avoided by systems such as Open Gate from Skidata [Skidata], which can easily operate at distances up to 1.5 m using ISO/IEC 15693 communication and can even communicate with a ski pass regardless of its position or orientation. For example, this means that it makes no difference whether the ski pass is in the user's left or right pocket. The reading speed is so high that skiers can pass through the gates without stopping. Particularly for skiboarders, who do not have poles for pushing off, this is a distinctly perceptible advantage. The Open Gate system uses multiple antennas on the terminal side so that the ski pass does not have to be aligned parallel to the terminal antenna and data can be exchanged with the pass in any orientation. The user only has to ensure that the ski pass is not stored in a pocket next to a mobile telephone or a sandwich wrapped in aluminum foil, since under unfavorable conditions this can screen the signal so much that it is not possible to establish a connection.

Another function that is already available in some ski areas is acquiring personal performance profiles for users. For this purpose, all lift accesses are stored in a database. Later on, the user can request various information via the Internet by using an access code that is visually personalized on the ski pass. This information may include the run kilometers skied by the user, the net difference in elevation, or a map showing the skied runs. Naturally, it is also conceivable that the data requested by the skier could be shown directly on the user's mobile telephone after suitable preparation.

However, the major topic in the infrastructure area is destination management. This means using information technology to link all of the elements of a ski area together. With this arrangement, visitors can use their ski passes not only for the lifts, but also to pay for all necessary goods and services for the duration of their stay. This requires the complete networking of all points of sale in the ski area, which has already been done in some areas.

With the constantly increasing use of ski passes, the financial turnover from ski passes is increasing as well. This makes them increasingly attractive objects of attack. In addition to conventional forms of attack that result in the production of cloned cards, contactless cards are especially vulnerable to relay attacks.<sup>17</sup> Although the user of a ski pass will not suffer any financial harm if another person manages to use the pass for lift access by using a relay attack, the situation is different if the ski pass also provides a contactless payment function. In this case the cardholder can suffer a financial loss.

Defense against this form of attack is certainly possible. For example, the ski pass can be equipped with a button that must be pressed to enable the payment function, and held for the duration of the payment transaction. This would make a relay attack impossible. Of course, requiring the user to press a button for lift access would be unacceptable because it would negate the advantages of contactless cards.

Methods similar to those commonly used with credit cards for many years now can also be used in ski pass systems to defend against relay attacks at ski lifts. In credit card systems, the location data of online transactions is checked to determine, based on the distance between two locations, whether the same card was used for both purchases or a cloned card was used.

<sup>17</sup> See also Section 16.5.3, 'Attacks on applications', on page 727

This approach can also be used in ski areas. Cloned cards can be detected rather easily by analyzing the presumed locations and typical elapsed times between gate passages at the base stations. As the system operates online, the cards concerned can also be blocked immediately. This can be extended to blocking the ski passes for the duration of lift travel. This only requires storing data in the background system to indicate that a particular ski pass passed the gate of a particular base station. After the user leaves the lift, the card is again enabled for the next trip. This temporary blocking requires either an additional gate at the top station or using the typical travel time to the top station to determine how long to block the pass.

## 21.3 TACHOSMART

In accordance with a regulation of the Council of the European Community [EC 98], the monitoring devices for road transport, which are called tachographs or in common usage trip recorders, must be replaced in the medium term by electronic tachographs with smart cards. The driver cards for these devices are designated by the highly generalized term ‘memory card’ in the regulation, since they must be able to store data.

The operating principle of the Tachosmart system is as follows. Using suitable distance and speed sensors, the tachograph (which is protected against manipulation) measures the distance traveled by the vehicle as well as its speed. It also has a real-time clock that is protected against external influences, as well as a smart card terminal. The vehicle driver is issued a personalized smart card called a driver card, which is used to identify the driver with respect to the tachograph unit.

The tachograph can thus monitor and log the amount of time each driver spends at the wheel and the vehicle speed. The EC ordinance stipulates that this information must be held in the smart cards for at least 28 calendar days for each driver. Data older than this may be overwritten as necessary. However, the actual tachograph must store data for the previous 365 days in a manner that is secure against manipulation.

As usual with EC regulations, the details are governed by national legislation and specifications generated by tachograph manufacturers. Unfortunately, the technical documents are confidential, so it is not possible to publish any details. However, it appears that the system will use digital signature smart cards that are compliant with the ISO/IEC 7816-4 and ISO/IEC 7816-8 standards. Digital signatures are necessary to protect the data for the last 365 days stored in the tachograph against modification.

There are also other types of cards in this system. They include test center cards, which are issued by government authorities to authorized centers so they can calibrate and program tachographs; control cards, which are used by government authorities; and company cards, which are used by vehicle owners to access the data stored in the tachographs and driver cards. A relatively extensive public key infrastructure (PKI) is necessary to allow the rather elaborate infrastructure of this system to be used throughout Europe reliably and economically.

## 21.4 ELECTRONIC TOLL SYSTEMS

In some countries, it is common practice to charge a toll for using certain roads. In contrast to flat fees collected by selling windscreen stickers, tolls are collected according to usage, which means that the amount to be collected depends on the frequency of use and the type of vehicle. In the past, tolls were usually paid in cash at toll booths. In recent years, a few isolated

instances of electronic toll systems (road-pricing systems) using various types of cards have been introduced. All systems used up to now have the drawback that they significantly impede traffic flow because vehicles must either stop or slow to a walking pace. The toll stations also require a large amount of space.

In light of this, the German traffic ministry decided in 1993 to launch a large-scale field trial of automatic toll collection systems. The road selected for this test was the A555 Autobahn between Cologne and Bonn. A variety of systems from ten different suppliers were tested on this stretch of road between May 1994 and June 1995.

The systems that were tested were evaluated with respect to several basic requirements. First, traffic should be able to flow normally and unimpaired past the toll collection points. In Germany, normal traffic flow can mean anything up to 250 km/h, so systems must be designed such that drivers cannot evade toll collection by driving very fast. Furthermore, the use of toll booths or baskets to collect tolls was not allowed because the authorities were only interested in systems with equipment suitable for installation on flyovers and sign bridges. In addition, the systems had to support both single-lane and multi-lane operation, since it was considered undesirable to fan traffic out over several lanes because this would entail additional construction and strongly impede traffic flow.

A supplementary requirement that was not originally foreseen arose during the course of the project, and the public increasingly regarded this requirement as the crucial criterion for the success of the entire field trial. This requirement was full anonymity with regard to toll collection. It should not be possible to generate vehicle movement profiles or monitor the routes traveled by individual vehicles. All proposed systems had a payment option that maintained vehicle anonymity as long as the toll was paid. If a toll collection failed, the vehicle was photographed and the registered owner was identified. The owner would then receive a suitable penalty notice. Naturally, this did not actually occur during the trial because real money was not used.

Almost all the proposed automated toll collection systems used smart cards to hold the electronic money. This is why we discuss this topic here, since it may become important to the smart card industry in the future.

All the systems tested had a device mounted in the vehicle, which was called the onboard unit (OBU) or the in-vehicle unit (IVU), along with additional equipment as necessary. The OBU or IVU was powered by the car battery. In the passenger compartment, each system had a smart card terminal with a display and a simple keypad, along with a link to the outside world, which was either unidirectional or bidirectional, depending on the system. This was usually a microwave link operating in the 5795–5805 GHz frequency band, as recommended by CEPT for this application. However, some systems used radio signals in the 400–500 MHz band or infrared data transmission. The disadvantage of an infrared link is naturally that transmission can be severely impaired by weather conditions, such as heavy snow or fog. The stated mass-production price of the OBU was €50–150 for the systems involved in the trial, depending on the configuration.

The control stations were installed along the motorway on flyovers and sign bridges as necessary. No modifications to the road infrastructure were necessary. The smart cards did not have sophisticated electronic purses, but instead only very simple and fast debiting commands. This was in part due to the fact that no real money was used, and in part due to the short time available for the transactions. In some systems, optimization extended as far as reducing the ATR to four bytes in order to leave sufficient time for debiting. This is understandable in light of the requirement for unrestricted traffic flow. At 250 km/h, a vehicle covers 70 m each

second. The control stations operating in the 5.8-GHz frequency band had a communication range of approximately 5 m. The resulting dwell time (the interval during which the vehicle is within range of the station) was approximately 70 ms. The following processes had to occur in the smart card during this interval:

- smart card reset and ATR transmission ( $\approx 10$  ms);
- DES encryption for authentication of the smart card ( $\approx 12$  ms);
- an EEPROM write operation to store the new balance ( $\approx 2 \times 3.5$  ms);
- data transmission from and to the card ( $\approx 30$  ms).

Additional time was necessary for data transfer between the OBU and the electronic toll station. It's easy to see that the amount of time available was very tight. The smart cards all operated at the maximum allowable clock frequency (usually 5 MHz). The data transmission rate between the OBU and the toll station was 1 Mbit/s, which was substantially higher than the data rate between the OBU and the card, so it had only a small effect on the total transfer time.

Here we describe three representative systems chosen from the ten systems involved in the field trial. Some of the systems in the trial were nearly the same except for technical details, so none of the systems described here is identified by name. They can be regarded as typical examples of their kind.

### System 1

The first system has a conventional infrastructure design. The necessary equipment is installed on two successive flyovers or sign bridges. The system is designed such that toll collection is not affected if vehicles change lanes between the two stations. There are two payment modes. In postpaid mode, accumulated tolls are debited from a bank account in the same way as with a credit card, which makes it very difficult to preserve vehicle anonymity. In prepaid mode, prepaid smart cards are used and the appropriate amount is debited from the card balance each time a toll is collected.

In technical terms, the system operates as follows. At the first station, the OBU and smart card are activated. Following this, the toll is levied and the vehicle size is measured roughly in order to classify it as a car or lorry. As the toll depends on the vehicle class, it is necessary to verify the class noted in the card. This is done by measuring the vehicle's height profile when it passes under the station, which is sufficient for reliable classification. When the vehicle is within the communication range of the second station, another link to the card is established via the OBU. The second electronic toll station checks whether the toll payment initiated by the first station was completed successfully. If it was not, the vehicle is photographed. With modern technology, vehicles can be identified uniquely from their number plate data using fully automated processes. The registered owner of the vehicle can then be sent a suitable penalty notice.

This system can be further extended as desired with various functions. For example, the card can be checked against a blacklist at the time of payment in order to eliminate the use of stolen cards. A subject of debate during the trial was whether the acquisition cameras should be able to photograph all passing vehicles in certain special cases, such as after a bank robbery with escape by car, in order to obtain information about the escape route. The advantage of

this system is that the OBUs are simple in construction and thus inexpensive. However, the electronic toll stations are correspondingly more elaborate and expensive.

### System 2

The second system is not based on a roadway infrastructure. Instead, it uses virtual toll stations that exist only as databases in the vehicle OBUs. The OBUs contain the coordinates of all available toll stations, along with the toll rates for the corresponding road segments. When a vehicle enters a defined road segment, the toll is debited from the smart card. Naturally, this transaction is much less time-critical than the debiting transaction used in the first system.

The OBU is constantly aware of the vehicle's geographic coordinates thanks to a connected Global Positioning System (GPS) module. GPS is a worldwide system for determining positions that was originally developed in the USA for military purposes with funding from the US Department of Defense.

The second toll system also requires monitoring, but this is done by random sampling, similar to current speed controls. The main advantages of this system are that equipment does not have to be installed along the roadway and tolls are levied at virtual locations. However, monitoring is more difficult and the smart cards cannot be used in postpaid mode because there is no information exchange between the OBUs and the outside world. Of course, this is advantageous in terms of anonymity.

### System 3

The third system represents the most ambitious technical solution for automated vehicle toll collection. The OBU is equipped with a GPS receiver and a GSM mobile telephone. The vehicle position is determined using the GPS unit, while the mobile telephone is used to obtain specific pricing data. As with the other two systems, the OBU contains a terminal with a smart card that is used for toll collection.

As in System 2, the toll stations are simply virtual locations along the roads that are defined by geographic coordinates. In this system, the road operator can adjust the rates as desired by using bidirectional communication with the OBU. The tolls can thus be made to depend on the time of day, location, vehicle class, and environmental considerations. The amount of the actual toll is calculated by the OBU in the vehicle.

In order to ensure a certain amount of anonymity, the tolls debited from the smart card in the OBU are not communicated immediately to the background system. Instead, they are accumulated in the card until a certain threshold is reached. The accumulated amount is then transmitted by mobile telephone for verification and billing. Thanks to its capability for bidirectional data exchange, this system can support debit cards (prepaid) as well as credit cards (postpaid). However, anonymity cannot be maintained in postpaid mode. With prepaid smart cards, which are used like electronic purses, the fact that only cumulative tolls are paid means that the motorway operator cannot generate vehicle movement profiles. However, the GSM network operator can continuously monitor the position of a moving vehicle by tracking its mobile telephone, which is constantly active.

In this system, as in the others, continuous monitoring of the traffic stream is necessary to check for drivers who try to evade tolls. This is done by using automatic cameras mounted on flyovers and mobile checkpoints. The associated monitoring computers can read vehicle registration data from the OBUs or smart cards via the mobile telephones and compare it

with the vehicle number plates. If they do not match or a connection to the OBU cannot be established, the vehicle is photographed and a penalty process is initiated.

The main advantage of this system is that it does not require the installation of a specific infrastructure other than what is needed for monitoring. However, this makes the OBU distinctly more expensive than in the other two systems. This system also has the advantage that it supports two payment modes (prepaid and postpaid) and data transmission to the vehicle.

# 22

## Smart Cards for Identification and Passports

Smart card technology is very well suited to use in the realm of personal identification, such as ID cards, identification documents, and passports. This use is also supported by the fact that smart card microcontrollers have a high level of security against attacks. The only noteworthy aspect is that at least in the case of identification documents and passports, the form factor does not correspond to the usual formats in the smart card world. However, these alternative form factors do not create any perceptible restrictions, since this application area is dominated by contactless data transmission.

Smart card technology is already well established in the passport area. The next step is citizen cards or ID cards issued by governmental organizations, such as the European Citizen Card (ECC). This still nascent application area certainly has the potential to considerably increase the general use of applications such as digital signatures.

### 22.1 FINEID PERSONAL ID CARD

In December 1999, Finland became the first country in the world to start issuing electronic personal ID cards. The name of this system is Finnish Electronic Identification Card (FINEID). Personal ID cards are issued in ID-1 format with a photograph, signature, and various other data. The card also has a smart card microcontroller in the standard location for contact data transmission.

FINEID smart cards are part of a public-key infrastructure and are based on a signature application compliant with PKCS #15.<sup>1</sup> The related specifications are public and can be obtained free of charge via the Internet [FINEID]. Each FINEID smart card is valid for three years, and it can be used as a travel document in most European countries. Only the human-readable visual information on the card is necessary for this purpose, which does not require using the microcontroller.

<sup>1</sup> See also Section 23.2, ‘Signature Applications Compliant with PKCS #15’, on page 909

However, the signature application on the smart card allows the FINEID card to be used to generate legally valid digital signatures, which makes the entire spectrum of e-commerce and e-government applications available to card users.

## 22.2 ICAO-COMPLIANT PASSPORTS

A passport is an official document that is issued by a national government to a citizen of the country concerned. Like an identification document, it establishes the identity of its holder with a claim to binding force, and in particular it authorizes the holder to travel to foreign countries and, as a rule, to return to the territory of the country that issued the passport.

Passports usually take the form of booklets with a page count of 32 or so, and they have various visible and invisible security features that can be checked visually or with special equipment. Passports are produced in ID-3 format, which means they have a height of 125 mm and a width of 88 mm. Any visas necessary for travel to a particular destination can be entered in the passport. A visa is an official confirmation that a person is allowed to enter, leave, or remain in a foreign country.

Passports are required to be suitable for manual and machine inspection at the borders of all countries. For this reason, they must be interoperable documents. The International Civil Aviation Organization (ICAO) [ICAO], which falls under the authority of the United Nations, has assumed responsibility for the standardization of documents such as passports. Almost all countries follow the specifications of these standards. The principal document is ICAO 9303, which consists of three parts that cover passports, visas, and travel documents.

Documents that comply with the ICAO standards are generally referred to as machine-readable travel documents (MRTDs). They have one or more machine-readable zones (MRZs) with text data. This data is printed in OCR-B font so it can be read unaided by humans and by relatively simple machines. The MRZ of a standard passport contains the following information: document type, issuing country, surname (primary identifier), given name (secondary identifier), passport number, passport number check digit, nationality, date of birth, date of birth check digit, sex, expiry date, expiry date check digit, and additional information at the discretion of the issuing country.

The events of 11 September 2001 stimulated many countries to incorporate RFID chips in their passports in the years following these events. This can be attributed in part to pressure from the USA, which threatened to suspend visa immunity for the citizens of certain countries if they did not incorporate RFID devices in their passports. At the time when this book went to print, more than fifty countries, representing 90 % of worldwide travelers, had introduced passports with RFID chips.

A RFID chip is used as a storage medium for the personal data already present in the passport as well as supplementary biometric feature data. This typically consists of a facial image and fingerprints, and in some cases an iris scan. The data in the EEPROM memory of the RFID chip, which has a capacity of at least 32 KB, is transmitted using the ISO/IEC 14443 Type A or Type B protocol, and the chip is located in a protective sleeve embedded in one of the two covers of the passport, along with its associated antenna.

The biometric data of the photo and fingerprint is stored in compressed JPG or JPG2000 format to make it independent of the evaluation method that is used. However, this has the disadvantage that it can be used as a basis for forgeries if it can be read out.

**Table 22.1** Data elements of the ICAO logical data structure (LDS). The data in DG1 is also present in the MRZ. DG2 to DG4 are called encoded identification features, while DG5 to DG7 are called displayed identification features. Data groups up to and including DG19 are specified

Data group	Description
DG1	<ul style="list-style-type: none"> <li>• Document type</li> <li>• Issuing state</li> <li>• Name</li> <li>• Document number</li> <li>• Check digit – document number</li> <li>• Nationality</li> <li>• Date of birth</li> <li>• Check digit – date of birth</li> <li>• Sex</li> <li>• Date of expiry of the document</li> <li>• Check digit – date of expiry of the document</li> <li>• Optional data</li> <li>• Check digit – optional data</li> <li>• Composite check digit</li> </ul>
DG2	<ul style="list-style-type: none"> <li>• Encoded face</li> </ul>
DG3	<ul style="list-style-type: none"> <li>• Encoded finger(s)</li> </ul>
DG4	<ul style="list-style-type: none"> <li>• Encoded eye(s)</li> </ul>
DG5	<ul style="list-style-type: none"> <li>• Displayed portrait</li> </ul>
DG6	<ul style="list-style-type: none"> <li>• Reserved for future use</li> </ul>
DG7	<ul style="list-style-type: none"> <li>• Displayed signature</li> </ul>
DG8 ... DG19	<ul style="list-style-type: none"> <li>• Additional data</li> </ul>

The passport data is organized into logical data groups (DGs). Accordingly, the passport data set is called a logical data structure (LDS). The data elements of the LDS as specified by the ICAO are listed in Table 22.1.

A mechanism called Basic Access Control (BAC) is specified to prevent uncontrolled (and under certain conditions unnoticed) reading of the data in the RFID chip. It prevents reading of any data in data groups DG1 (MRZ) and DG2 (photo) when the passport is closed. As this mechanism is not especially strong, a method called Extended Access Control (EAC) is specified to provide additional protection for the biometric data in data groups DG3 (fingerprint) and DG4 (eyes).

Sequence Diagram 22.1 on the next page shows the operating principle of Basic Access Control. The process begins with optical reading of the information in the passport's machine-readable zone (MRZ). This requires opening the passport and placing it on an optical scanner. This reliably prevents a wireless attack on the passport data by a party that does not have physical possession of the passport. It thus prevents surreptitious reading of the passport data.

Data in the MRZ is then used to generate a key for establishing a cryptographically protected link and a key for the authentication of the terminal and the passport. The following data is used for this purpose: the document number, the check digit of the document number, the date

Terminal (IFD)	MRTD (ICC)	
Optically read the MRZ of the MRTD		
Choose a random $K_{IFD}$		
Compute a key K from the MRZ data		
<i>Command</i> [fetch random number]	→ ← →	Generate $RND_{ICC}$ <i>Response</i> [ $RND_{ICC}$ ]
Generate $RND_{IFD}$		
$X_{IFD} := E(RND_{IFD} \parallel RND_{ICC} \parallel K_{IFD}, K)$		
<i>Command</i> [ $X_{IFD}$ ]	→	$RND'_{IFD} \parallel RND'_{ICC} \parallel K'_{ICC} := D(X_{IFD}, K)$ IF ( $RND'_{ICC} = RND_{ICC}$ ) THEN Terminal authentic ELSE abort Choose a random $K_{ICC}$ $X_{ICC} := E(RND_{ICC} \parallel RND'_{IFD} \parallel K_{ICC}, K)$ <i>Response</i> [ $X_{ICC}$ ]
$RND''_{IFD} \parallel RND'_{ICC} \parallel K_{IFD} := D(X_{ICC}, K)$ IF ( $RND''_{IFD} = RND_{IFD}$ ) THEN MRTD authentic ELSE abort		
Use ( $K_{IFD}$ XOR $K_{ICC}$ ) as a key for subsequent secure messaging	←	Use ( $K_{IFD}$ XOR $K_{ICC}$ ) as a key for subsequent secure messaging

**Sequence Diagram 22.1** Simplified process for mutual authentication of a terminal and a passport (MRTD – machine-readable travel document) and reading data from the passport in accordance with the Basic Access Control (BAC) mechanism

of birth, the check digit of the date of birth, the expiry date, and the check digit of the expiry date. This means that only data that is protected by a check digit is used, in order to increase the robustness of the method. It is thus possible to determine immediately after reading the MRZ whether all of the data necessary for key derivation is correct, instead of only later when authentication of the RFID passport has failed due to an incorrectly computed key.

The strength of the generated key can easily be calculated. There are  $10^9$  possible nine-digit numerical passport numbers,  $365 \times 100$  possible date of birth numbers, and  $365 \times 10$  possible expiry date numbers if the term of validity is 10 years. The resulting key has an entropy of  $1.3 \times 10^{17}$ , which approximately corresponds to the entropy of a DES key with a 56-bit key space ( $2^{56}$ ).

However, the statistical distribution of the input data for key generation is not uniform, and the possible range of values can be restricted if the passport holder can be seen. For these reasons, the access protection provided by the BAC is not especially strong. However, only data that can anyhow be read optically from the passport can be obtained in this way, so a highly secure method would not be especially appropriate in this case.

After the two keys have been generated, a communication link to the passport's RFID chip is established using a contactless data transmission protocol compliant with ISO/IEC 14443. After successful challenge-response authentication of the terminal and the chip, the chip grants read access to the passport data. The subsequent read access is performed using cryptographically protected data transmission (secure messaging) with a session key and a send sequence counter.

The Basic Access Control mechanism only protects data that is anyhow accessible to the passport holder. However, it does not provide adequate protection for reading more sensitive data, such as fingerprint data. Two or more fingerprints are stored in the latest RFID passports, and they must be reliably protected against secret reading, so stronger access protection is necessary. This is provided by the Extended Access Control mechanism, which is based on public key authentication of the terminal and the RFID passport and secure messaging proceeded by a Diffie–Hellman key exchange. The Extended Access Control mechanism is described in detail in BSI Technical Guideline TR-3110, among other places. The EU member states will probably agree to use Extended Access Control for the protection of sensitive data such as fingerprints and iris scans. Although this requires the establishment of an extensive public key infrastructure, it has the major advantage of providing significantly better data protection. In addition, the data remains under the control of the country that issues the passport, since it is not compulsory to tie every possible country of origin into the public key infrastructure.

# 23

## Smart Cards for IT Security

Up to now, smart cards have not achieved an especially significant position in the realm of information technology system security. Doubtless one of the reasons for this is that smart cards cannot be used directly with everyday computers, since a terminal is always needed to provide the interface between the card and the computer. Nevertheless, a reasonable number of quite interesting applications, which are also beneficial for users, are already in use in many locations. Signature cards, which among other things are well suited to protecting all sorts of business processes in public networks, are a typical example of such applications.

### 23.1 DIGITAL SIGNATURES

There are two fundamental prerequisites for the use of legally binding digital signatures: a microcontroller smart card with a powerful numeric coprocessor, and a clearly defined legal framework. The smart card is used to securely store the secret signature key and to generate the digital signature. The legal framework establishes binding conditions that apply to all parties to this smart card application.

Of course, digital signatures can also be used in a closed application that is fully independent of any legal context. However, in this case suitable contractual arrangements between the participating parties are necessary to make a digital signature binding on the party that generates the signature. In the business-to-business sector, for example, the use of digital signatures has been standard practice for several years. However, if the objective is an open system that can be used by parties that are not known to each other, a suitable legal framework is necessary. This legal framework makes digital signatures equivalent to normal signatures and ensures that they are enforceable in case of dispute, the same as handwritten signatures.

The potential applications for legally recognized digital signatures, which are thus equivalent to handwritten signatures, are practically unlimited. In the simplest case, they can be used to sign electronic letters and orders, but they can also be used in an electronic payment system to sign direct debit authorizations or in a home banking system to sign remittance orders. The most important application for digital signatures will doubtless be signing all types of contractual agreements.

### 23.1.1 Applicable standards

To enable interoperable use of digital signatures in an open commercial environment, the digital signatures must fulfill the relevant security requirements of various standards. With regard to signature cards, the applicable standards are ISO/IEC 7816-4 for the general smart card commands and ISO/IEC 7816-8 for the signature commands. With regard to card bodies, electrical properties and data transmission, the relevant standards of the ISO/IEC 7816 family are applicable. The authentication process between the smart card and the rest of the world is covered by ISO/IEC 9796-2. The basic mechanisms and procedures for digital signatures are addressed by ISO/IEC 14888. In addition, the structure and coding of certificates usually comply with X.509.

A digital signature card can be used to generate a signature that is equivalent to a real signature. This means that large monetary or material values can be involved, depending on the circumstances and the document bearing the signature. The system elements of a digital signature application that are relevant for security must therefore be evaluated. The proven means for this is an evaluation in accordance with the Common Criteria (CC).

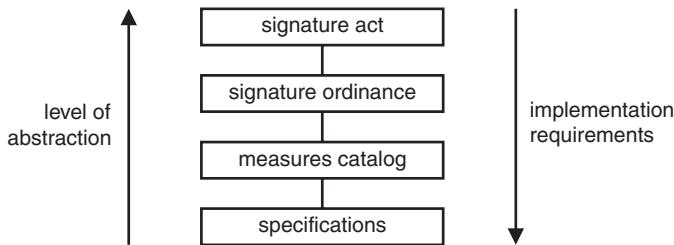
### 23.1.2 The legal framework in Germany

In order for digital signatures to be regarded as generally valid, binding and enforceable, it is essential to establish a suitable legal framework. The first legislation in this area was introduced by the state of Utah in the USA in 1995. It has served as a model for legislation in many other countries, including the German Digital Signature Act.

The *Gesetz zur Regelung der Rahmenbedingungen für Informations- und Kommunikationsdienste* (IuKD) was adopted by the German federal legislature (Bundestag) on 13 June 1997. It entered into force on 1 August 1997, and an amended version was published on 22 May 2001. Article 3 of this legislation is the German Digital Signature Act, commonly known as the *Signaturgesetz* (SigG). This article is divided into 16 sections. The first paragraph of Section 1 states the objective of the Digital Signature Act. It reads (in translation): ‘The purpose of the Act is to establish general conditions for digital signatures, under which they can be regarded as secure and which allow forgeries of digital signatures or falsifications of the signed data to be detected reliably.’ This statement clearly indicates that the Act does not specify a particular technical solution, but only general conditions regarding the use of digital signatures. As a natural consequence, the legal requirements are described at a relatively abstract level.

As shown in Figure 23.1, digital signature statutes in Germany have a hierarchical structure with an increasing level of detail and a decreasing level of abstraction in the progression from acts of parliament to ordinances, measures catalogs, and specifications. The essential stipulations of the German Signature Act are:

- Operating a certification authority (a trust center) requires the approval of the responsible authority.
- Any person who requests a certificate must be reliably identified by the certification authority.
- A signature key certificate may bear a pseudonym of the key holder in place of the true name of the key holder.



**Figure 23.1** The hierarchic structure of the specifications for digital signature applications in Germany. The system has a top-down structure, which means that the documents at each level are based on the documents at the next higher level

- The true name behind a pseudonym must be revealed on request of certain authorities.
- The certification authority must make certificates for public signature keys available in a manner that allows them to be verified online.
- A certificate can be blocked under certain conditions, such as when this is requested by the holder of the signature key.
- The technical components for generating and verifying digital signatures must provide a unique representation of the data referenced by the digital signature.
- Digital signatures from other countries may be recognized in Germany as well by means of suitable agreements.

According to the German Digital Signature Act, a signature key certificate, which usually complies with the X.509 standard, must contain the following information:

- the name or pseudonym of the signature key holder;
- the public signature key associated with the signature key holder;
- the identifiers of the algorithms used to generate the certificate (hash algorithm and signature algorithm);
- the serial number of the certificate;
- the initial and final validity dates of the certificate;
- the name of the certification authority;
- information regarding whether the signature key can only be used for certain applications.

Digital signature applications cannot be regulated and described by statutory means alone. This requires a hierarchy of requirements with the statutory regulation – the Digital Signature Act – at the top of the hierarchy. It is followed by the associated implementation provisions, which in Germany are contained in the *Signaturverordnung* (SigV) (Signature Ordinance). The level below this consists of the *Maßnahmenkataloge* (catalogs of measures), which describe the requirements for digital signatures in very specific technical terms. In Germany, they are issued by the regulatory authority for telecommunication and postal services. These three levels are mandatory for all digital signature applications in Germany if the digital signatures are intended to be equivalent to normal signatures.

The level below the catalog of measures consists of the technical specifications, which define specific implementations in a manner that is unambiguous and not subject to interpretation. The specifications depend on the individual application operator, and they must be consistent with the three higher levels of the hierarchy. However, if an application operator does not need legally binding digital signatures, the operator is free to configure a system in any desired manner that best meets the operator's perceived objectives. Such systems are referred to as legally noncompliant solutions, as opposed to legally compliant solutions.

The German Digital Signature Ordinance (SigV) contains the implementation provisions for the Digital Signature Act. It was adopted by the German federal government on 8 October 1997 and came into force on 1 November 1997. For each of the sixteen sections of the Digital Signature Act, it contains detailed descriptions worded significantly less abstractly than the Act. The Signature Ordinance includes the following provisions:

- A person who requests a certificate must be identified by means of a personal identification document, passport, or other suitable means.
- The data storage medium (usually a smart card with a signature key) must be held in personal safekeeping, and the associated identification data (such as a PIN) must be kept secret.
- When verifying a digital signature, the signature key certificate must be checked to verify that it was valid when the signature was generated and to determine whether it is subject to any use restrictions.
- The maximum term of validity of a certificate is five years.
- A blocked certificate cannot be unblocked.
- A catalog of measures with suitable security measures must be generated.
- The certification authority (trust center) must be audited by the responsible authority.
- The signature keys must be generated in a manner such that the keys are almost certainly unique.
- Signature keys may not be duplicated.
- All security-related modifications to technical components must be recognizable to the user concerned.
- A signature key may be used only after its holder has been identified by means of possession and knowledge.
- The signature key may not be revealed when it is used.
- A list of suitable algorithms for generating signature keys, computing the hash value of the data to be signed, and generating or verifying digital signatures shall be published in the *Bundesanzeiger* (German Federal Gazette). The time frame during which they may be used shall also be stated.
- If signed data must remain valid for longer than the terms of validity of the approved algorithms, the data must be signed anew, with a time stamp, before the expiry date of the relevant algorithm.

- The essential components of a legally compliant signature application must be evaluated according to the ITSEC criteria with the mechanism strength ‘high’. The evaluation level depends on the individual component and its use.

The catalogs of measures for the Digital Signature Act and Digital Signature Ordinance specify the technical requirements for this system. They are distinctly more specific than the Digital Signature Act. The technical specifications, which define the details of the technical implementation, are based on the catalogs of measures. In Germany, the most important technical specification for digital signatures is a DIN specification titled *Spezifikation der Schnittstelle zu Chipkarten mit digitaler Signatur-Anwendung/Funktion nach SigG und SigV*.

This DIN specification addresses the following topics with regard to signature cards:

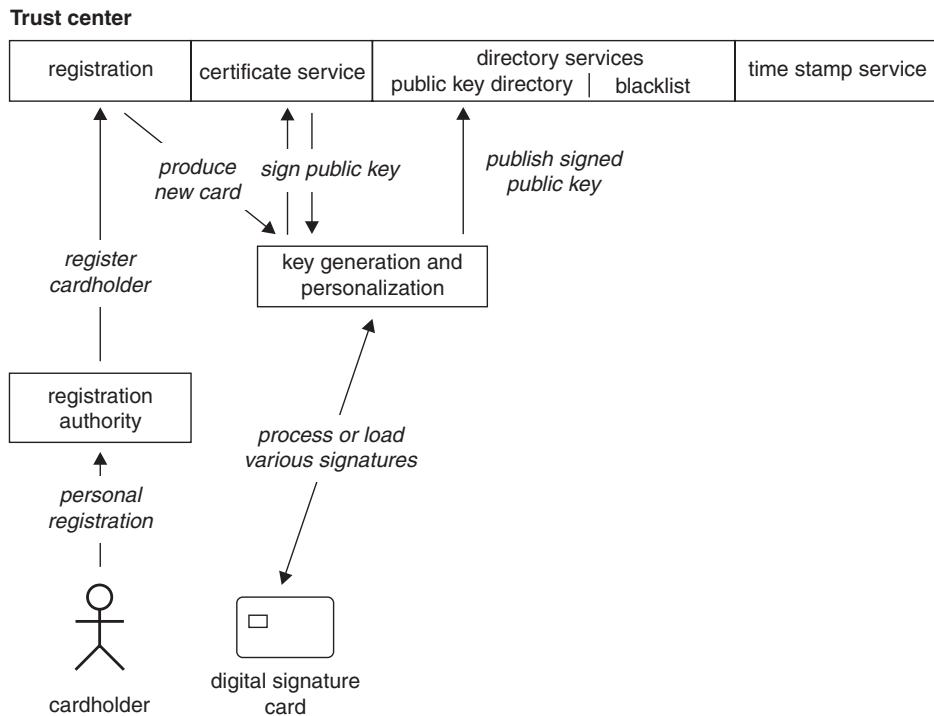
- answer to reset (ATR) and protocol parameter selection (PPS);
- transmission protocols ( $T = 0$  and  $T = 1$ );
- files, data objects and data formats;
- authentication of the cardholder;
- computation and verification of signatures;
- logging during the signature generation process;
- signature generation and verification processes;
- operation with terminals and associated command sequences.

### 23.1.3 System architecture

The two essential components of a digital signature system are the trust center and the digital signature cards. They form the main objects of attention here, since all processes related to security occur in them. In this regard, in a system for digital signatures we can make a fundamental distinction between issuing cards and the processes of signing and verifying signatures.

### 23.1.4 Card issuing

The basic process of issuing a signature card is depicted in Figure 23.2. The future user of a digital signature card must first register with a registration authority. The user must appear in person and present an acceptable proof of identification, such as a personal ID card. At this point, the future user of the digital signature card can elect to have a pseudonym appear in place of the user’s true name. The registration authority transfers the received and verified information to the trust center, which initiates the generation of a key and the personalization of a signature card. These activities can take place in the trust center, but they can also be carried out by a third party or directly within the registration authority. In this process, the public key of the new smart card is signed by the certification service of the trust center, which makes it a recognized authentic public key. The signed public key is then recorded in the



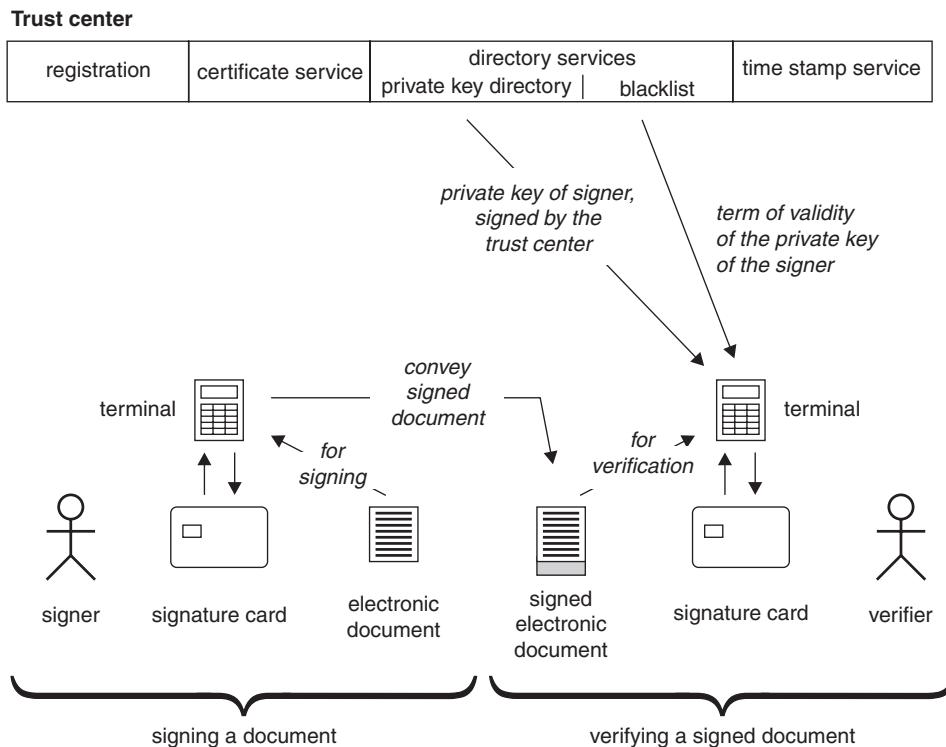
**Figure 23.2** Basic process for issuing a digital signature card

public key directory of the trust center as a valid key, which makes it available for use by every system user. After all this is finished, the new user receives a digital signature card and a PIN letter. This completes the process of issuing the card, which can now be used.

### 23.1.5 Signing and verifying documents

The basic processes for signing a document with an electronic signature and verifying the signature are depicted in Figure 23.3. When an electronic document is signed, it is first compressed to a hash value and then signed using the digital signature card with the private key. This process can be initiated only after unique identification of the cardholder (the signer), which currently means after the cardholder enters the correct PIN code. In the future, proof of identity may be based on a biometric feature (such as a fingerprint). The system components that are used for signing are the signature card, a terminal, and usually a PC.

The digitally signed document can now be sent in any desired manner. In order to verify the document, the hash value must be computed again. If necessary, the public key of the signer can be retrieved from the public key directory of the trust center. With this key, it is possible to verify that the digital signature of the electronic document is genuine. The revocation list of the trust center must also be consulted to ensure that the private key of the signer has not been compromised. This assures the signature verifier that the private key of the signer has not been blocked.



**Figure 23.3** Basic processes for signing and verifying a document with a digital signature

If the verifying party requires confirmation that the revocation list was consulted, in a form that can be checked by a third party, the verifying party can obtain suitable confirmation with reference to the verification data from the time stamp service of the trust center.

The system architecture and processes for issuing digital signature cards and signing and verifying documents with digital signatures that are shown here are examples, which may differ from actual implementations. However, they provide a realistic if simplified overview of a typical digital signature system.

### 23.1.6 Trust center (TC)

Along with the signature cards, the trust center is an essential component of a digital signature system. It provides six specific functions: registering new users (registry service), key generation and personalization of new digital signature cards, a certification service, a public key directory service, a directory service for revocation lists, and a time stamp service.

The registry service collects personal information and verifies the identity of new users of digital signature cards. A registration authority normally acts as an intermediary between the trust center and a new user, so the user does not have to appear personally at the trust center. Key generation and personalization of the signature card do not necessarily have to be performed by the trust center; they can also be performed by another party, such as a card

producer. Key generation can be performed either oncard or offcard, although this function is only needed when a new card is issued. The certification service of a trust center signs public keys for digital signature cards using a private key belonging to the trust center, so that they can be recognized as being genuine. Naturally, a trust center can have more than one private key for generating certificates.

The directory service for public keys contains the public keys of digital signature cards signed by the trust center. The directory service for the revocation list contains a list of all blocked keys, which are keys belonging to digital signature cards that may have been lost or compromised. According the German Digital Signature Act, these directory services must be accessible via generally available public networks.

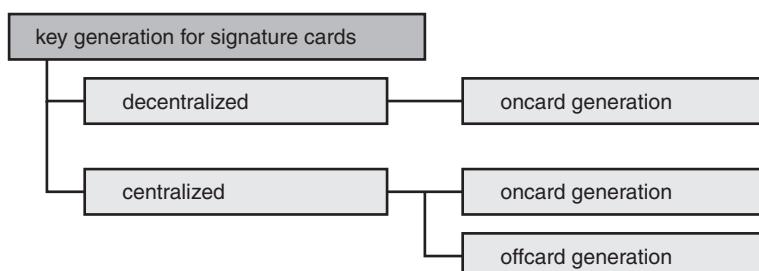
The time stamp service need not necessarily be included in a trust center, but it is certainly available in most trust center implementations. This service is used to mark electronic data presented to the trust center with the current date and time. The combination of the data received by the trust center and the date and time is signed using the private key of the time stamp service. This allows the provider of the information to prove to a third party that the data signed by the time stamp service must have been available no later than the indicated time.

### 23.1.7 Signature cards

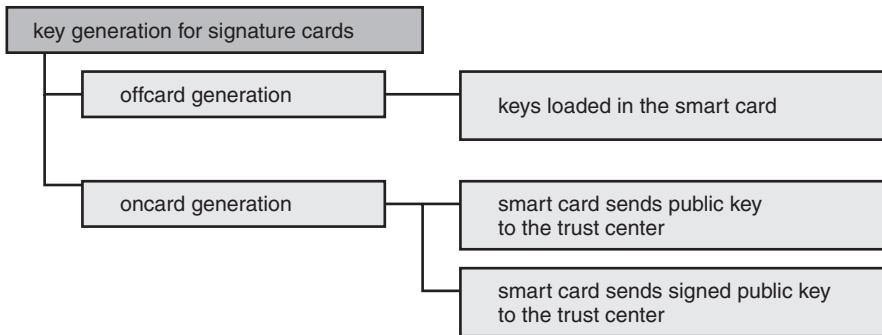
A secure hardware environment is necessary for storing and using private signature keys. Smart cards represent an ideal solution to this requirement, since they are physically small, inexpensive, and provide a high level of protection against external reading or modification of the stored data. It can be stated without reservation that digital signatures only became possible with the advent of smart cards.

The cryptographic algorithms used for this purpose require a microcontroller with a supplementary numeric processing unit (NPU) to allow signatures to be generated and tested in an acceptable length of time. In theory, a signed document could also be verified outside the signature card, since all of the necessary information is available outside the card. Whether this will become significant in practice remains to be seen. In terms of security, however, it is better to perform the verification in the card.

As shown in Figure 23.4, public and private keys for digital signature cards can be generated in either a centralized or a decentralized manner. In any case, the keys must be generated in a cryptographically secured environment, such as in a security module or smart card. It is also



**Figure 23.4** Key generation options for signature cards



**Figure 23.5** Three basic options for key generation and key distribution between the trust center and the signature card

**Table 23.1** Comparison of the three methods used for key generation. The method shown in the middle column is commonly used in large real-life applications

Method:	Offcard key generation	Oncard key generation with transfer of the public key to the trust center for signing	Oncard key generation with transfer of the signed public key to the trust center
Pro:	Fast personalization Card replacement with an identical card is possible	Private signature key never leaves smart card	Private signature key never leaves smart card
Neutral:	Key escrow easy in principle	Key escrow difficult	Key escrow difficult
Con:	Card replacement with an identical card not possible Private key generated offcard	Card replacement with an identical card not possible Public key must be sent to trust center by a secure route	Personalization time-consuming Private key of trust center in smart card

important for the generated public key to be signed using the private key of the trust center. This is necessary to ensure that the private and public keys of the signature card are recognized as authentic by the trust center. Without this recognition, it is not possible for a third party to verify that the digital signature of an electronic document actually originates from a genuine user of the digital signature system.

Keys can be generated using a centralized or decentralized process. With centralized key generation, all key pairs are generated in one place and signed using the private key of the trust center immediately after they are generated. As indicated in Figure 23.5, the keys can be generated and then signed by the trust center either oncard or offcard. Decentralized key generation can only be performed oncard, since the card is the only cryptographically secured environment in this situation. The pros and cons of the three options are listed in Table 23.1

**Table 23.2** File tree of a signature card compliant with the German Digital Signature Act (simplified version with only the most important files)

File type	FID	Structure and size	Description
MF	'3F00'	—	Root directory
MF.EF <sub>GDO</sub>	'2F02'	transparent, $n$ bytes	Card serial number and cardholder name
DF <sub>SigG</sub>	not defined	—	Digital signature application (AID = 'D2 76 00 00 66 01')
DF <sub>SigG</sub> .EF <sub>KEY</sub>	—	undefined, $n$ bytes	PINs and public/private keys
DF <sub>SigG</sub> .EF <sub>SSD</sub>	'1F00'	transparent, $n$ bytes	Security service description (SSD) (information about the properties of the signature card)
DF <sub>SigG</sub> .EF <sub>C.ICC.AUT</sub>	'C000'	transparent, $n$ bytes	Signature card authentication certificate
DF <sub>SigG</sub> .EF <sub>C.CA.AUT</sub>	'C008'	transparent, $n$ bytes	Certification authority authentication certificate
DF <sub>SigG</sub> .EF <sub>DM</sub>	'D000'	transparent, 8 bytes	Display message (DM) for the terminal so the user can recognize it as genuine
DF <sub>SigG</sub> .EF <sub>C.CH.DS</sub>	'C100'	transparent, $n$ bytes	Cardholder signature certificate
DF <sub>SigG</sub> .EF <sub>C.CA.DS</sub>	'C108'	transparent, $n$ bytes	Certification authority signature certificate
DF <sub>SigG</sub> .EF <sub>PK.CA.DS</sub>	'B000'	transparent, $n$ bytes	Public key of the certification authority
DF <sub>SigG</sub> .EF <sub>PROT</sub>	'A000'	cyclic, $n$ bytes	Signature generation log

on the preceding page. In practice, centralized oncard key generation will probably become predominant because it is very secure and corresponds to the usual smart card production process.

In Germany, the most important document for the signature card interface is the previously mentioned DIN specification. It extensively describes the necessary commands, files and processes for digital signature cards.

The digital signature application is located in a DF directly below the MF. The DIN specification defines nine EFs for this application, which contain all of the necessary data. All of the data elements are TLV coded. The EFs and their data contents are summarized in Table 23.2.

The following commands are used with signature cards:

- APPEND RECORD
- EXTERNAL AUTHENTICATE
- GET CHALLENGE
- INTERNAL AUTHENTICATE
- MANAGE SECURITY ENVIRONMENT
- PERFORM SECURITY OPERATION
- READ BINARY
- READ RECORD

- SELECT
- UPDATE BINARY
- MANAGE CHANNEL (optional)
- GET RESPONSE (optional for T = 0)

The hash functions and signature algorithms specified for German signature cards are SHA-1, RIPEMD-160, RSA, DSA, and ECC. As it would take too long to compute the hash value of a large electronic document inside the smart card, the hash value can be computed in a PC and then passed to the card for signing. Alternatively, the hash value can be computed in the PC up to the final block, following which the final block can be computed from the corresponding part of the document in the smart card and then signed. It is also possible to completely compute the hash value in the smart card, but this is often not practical due to the well-known performance limitations of smart cards.

### 23.1.8 Summary and prospects

In Germany, the Digital Signature Act is the legal framework necessary to enable digital signatures to be used reliably in everyday life. They will probably be used most often in everyday activities such as e-mail correspondence and commercial transactions via public networks, which are inherently insecure, rather than wills or private purchase contracts for property.

It can safely be assumed that beside legally compliant solutions, there will be many solutions that are not legally compliant because the implementation of a noncompliant solution is easier and much less costly. The requirements imposed on the security of the components used for digital signatures are very stringent in Germany, which means they do not favor quick or inexpensive solutions. However, this level of security is necessary if digital signatures are to be made equivalent to handwritten signatures. If successful manipulation of digital signatures ever proves to be possible, the entire digital signature system will lose credibility, and in the worst case it will have to be shut down.

Another important consideration in this regard is the behavior of multinational organizations, which are not inclined to pay significant attention to the legislation of any individual country. Consequently, it would certainly be possible for a solution that is not legally compliant in Germany or elsewhere to become established as an international quasi-standard.

Despite these risks, the German digital signature implementation has worldwide influence with regard to legislation as well as the technical implementation of similar systems. If the potential of this system is properly utilized, Germany will have a technically superior and exemplary implementation of the digital signature application.

## 23.2 SIGNATURE APPLICATIONS COMPLIANT WITH PKCS #15

In 1998, RSA Inc. [RSA] presented a paper titled ‘Cryptographic Token Information Format Standard’. This specification encompasses the description of files and associated data formats for a cryptographic token. Such a token is preferably a smart card with a cryptographic

processor. The PKCS #15 specification ('PKCS' stands for 'public key cryptography standards') is based on the Digital Certificates on Smart Cards (DC/SC) and Secured Electronic Information in Society (SEIS) specifications for digital signature applications using smart cards. The PKCS #15 specification is also described now in a compatible form in the ISO/IEC 7816-5 standard for smart cards, but the following description is based on the PKCS #15 specification because it has become established worldwide as a reference document.

Various requirement criteria were observed in generating the PKCS #15 specification. It had to be neutral with respect to specific platforms, manufacturers and applications, and to comply with the usual standards. Furthermore, it was required to be modular and extendable. Another requirement was that all application data objects must be adequately described so they can be used as references for access.

The ISO/IEC 7816 family of standards was selected as the basis for representing files, with ASN.1 being used to describe formats. The commands necessary for processing the data do not form part of the PKCS #15 specification. Consequently, PKCS #15 can be implemented relatively easily in a smart card with a typical multiapplication operating system. However, the concept allows any desired security token to be used for the PKCS #15 functions.

PKCS #15 has presently been implemented in a number of large smart card applications, such as the Finnish FINEID<sup>1</sup> personal ID system based on smart cards, the Wireless Identification Module<sup>2</sup> (WIM), digital signature applications in SIMs and USIMs for the Wireless Application Protocol (WAP), and similar purposes in the telecommunication sector. Within a relatively short time, the PKCS #15 specification has become established as a broad-based international industry standard for application data representation in smart card digital signature applications.

The entire PKCS #15 application is located in a DF with a fixed AID, as illustrated in Figure 23.6. The RID is 'A0 00 00 00 63', and the associated PIX is the ASCII encoding of "PKCS-15" and thus consists of the byte string '50 4B 43 53 2D 31 35'. However, the PIX may also have a different value, depending on the implementation.

An entry in an optional EF<sub>DIR</sub> file directly below the MF, as specified by ISO/IEC 7816, may be used as a reference to the DF holding the PKCS #15 application. If several PKCS #15-compliant applications are present in the smart card, they can be identified uniquely and selected using the EF<sub>DIR</sub> file. All other files for the application are contained in a directory named DF<sub>PKCS #15</sub>.

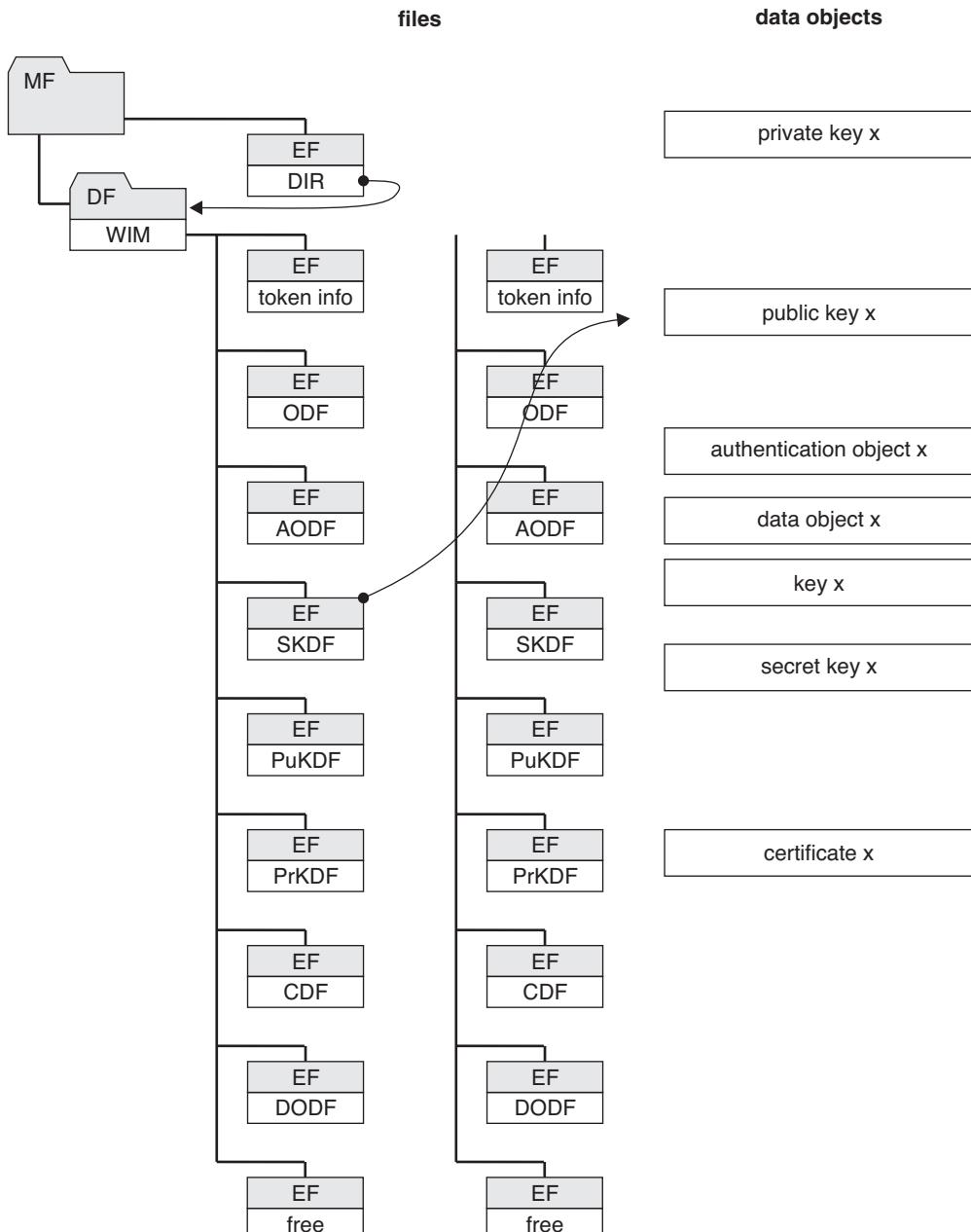
The EF<sub>TokenInfo</sub> file in the DF<sub>PKCS #15</sub> directory holds general information about the smart card containing the PKCS #15 application. This includes a serial number, a manufacturer identifier, and the supported cryptographic algorithms.

The central and most important EF of the PKCS #15 application is the object directory file (EF<sub>ODF</sub>), which holds a list of directory files. These files are referenced by two pointers that allow them to be identified uniquely. This EF is effectively a sort of root directory, from which all other EFs of the application can be accessed using pointers.

The directory files EF<sub>PBKDF</sub>, EF<sub>PoKDF</sub>, EF<sub>SKDF</sub>, EF<sub>CDF</sub>, and EF<sub>AODF</sub> have the same internal structure, consisting of a directory with each directory item having a pointer to the associated data. This data can be stored in EFs in the DF<sub>PKCS #15</sub> directory or in files located in another DF of the smart card. The pointer addressing scheme can also be used to reference data in EFs with transparent structure using offset and length parameters. It would thus be conceivable,

<sup>1</sup> See also Section 22.1, 'FINEID Personal ID Card', on page 893

<sup>2</sup> See also Section 19.6, 'The WIM', on page 854



**Figure 23.6** Overview of the files and data referenced by pointers in a PKCS #15 signature application. The variable  $x$  represents a nonzero positive integer index that can assume different values for the various data objects

at least in theory, to store all the data in a single large EF, although this would hardly be a sensible approach to data organization. The pointer linking system is even designed to allow data external to the card to be referenced using URLs. A typical use for this mechanism would be depositing a signature verification certificate in a trust center so that it does not have to be stored in the smart card. The smart card would then only reference the certificate by means of its URL.

At least one set of directory files must be present in the card, but it is possible to create more than one file with the same functional purpose in a single card. In this case, EF<sub>ODF</sub> acts as a dispatcher to the individual files.

The private key directory file (EF<sub>PrKDF</sub>) and the public key directory file (EF<sub>PuKDF</sub>) hold directories of the private and public keys, respectively, that are present in the smart card. These two files thus effectively contain keys for asymmetric cryptographic algorithms, such as RSA, DAS, and elliptic curve algorithms. The file directory holds pointers to the associated keys. The secret key directory file (EF<sub>SKDF</sub>) has a similar structure and holds a directory with pointers to the secret keys of the symmetric cryptographic algorithms in the smart card.

The certificate directory file (EF<sub>CDF</sub>) and the authentication object directory file (EF<sub>AODF</sub>) hold directories with pointers to certificates and authentication objects, respectively. The certificates are usually X.509 certificates, while the authentication objects are typically PINs and/or data for biometric user identification. Similarly, the data object directory file (EF<sub>DODF</sub>) holds a directory with pointers to data objects.

The optional EF<sub>UnusedSpace</sub> file can be used for managing unused memory in EFs belonging to the PKCS #15 application that are already present in the smart card.

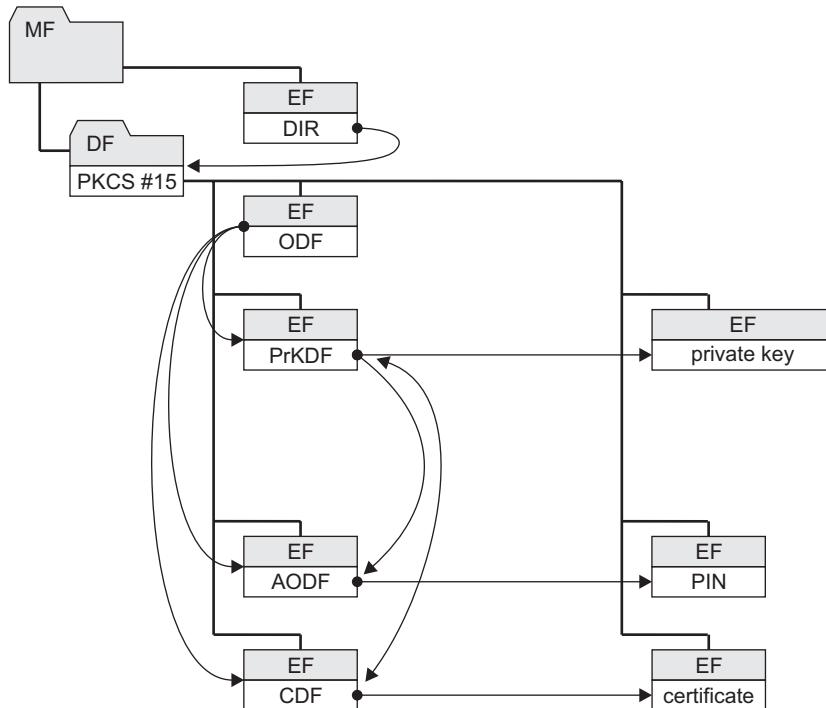
In the original specification, the ASN.1 description of the possible data for a PKCS #15 application encompasses 14 pages. We do not consider it advisable to reproduce this description line by line here, since the original document can be obtained at no charge online from the RSA website [RSA], and we are not fond of multipage listings that only serve to increase the size of this or any other book.

The PKCS #15 specification supports rule-based access control for certain data objects, such as signature keys, private keys, and certificates. In accordance with the concept of this specification, this is again implemented using pointers that link the protected data objects to corresponding directory files holding information about the type of authentication to be used. For example, this provides a simple means to compel the user to enter a PIN before using a signature key. In this case, the signature key data in EF<sub>PrKDF</sub> is linked by a pointer to an entry in EF<sub>AODF</sub> that holds information about the PIN to be used. The individual records of EF<sub>PrKDF</sub> and EF<sub>AODF</sub> hold pointers to the corresponding EFs that hold the actual data. To complete this example, a pointer in EF<sub>PrKDF</sub> marks an entry in EF<sub>CDF</sub> holding information about the certificate associated with the signature key, so that the signature can be verified by third parties. This example is illustrated in Figure 23.7 on the next page. Although this pointer-based concept may appear complex and nonintuitive at first glance, it is very powerful, flexible, and above all open to future extensions.

### 23.3 SMART CARD WEB SERVER (SCWS)

For many years, the SIM Application Toolkit<sup>3</sup> (SAT) has made it possible to generate applications that are controlled by the smart card (SIM or USIM). However, the user interface

<sup>3</sup> See also Section 19.4.4.6, ‘SIM Application Toolkit’, on page 833



**Figure 23.7** Example of a typical PKCS #15 pointer structure used to protect a signature key by means of prior PIN query, with a link to a certificate that can be used to verify the data signed using the signature key. This diagram shows only the files necessary for a basic understanding of the structure

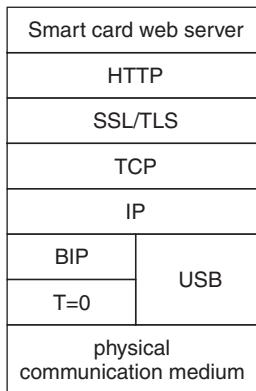
is text-based and thus outdated. This was presumably the main rationale for developing an improved user interface, which takes advantage of the user interfaces of networked computers in order to avoid developing an entirely new approach.

The solution consists of implementing an HTTP server that runs in a smart card. This allows established user interface technologies to be utilized, while at the same time achieving good integration of smart cards in the existing Internet world. HTTP servers in smart cards are called smart card web servers (SCWSs), and they have been available for many years. For instance, an HTTP server was implemented in a Java card as early as 1999 [Rees 99].

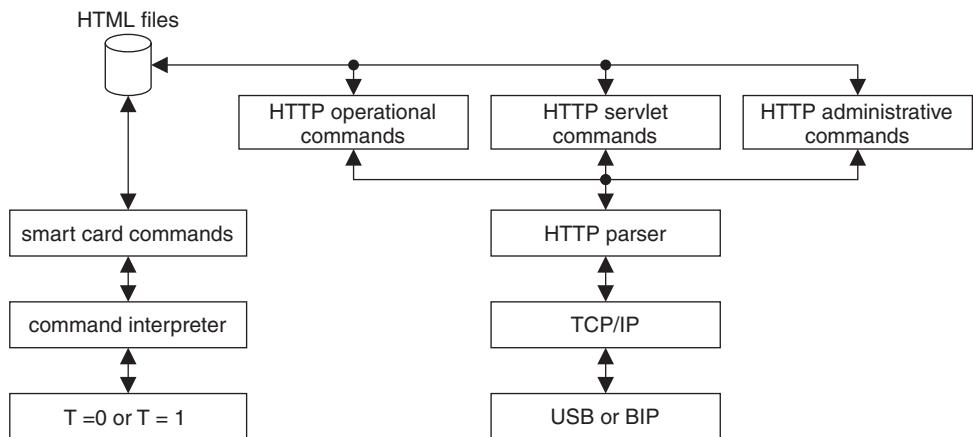
An HTTP server, which is also called a web server, is simply a program that interprets HTTP requests and returns corresponding responses to clients.<sup>4</sup> For this purpose, it supports the following HTTP commands: GET, HEAD, POST, PUT, DELETE, TRACE, and CONNECT. The functional scope of a smart card web server is defined in the OMA specification ‘Open Mobile Alliance Smartcard Web Server’, and the corresponding Java Card interface is specified by TS 102 588.

To provide cryptographic protection of data transmission, HTTPS (HTTP with SSL or TLS) can be used instead of plain HTTP. This allows authentication of the server, the client, and the user to be performed as specified in RFC 2617.

<sup>4</sup> See also Section 8.6.2, ‘HTTP protocol’, on page 235



**Figure 23.8** The underlying protocol stack of a web server in a smart card with the two communication options BIP and USB



**Figure 23.9** Basic architecture of a smart card web server with the two typical communication paths of a SIM or USIM card. The USB or BIP interface is used for the regular exchange of HTML data with the smart card as well as the administration of this data, while standard smart commands are used exclusively for administration

Figure 23.8 shows a typical protocol stack of a smart card web server. As smart cards are not typical network devices, there are always transport layers located below TCP/IP that handle the connection to the protocols used in the smart card realm. There are two options for this. The first option is a USB connection, which means that the TCP/IP protocol communicates with the USB interface of the smart card via a suitable device driver. If USB is not available or should not be used, the TCP/IP protocol can use the Bearer Independent Protocol<sup>5</sup> (BIP) to communicate with smart cards used in the telecommunication sector.

From a superficial perspective, the architecture of a smart card web server is relatively simple, as illustrated in Figure 23.9. A parser that interprets incoming HTTP requests is located

<sup>5</sup> See also Section 8.6.3, ‘BIP protocol’, on page 236

on top of the obligatory protocol stack. The interpretation utilizes the HTTP command, the associated parameters, and the supplied URL. If the parser cannot interpret the received string, it generates a suitable HTTP error message; otherwise it branches to the processing of the identified HTTP command. There are typically three categories of commands, which differ in terms of their functions and the permissions needed for their processing. The simplest category consists of commands that only request HTML contents. Typical examples of this category are the GET and POST commands. The second category consists of commands for managing HTML files, such as PUT, DELETE, POST, and GET. The third category, which is technically the most demanding, consists of commands that require the launching of a servlet.

The storage scheme of the HTML files depends on the file system supported by the smart card. Smart card microcontrollers with relatively large NAND flash memories often have a FAT-16 or FAT-32 file system that supports file names similar to those used with PCs. In this case, the HTML files addressed by HTTP commands can be returned directly to the client in the response, without any need for file name conversion. If on the other hand the smart card only has an ISO/IEC 7816 file system, file names requested in conventional PC notation must be converted to file names supported by the smart card. One way to do this is to use a simple translation table stored in an EF with linear variable structure. For each available HTML file, there is a record that holds the file name in PC notation and the corresponding file identifier (FID) in the ISO/IEC 7816 file system. In this case the contents of the HTML files are typically held in EFs with transparent structure.

One of the technical difficulties with HTTP servers in smart cards is that the client can easily send a series of GET commands to the server. Many browsers do this routinely because it allows page loading to be accelerated if the server has suitably high performance. However, with an HTML server in a smart card this leads to a not inconsiderable increase in demand for RAM and processing capacity. A smart card web server of moderate processing capacity with TCP/IP and SSL/TLS support needs around 100 KB of program code space and approximately 5 KB of RAM. This requires a smart card microcontroller at the upper end of the currently available performance range.

Although implementations of HTTP servers in smart cards have been available for many years, they have not yet achieved a large market penetration. However, this could certainly change in the future, since they allow smart cards to be used in combination with the user interfaces of modern HTML browsers, which are widely respected for their good features. In the telecommunication sector, this would involve running a normal HTML browser on the mobile equipment, which in certain applications would obtain its data from a smart card web server in the SIM or USIM card instead of the Internet. With the capability of providing imagery and multimedia contents in addition to text, this would enable the creation of entirely new applications that transcend what can be done with current SIM toolkits.

# 24

## Application Design

The first section of this chapter contains general information and characteristic data related to using smart cards. This information, which has been distilled from many applications, can be used directly for designing smart card applications. It thus represents a capsule summary of the current state of the art.

Much of the technical information in this chapter is strongly dependent on the actual hardware used, but it can nevertheless be used to generate estimates for projected applications that are adequate for practical purposes. In part, this is because the physical and electrical characteristics of all commonly used smart card microcontrollers are largely the same within certain limits. The only major differences are in their memory capacities.

In the second section of this chapter, we describe the basic features of a number of tools that allow even complex applications to be created simply and quickly without any need for programming. The chapter concludes with the description of a method for analyzing an unknown card.

The information in this chapter is structured in a general and flexible manner so it can be used to fashion design models for any desired application. Several suggestions, templates and examples are described in a book by Wolfgang Rankl [Rankl 06].

### 24.1 GENERAL INFORMATION AND CHARACTERISTIC DATA

#### 24.1.1 Microcontrollers

##### 24.1.1.1 *Production*

If a new application requires the development of a special smart card operating system, this will occupy a great deal of time. As a general rule, it takes around twelve months to design, program and test a completely new operating system. If library routines and existing modules can be used, a single programmer will still need around four to six months for this task. Performing these activities in parallel is possible only to a limited extent, due to the highly complex nature of programming in assembly language or C. As a result, generating a new smart card operating system takes a minimum of two to three months with current software

technology, even if no expense is spared and as many people as possible are assigned to the task.

Developing an operating system that supports executable program code is even more complex and costly. As a guideline, even under favorable conditions a development time of around eighteen months can be assumed if it is not possible to reuse existing designs or source code.

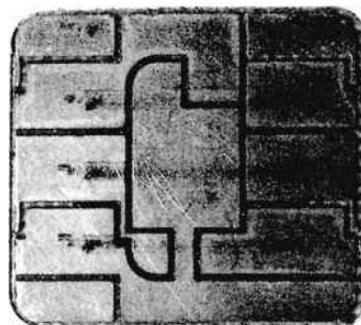
After the start of microcontroller fabrication by the semiconductor manufacturer, it takes eight to twelve weeks until finished dice are available in moderate quantities (several tens of thousands). Packaging the dice in the modules takes another one to two weeks. The overall lead time for producing the microcontrollers is thus around twelve weeks.

#### 24.1.1.2 Service life

The service life of a smart card essentially depends on the flexural strength of the card body, the corrosion resistance of the module contacts, and the number of EEPROM write/erase cycles.

The life expectancy of the card body is strongly dependent on the application area. With typical telecommunication cards (SIMs or USIMs), which are constantly located in a mobile telephone and never removed, there is practically no limit to the useful life of the card body. At the other extreme, such as employee identification cards that are used in the canteen and for access control, the card body may crack after two to three years. Keeping the card in a wallet carried in a hip pocket further increases the probability of failure of the card body.

The second limitation on the life expectancy of the card is the maximum number of insertion cycles. The card contacts have hard gold plating to increase their wear resistance, and they can survive approximately 20 000 contact cycles. After this, the contact surfaces will have become so scratched that dirt and grease will adhere to them and hinder reliable electrical contact (see Figure 24.1). In addition, extensive wear of the gold plating can lead to oxidation, which also adversely affects the electrical conductivity of the contact surfaces. Naturally, the life expectancy of the contacts (and thus the card) also depends very much on the type of contact unit used and the applied contact force. If the contact unit has an optimal design, the life expectancy of the card may be increased by a factor of two to four.



**Figure 24.1** A module after 50 000 insertion cycles. Very high-quality contacts designed for this type of tester were used in this test. In practice, the card contacts may wear significantly faster, depending on the type of contact unit used

The primary limitation on the life expectancy of a smart card arises from the limited number of write/erase cycles in the EEPROM.<sup>1</sup> Semiconductor manufacturers typically guarantee at least 100 000 write/erase cycles for each EEPROM page. However, in practice EEPROM failure will not occur before 2 million to 5 million cycles if the card is used at room temperature with only small deviations from the nominal supply voltage.

EEPROM pages usually fail gradually instead of suddenly. Two signs of incipient failure are that EEPROM locations cannot be set to the desired value on the first write attempt and that data written to EEPROM locations is no longer present in the memory after a few hours. If a memory page continues to be used under these conditions, it will not be possible to store any data at all in the EEPROM page after a few thousand additional cycles, since the contents will be lost immediately. However, only one page at a time is affected (typical page sizes are 4 or 32 bytes). Other memory pages remain unaffected by the failure of a particular page. This characteristic can be utilized to devise error recovery strategies when designing data storage structures.

The data storage principle of EEPROM is based on electrical charges stored in tiny capacitors. Like all capacitors, they have leakage currents that cause the charge to be lost over time, which results in the loss of the stored data. This process is accelerated at elevated temperatures. The data retention time of EEPROM is thus not unlimited, and the period guaranteed by the manufacturer is only ten years.

The difficulty with specifying the data retention time is that it cannot be measured directly, since this would require waiting ten years. Instead, the discharge time must be calculated by determining the value of the leakage current. Since this current is also virtually impossible to measure, the maximum data retention time is determined indirectly by extrapolating the results of tests made under various ambient conditions, such as at different temperatures and programming voltages. The data retention time also depends on the number of write/erase cycles. The specified value of ten years applies to maximum-stress conditions, so a much longer lifetime can generally be expected under normal conditions. In terms of application design, it can be assumed that data will be retained for ten years, but the design should not rely on this assumption.

All of the general conditions and constraints must be taken into consideration in order to estimate the lifetime of a smart card. A practical rule of thumb is three to five years in a normal application scenario without special requirements. In some areas, such as GSM, many organizations replace cards only as necessary (when they actually fail), which can be economically advantageous.

#### 24.1.1.3 Data transmission

The time required to process commands in the smart card depends primarily on the data transmission rate of the interface between the terminal and the smart card, as well as the internal processing speed.<sup>2</sup>

As an asynchronous serial interface is used, each transmitted byte is extended to twelve bits because a start bit, a parity bit, and two stop bits must be transferred in addition to the eight bits of user data. This means that a guideline value of 1.25 ms per byte can be assumed

<sup>1</sup> See also Section 5.3, ‘Memory Types’, on page 82

<sup>2</sup> See also Chapter 8, ‘Communication with Smart Cards’, on page 201

with a data transmission rate of 9600 bit/s. Adding a margin of 20 % for the unavoidable transmission protocol overhead yields a value of 1.5 ms/byte for data transmission. This is sufficiently accurate for making practical estimates of the time required for data transfer.

#### **24.1.1.4 Algorithm execution times**

As smart cards are often used as secure computers for processing algorithms, a few typical values of processing times for cryptographic algorithms are provided separately in Chapter 7, ‘Security Foundations’, on page 133, arranged by algorithm type.

### **24.1.2 Applications**

Besides the factors specifically related to the microcontroller, there are other aspects that should be considered in the conceptual design and development of a smart card application. With regard to conceptual design, they primarily relate to key distribution, managing application data, and the basic principles of data exchange.

#### **24.1.2.1 Key management**

In all applications that use cryptographic algorithms, security is based on the secrecy of the associated keys. If a secret key becomes known for any reason, all of the security mechanisms based on it are rendered worthless. In principle, this eventuality cannot be ruled out with complete certainty, so suitable precautions must be taken.<sup>3</sup>

The simplest and most costly option would be to replace all of the cards, but this is not economically feasible with a large application. As a guideline, the direct cost of card replacement can be taken to be €30 per card. Accordingly, certain principles are employed in practice to minimize the consequences of a key becoming known.

The first principle is that only card-specific keys should be located in smart cards. This means that each card has its own keys, which can only be used to clone a particular card if they become known. The associated master key must remain secret under all conditions.

In order to provide protection in case a master key becomes compromised, several generations of card-specific keys can be stored in the cards, so that the system can change to a new key generation if necessary.

The third principle with regard to keys is to use different keys for different purposes. For example, an authentication key is not allowed to be used for data encryption. This helps minimize the consequences of a key becoming known.

These three principles should always be observed in every large application, as they assure the application operator a significantly more secure system. They can also save the operator a great deal of money that would otherwise be spent on replacing the entire volume of cards.

However, in this connection it must be noted that each of these principles increases the number of keys in the system. This can quickly lead to memory problems, and it requires

<sup>3</sup> See also Section 7.7, ‘Key Management’, on page 180

more or less elaborate key management. In practice, it is necessary to consider very carefully whether all three of these principles should be employed, or whether it is possible to accept certain compromises.

#### 24.1.2.2 *Data*

The standard principles of information technology should be observed with regard to storing data in smart cards. In case of software applications, this means that classificatory numbering schemes should be avoided as much as possible because even small modifications or extensions often cause such schemes to collapse. However, identificatory numbering schemes are often excessively abstract, so mixed schemes predominate in practice.

Telephone number systems are a good example of a mixed numbering scheme. The first part of the number (the exchange number) is classificatory. If this number is known, the region where the telephone is located can be determined exactly. The second part of the number (the subscriber number) is purely identificatory, since at least in small towns, it provides no information about the location of the subscriber. The two parts together form a typical mixed number, which can be extended in a fairly straightforward manner.

ASN.1-coded data objects are very suitable for flexibly handling data objects that have several different versions.<sup>4</sup> Based on practice, the coding overhead of BER-TLV can be assumed to be approximately 25 % of the volume of user data. This applies only to small data sets, but this is precisely the usual situation in the smart card domain.

In this connection, it should be borne in mind that in the smart card realm applications with more than 10 KB of user data are rare, since the available amount of memory is usually very limited.

When a new application is being planned, the amount of memory that will be needed in the smart card should be estimated at least approximately. In order to do this, it is necessary to know not only the volume of user data, but also the necessary volume of administrative data. In modern operating systems, which have object-oriented file management systems and allow several applications to be present in a single smart card at the same time, the proportion of administrative data is relatively high. The size of the header for each file is usually in the range of 16 to 32 bytes. The amount of administrative data per file is fixed, which means it does not depend on the amount of user data in the file. It makes no difference to the size of the file header whether there is only 1 byte or 200 bytes of user data in the file. Consequently, designers try to avoid creating a separate file for each data element, as otherwise too much memory would be taken up by administrative data.

#### 24.1.2.3 *Data exchange*

Two primary considerations must be borne in mind with regard to data exchange between the terminal and the smart card. The first is that the card's serial interface is very slow compared with the performance of modern computer systems. Although the commonly used rate of 9600 bit/s makes data transmission highly immune to interference, it means that data

<sup>4</sup> See also Section 6.1, ‘Data Structures’, on page 109

exchange take a long time. It is therefore a good idea to limit the data exchanged between the terminal and the card to the essentials. During a session, the terminal should not again request any data it has already received. Naturally, this does not apply to applications in which the time required to perform operations is a very minor consideration. However, minimizing the volume of data exchanged via the interface must always be given high priority when people are involved in the process.

With regard to data exchange, the following considerations are also relevant. In principle, all of the data exchanged over the terminal–card interface can be protected very well by using secure messaging,<sup>5</sup> since this makes attacks via this interface virtually impossible. However, secure messaging reduces the effective transmission rate, so it should only be used for data transmission when it is indispensable for system security.

With the exception of secret keys, most user data can be protected adequately by simply using a MAC (authentic mode). This does not reduce the transmission rate as severely as the additional encryption required in combined mode. Data transmission at the interface level is also transparent in authentic mode, which means the data can be checked externally without any need for special tools or methods. Under certain conditions, this may be necessary in connection with data privacy legislation because it allows the data sent over the interface to be checked at any time without any knowledge of the secret keys.

### 24.1.3 System aspects

#### 24.1.3.1 Security

Many applications use triple DES as the cryptographic algorithm. There are many reasons for this, but in most cases the decisive reason is probably that DES and its cascaded version, triple DES, are very well known, so there is no need to venture into unfamiliar territory. However, there are many other good data encryption and decryption algorithms besides DES. The fact that DES is well known also means that it is subjected to the greatest number of attacks, one of which may sometimes be successful. For this reason, in fairly large smart card projects consideration should be given to using a cryptographic algorithm other than DES, since this will effectively minimize the consequences to the system if there is a successful attack on the DES algorithm. The algorithms currently used in the GSM system provide an excellent illustration of this approach. They were specially developed for this application, and to a certain extent they are fully independent of the DES algorithm.

#### 24.1.3.2 User interface

User acceptance is critical to the success of a smart card project. Although this is strongly dependent on the man–machine interface of the terminal, the interface between the terminal and the card also has an effect. Experience shows that user acceptance suffers when individual actions take longer than one second. If this happens, users often assume that there is a technical problem and attempt to pull the card out of the terminal, resulting in an uncontrolled termination

<sup>5</sup> See also Section 8.4, ‘Secure Data Transmission’, on page 225

of the session. In order to avoid this, all processes between the terminal and the smart card should be optimized to take less than one second.

One thing must always be borne in mind with regard to the man-machine interface: if user acceptance is inadequate, the system will experience a significantly higher level of technical problems due to unforeseeable user actions, such as prematurely withdrawing cards from terminals.

#### 24.1.3.3 High-level design

Kerckhoff's principle should always be applied in the high-level design of an information technology system that uses smart cards. This principle states that the security of a system should rely solely on its secret keys.

It is very difficult to evaluate a system whose security is based on the confidentiality of the data, since the evaluation can only be performed by a very restricted group of people. As in most cases this group of people is also responsible for designing the system, the quality of the resulting evaluation is necessarily relatively low. On the other hand, revealing all of the internal information is equally undesirable because it gives potential attackers free access to this information. In practice, the most reasonable solution is a mixed approach in which the fundamental design of the system is public but some specific features are kept confidential.

If, entirely contrary to Kerckhoff's principle, it is necessary for certain aspects of a smart card project to remain confidential, it is advisable to split the information between several people, rather than entrusting a single person with the security of the entire system. In this way, each of these people knows only part of the system and none of them knows the entire system.

With regard to designing a system based on smart cards, it is important to always keep the entire system in mind, rather than concentrating only on the cards and their immediate environment. This is the most commonly seen mistake in the course of many projects. As smart cards are active devices, systems that use them are always distributed systems, whose individual elements must act autonomously and in which pure master-slave relationships do not prevail. After the system has been integrated, it is essential that all of its components work harmoniously together – not just the cards with the terminals. It is thus necessary to acquire and maintain an overall perspective from the very beginning.

#### 24.1.4 Compliance with standards

Smart cards are one of the very few components in the entire realm of information technology that are strongly based on international standards. This is in part due to the necessity for compatibility between the products of many different producers of smart cards and terminals, and in part due to the requirement for interoperability between various smart card systems. Consequently, here we mention several general international standards that are usually supported by current smart card operating systems or with which these systems should be compatible. Naturally, the relevant specifications and national standards must also be observed in the development of an actual application.

The most important family of international standards is ISO/IEC 7816. It specifies the essential general physical and information technology constraints. Parts 4, 8 and 9 of this

family of standards are especially important with regard to file systems, commands, and file access conditions. However, it should be borne in mind that this family of standards specifies a sort of construction kit from which various items may be selected as desired. In other words, it is not necessary for every smart card to fulfill every aspect of the standard in all of its details.

The TS 102 221 specification for the UICC is a key document in the telecommunication sector. In addition, the TS 51.011 specification for the SIM and the TS 31.102 specification for the USIM are fundamental documents for worldwide telecommunication applications.

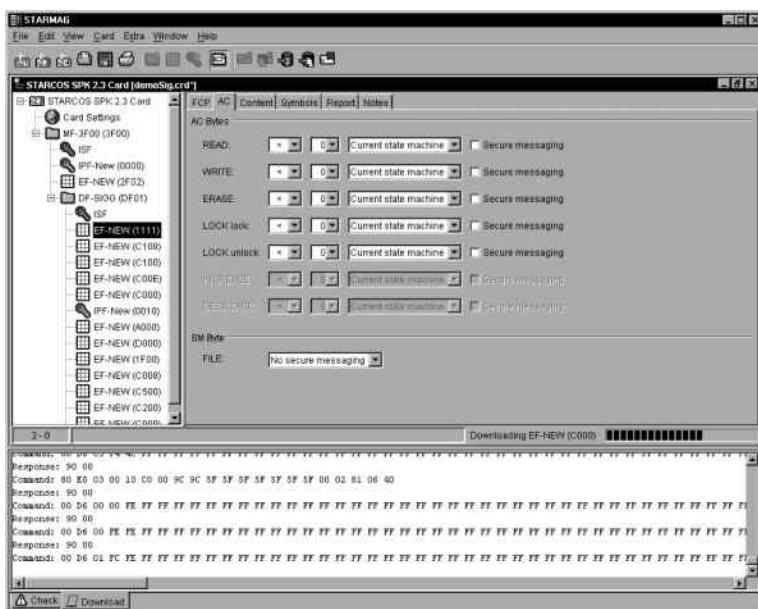
In the payment systems sector, the EMV specification is a fact of life that must be recognized by all other specifications. With regard to electronic purses, EN 1546 is a basic standard and CEPS is an exemplary model of an international purse system.

At present, smart card operating systems with executable program code are almost without exception based on Java Card and its associated specification. The Global Platform specification is used almost exclusively as the reference for downloadable applications and related topics.

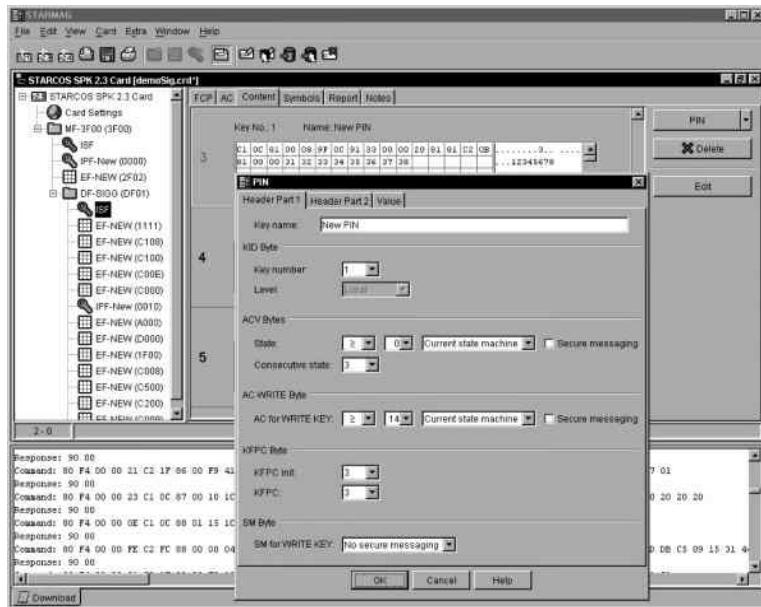
Smart cards in the digital signature environment are often based on the PKCS #15 specification. Certificates used with digital signatures are often stored in X.509-compliant formats.

## 24.2 APPLICATION GENERATION TOOLS

Several PC-based programs are available for developing smart card applications. They allow users to quickly and easily develop complete applications without having any particular knowledge of the internals of the operating system used in the smart card. Figures 24.2



**Figure 24.2** Screen display of a PC-based smart card application generator. The file tree in the smart card can be seen on the left, with an pane for entering the access conditions for a file on the right. A pane for displaying the data transmitted at the APDU level is located at the bottom (Reproduced with permission from Giesecke & Devrient)



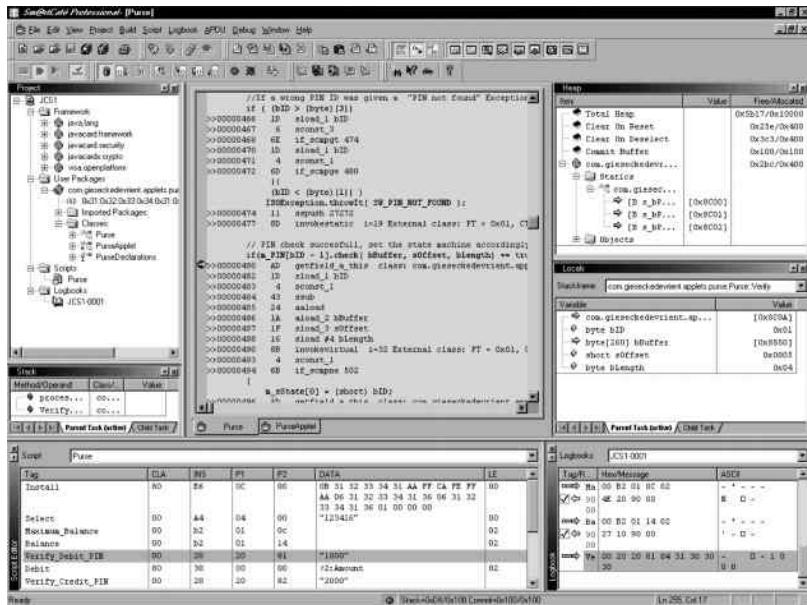
**Figure 24.3** Screen display of a PC-based smart card application generator. The file tree of the smart card can be seen on the left. To the right of the file tree, the content of the currently selected file is shown in the background, and the access rules contained in an EFARR file can be edited in the superimposed window (Reproduced with permission from Giesecke & Devrient)

and 24.3 show sample screen displays of a PC-based application generator, and Figure 24.4 shows the screen display of a PC-based development environment for smart card Java applications.

With such tools, the first task is usually to construct a file tree to hold the various applications (i.e. DFs) and their associated EFs. Naturally, it is necessary to specify the file structures and relevant access conditions for the EFs. If the smart card operating system has a state machine for commands, its parameters can also be defined using the graphical user interface of the application generation tool. Some application generator tools can also check the consistency of state machines. Since the application needs various data and keys in its EFs, a link to a database can be established after the files have been defined, in order to link the contents of the EFs in the individual cards to the data sets in the database.

Once the complete application has been defined using the application generator, various general parameters of the smart card operating system can be configured, such as the transmission protocol and associated divisor value. The application can then be experimentally loaded into one or more smart cards with suitable memory capacity. After several test cards have been produced in this manner, they can be tested with a terminal. If this shows that modifications are necessary, it is possible to delete the application from the smart card and then load a revised version.

If the results of all the tests are satisfactory and a large number of cards are needed, the necessary quantity can be produced using the normal card production process or a desktop personalization machine. The application data generated using the PC program (files, commands, states, etc.) forms the basis for the completion, initialization and personalization of the



**Figure 24.4** Example development environment with an integrated Java Card simulator for developing and testing Java programs for smart cards. The classes and methods are shown at the upper left in a tree structure, while the Java source code and the bytecode translated from the source code are shown in an adjacent pane to the right. The panes to the far right show the stack, the heap and several variables. The panes at the bottom show the script processor and decoding (Reproduced with permission from Giesecke & Devrient)

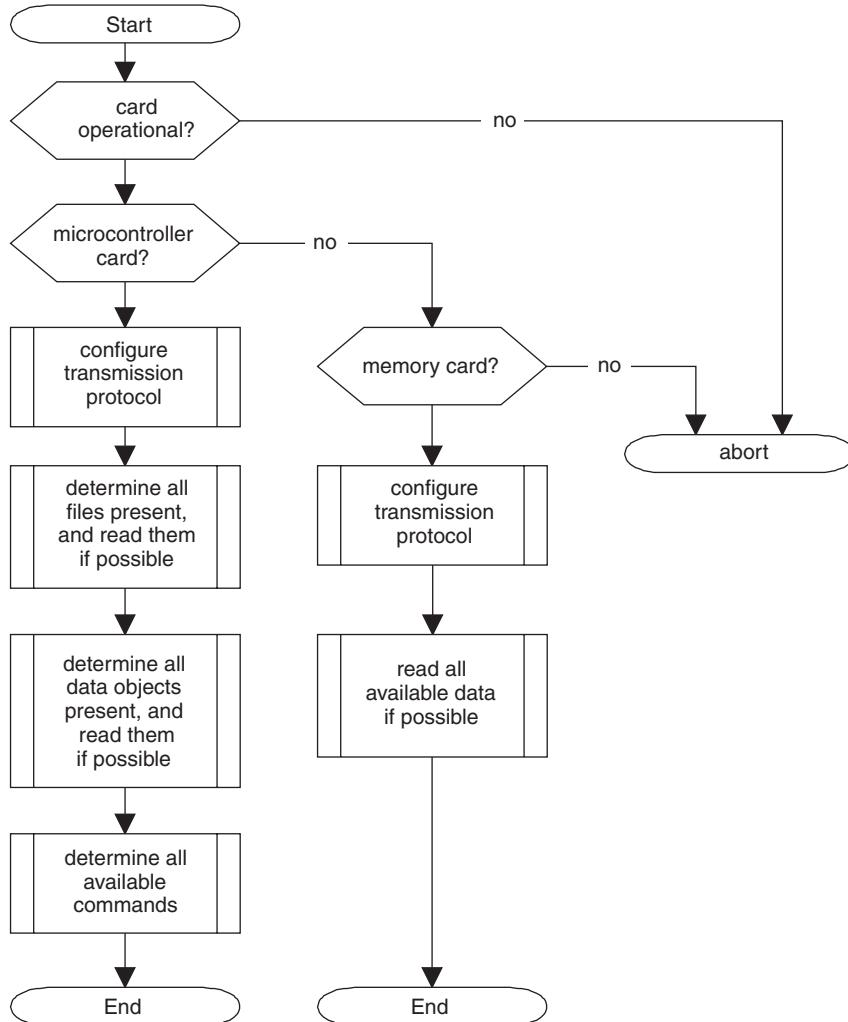
smart cards, so this approach combines a high level of flexibility with a very short lead time for producing finished cards.

In addition to these tools for designing and generating smart card applications, smart card simulators are available. A smart card simulator behaves exactly as a normal smart card, but it consists of a dummy smart card connected to a PC interface by a cable. Suitable software in the PC simulates the card in real time. Naturally, applications can also be generated and tested in the PC as described above. However, the PC is always required for the simulation, which frequently creates difficulties due to the complexity of the necessary setup.

If smart cards that support downloading of executable program code (such as Java Card smart cards) are used, an extra step for generating the smart card application must be added to the process described above. However, in practice the actual applications are indistinguishable from applications based on traditional smart card operating systems, even with current Java-Card-based operating systems, so the remaining steps are the same.

## 24.3 ANALYZING AN UNKNOWN SMART CARD

It is sometimes necessary to analyze an unknown smart card, for example to determine which smart card operating system it uses or which applications it contains. It is usually desirable



**Figure 24.5** A basic procedure for analyzing an unknown smart card

to keep the necessary effort within reasonable limits, so the equipment used for the analysis often consists of nothing more than a computer and a terminal.

The method for analyzing an unknown smart card described here is only one of many possible approaches. However, it has frequently proved itself in practice, although we should mention that considerable experience and extensive knowledge of smart card applications are necessary for successful evaluation of the results. Naturally, the method described here cannot be used to determine secret data or commands that are blocked by state machines. Nevertheless, experience has shown that this method can at least allow an unknown smart card to be roughly classified.

The procedure is outlined in the flow chart shown in Figure 24.5. The first step is to determine whether the smart card is actually operational. If it is, the next step is to determine

whether it is a memory card or a microcontroller card. Following this, for both types of card an attempt is made to configure the appropriate transmission protocol and read data held in the memory, files, or data objects. Based on this information, an attempt is then made to manually determine which applications are present in the smart card. In the case of a microcontroller card, it is often possible to determine which commands are supported and thereby draw conclusions about the smart card operating system present in the card.

# 25

# Appendix

In technical books such as this, the appendix usually contains a collection of general information, presented in tabular form, that is not specific to any particular chapter. Unfortunately, space limitations prevent the inclusion of this multitude of tables in the present edition of this book. In addition, information of this sort is especially prone to frequent modification, with the result that printed books are no longer the ideal data storage medium for this information. For these two reasons, the lists and tables located in the appendix of previous editions of this book have been moved to the website of Wolfgang Rankl [[www.wrankl.de](http://www.wrankl.de)], where they can easily be maintained and updated with the latest information as necessary.

## 25.1 GLOSSARY

The following pages contain a list of terms typically used in the smart card world. Precise, comprehensive definitions of terms can also be found in the ISO/IEC 7816 family of standards. The corresponding standard for terminology in the electronic purse sector is EN 1546, which comprehensively and concisely defines and explains all of the related technical terms. Terms are listed in this glossary either as abbreviations or in full according to customary usage.

A larger, generally available collection of general terms used in information technology and cryptology can be found in the Wikipedia (English version: [WikiEN]; German version: [WikiDE]).

**μC** → *microcontroller*

**μP card** An abbreviated term for → *microprocessor* card.

**1-μm, 0.8-μm, ... technology** The capacity of the technology used to fabricate semiconductor chips is traditionally expressed in terms of the dimension of the smallest possible transistor structure on the semiconductor material, which is usually the width of a transistor gate oxide strip. Currently, the smallest possible structure widths are approximately 0.13 μm to 45 nm. Naturally, it is always possible to fabricate chip structures that are larger than the minimum dimension.

**1G (first generation)** Refers to the first generation of mobile telecommunication networks, which have a cellular architecture and use analog technology. Typical examples of 1G systems are → *AMPS* and the German C-Netz.

**1K/2K/4K/. . . /nK chip** The term ‘*nK chip*’ (where *n* is a positive integer) is commonly used as a simplified designation of a → *microcontroller* with a certain → *EEPROM* capacity in kilobytes. A 32K chip is thus a smart card microcontroller with 32 KB of EEPROM. The EEPROM size is suitable for rough comparisons of typical smart card microcontrollers.

**2G (second generation)** Designates the second generation of mobile telecommunication networks, which have a cellular architecture and use digital technology. Typical examples of 2G systems are → *GSM* and → *CDMA*.

**3DES** → *triple DES*

**3G (third generation)** Refers to the third generation of mobile telecommunication networks, which have a cellular architecture and use digital technology. A typical example of a 3G system is → *UMTS*, which is a member of the → *IMT-2000* family.

**3GPP (Third Generation Partnership Project)** The task of the Third Generation Partnership Project [3GPP], which was founded by the five standardization organizations ANSI (USA), ARIB (Japan), ETSI (Europe), TTA (Korea) and TTC (Japan), is to generate internationally usable technical specifications for third-generation (3G) mobile telecommunication systems based on an enhanced GSM core system (*GSM*). The participating standardization organizations translate these specifications into corresponding standards. 3GPP was founded in Copenhagen by leading international standardization organizations in the telecommunication sector. The Third Generation Partnership Project 2 (3GPP2) [3GPP2] has similar responsibilities, although it focuses on upgrading non-GSM systems such → *CDMA* to the third generation.

**3GPP2** → *3GPP*

**4G (fourth generation)** Refers to the fourth generation of mobile telecommunication networks, which are still at the start of the conceptual design phase.

**8-bit/16-bit/32-bit CPU (central processing unit)** The width of the data register of the central processing unit (CPU) is a key factor in the processing power of a → *microprocessor*. It is stated in terms of the number of bits.

**A3 (algorithm 3)** Designation of a cryptographic algorithm used in a → *GSM* network for → *authentication* of the → *SIM* by the background system using a → *challenge-response method*. A3 is chosen by the network operator and is thus not the same throughout the entire GSM system.

**A5 (algorithm 5)** Designation of a cryptographic algorithm used in → *GSM* networks for encrypting voice data on the air interface between the mobile station and the base station or background system. A5 is the same throughout the entire GSM system.

**A8 (algorithm 8)** Designation of a cryptographic algorithm used in a → *GSM* network for generating a → *session key* (*Kc*) used for encrypting voice data on the air interface. A8 is chosen by the network operator and is thus not the same throughout the entire GSM system.

**Access conditions (AC)** In connection with a smart card file system, the access conditions of a file are a finite number of conditions that govern access to the file concerned via the operating system for operations such as reading, writing, or deleting data, with the conditions for each type of access generally being independent of each other. The access conditions must be satisfied in order to obtain the corresponding access.

**Access control** Access control consists of using equipment and/or services to control the entry and exit of employees or customers to or from secure locations. Magnetic-stripe cards are often issued to these individuals, with data stored on these cards granting access to specific groups of entry points at specific times.

**Acquirer** An entity that establishes and manages data links and data exchange between the operator of a payment system and individual service providers. An acquirer may consolidate the individual transactions that it receives so that the system operator receives only collective certificates.

**Activation sequence** Specifies the sequence of events for the activation of the electrical signals of a → *smart card microcontroller* in the process of powering up a → *smart card*.<sup>1</sup> It does not say anything about the sequence of events for mechanical contacting. The objective of the activation sequence is to protect the smart card microcontroller, which is sensitive to charges and voltages on its contacts. The sequence of events for deactivating the signals on contacts is defined by the → *deactivation sequence*.

**ADF (application DF)** An ADF<sup>2</sup> is a directory for an → *application* with subordinate → *EFs* that is located in the file system of a smart card but not located under the → *MF*.

**AES (Advanced Encryption Standard)** A symmetric → *cryptographic algorithm* originally developed by Joan Daemen and Vincent Rijmen and published as the Rijndael algorithm. Following a public competition and evaluation process, the → *NIST* selected this algorithm as the successor to the DES in 2000 and published it as a US standard (FIPS 197) in 2001.

**AFNOR (Association Française de Normalisation)** A French standardization organization based in Paris.

**AID (application identifier)** An AID identifies an → *application* in a → *smart card* in accordance with ISO/IEC 7816-5. Part of the AID may be registered nationally or internationally, in which case it is reserved for the registered application and is unique worldwide. An AID consists of two data elements: a registered identifier (RID) and a proprietary identifier (PIX).

**AMPS (Advanced Mobile Phone System)** A standard for cellular mobile telecommunication systems, predominantly used in the USA, Latin America, Australia, and parts of Asia. It utilizes analog technology and operates in the 800-MHz band. AMPS mobile telephones do not have → *smart cards* and are often successfully attacked, in part for this reason. The enhanced version of AMPS is D-AMPS, a digital system that also operates in the 800-MHz band.

**Analog** Refers to systems in which signals may assume an unlimited number of values.

<sup>1</sup> See also Section 4.6, ‘Activation and Deactivation Sequences’, on page 70

<sup>2</sup> See also Section 12.3.3, ‘Application dedicated file’, on page 425

**Analysis** In the context of software development, the process of determining the customer requirements for an information technology system and describing these requirements completely and unambiguously. In simplified terms, the result of the analysis is a description of what must be produced. The subsequent step in sequential software development is → *design*.

**Anonymization** Modifying personal data in a manner such that it is no longer possible to associate the modified data with the original person. → *Pseudo-anonymization* is a weaker form of anonymization.

**ANSI (American National Standards Institute)** A standardization organization with headquarters in New York.

**Anticollision method** A wireless data transmission method that enables access to multiple contactless cards without mutual interference.

**APDU (application protocol data unit)** An APDU<sup>3</sup> is a software data container used to package → *application* data for exchange between a → *smart card* and a → *terminal*. The transmission protocol converts the APDU into a transmission protocol data unit (TPDU) and sends it to the smart card or the terminal via the serial interface. APDUs can be classified into → *command APDUs* and → *response APDUs*.

**API (application programming interface)** A software interface, specified in detail, that provides access to specific functions of a program.

**Applet developer** A person or entity that develops an → *applet*.

**Applet** A program written in the → *Java* programming language and executed by the → *virtual machine* of a computer.<sup>4</sup> For security reasons, the functionality of an applet is restricted to a previously defined program environment. In the → *smart card* world, applets are sometimes called cardlets, and each applet usually corresponds to a smart card → *application*.

**Application** All of the data, files, commands, processes, states, mechanisms, algorithms and programs in a → *smart card* that enable it to be used in a particular system. An application and its associated data are usually located in a separate → *DF* directly below the → *MF* or in an → *ADF*. Such an application is often called an → *oncard application*. The counterpart to this is an → *offcard application*, which encompasses all of the programs and data not present in the smart card that are necessary for using the oncard application in the smart card.

**Application operator** An entity that operates an → *application* using → *smart cards*. The application operator is usually the same as the application provider.

**ASCII** American Standard Code for Information Interchange;<sup>5</sup> the most commonly used character code. A total of 128 ( $2^7$ ) characters can be represented with a 7-bit code.

**ASK (amplitude shift keying)** A modulation method in which the amplitude of the carrier signal is switched between two states.

<sup>3</sup> See also Section 8.3, ‘Message Structure: APDUS’, on page 221

<sup>4</sup> See also Section 13.16.2, ‘Java Card’, on page 499

<sup>5</sup> See also Section 6.2.1, ‘Seven-bit code’, on page 115

**ASN.1 (Abstract Syntax Notation One)** A data description language with a syntax and grammar that allow data and data types to be defined and represented unambiguously, independent of the type of computer system used.<sup>6</sup> The specific coding of the data is performed using the → *BER* (Basic Encoding Rules) and the → *DER* (Distinguished Encoding Rules). ASN.1 is defined by ISO/IEC 8824 and ISO/IEC 8825.

**Assembler** A program that translates assembly-language programs into machine language that can be executed by a processor. After the assembly process, it is usually necessary to link the resulting code using a → *linker*. ‘Assembler’ is also often used as a short form of ‘assembly language program code’.

**Asymmetric cryptographic algorithm** → *cryptographic algorithm*

**Asynchronous data transmission** A data transmission process in which data is transferred independently of any specified timing reference. It is the opposite of → *synchronous data transmission*.

**Atomic operation** One or more operations in a program that are executed either entirely or not at all. In the → *smart card* context, atomic operations are often used in connection with EEPROM write routines to ensure that the data content is consistent at all times.

**ATR (answer to reset)** An ATR<sup>7</sup> is a sequence of bytes sent by a → *smart card* in response to a software or hardware reset. Among other things, the ATR contains various parameters for the smart card transmission protocol.

**Attack** In the context of information technology, an attack is a method or procedure used to compromise the security of a system or a system component. The usual objective of an attack in this sense is to obtain or manipulate confidential data, manipulate program processes, or cripple the object of the attack. Known forms of attack on smart cards are → *timing attack*, → *differential fault analysis*, → *light attack*, → *SPA*, → *DPA*, → *side channel attack*, and → *relay attack*.

**Attribute** In the context of → *object-oriented programming*, an attribute is a data container (equivalent to a variable in a procedural programming context) that holds an → *object*. Attribute values can be read or modified using → *methods*.

**Authentication** Authentication<sup>8</sup> is the process of verifying the identity of a person or an entity, such as by using a particular method to determine whether an entity is actually what it claims to be. Authentication can be based on possession, knowledge, or a biometric feature.

**Authenticity** A property possessed by an entity or message that is genuine and unaltered.

**Authorization** The process of checking whether a particular action is allowed to be performed, which means whether someone has been granted the authority to do something. For example, when a credit card transaction is authorized by the credit card issuer, the card data is checked to verify that the data is correct, the purchase amount is less than the allowed limit, and other conditions are satisfied. The intended purchase is allowed

<sup>6</sup> See also Section 6.1, ‘Data Structures’, on page 109

<sup>7</sup> See also Section 8.1, ‘Answer to Reset’, on page 203

<sup>8</sup> See also Section 7.4, ‘Authentication’, on page 166

if all checks are positive. Authorization can be achieved by means of authentication of the entity concerned (such as a smart card). In simplified terms, authorization amounts to granting someone permission to perform a particular action.

**Auto-eject reader** A → *terminal* that can automatically eject an inserted card in response to an electrical or mechanical signal.

**Background system** Any computer system that processes and manages data and is located above the level of the terminals.

**Bad case** The case in which a logical decision leads to an unfavorable or undesired result.

**Bad-day scenario** Another term for → *bad case*

**Base wafer** A wafer that (from the perspective of the semiconductor manufacturer) does not yet include any customer-specific functions such as a → *ROM mask*.

**Baud** The term ‘baud’ designates the number of state changes per second during data transmission. Depending on the transmission method used, one or more data bits can be conveyed by each state change. For this reason, the baud rate is equivalent to the data rate in bits per second only in the specific case that one bit is conveyed by each state change.

**Bearer** Designates the bearer service used to transport data to a terminal device. For example, → *SMS* can be used as a bearer for → *WAP*.

**Belcore attack** → *differential fault analysis*

**BER (Basic Encoding Rules)** The Basic Encoding Rules (BER), which are defined in → *ASN.1*, allow data to be coded in the form of data objects.<sup>9</sup> A BER-coded data object has a tag, a length, and a value (the actual data component), and optionally an end marker, for which reason it is also called TLV-coded data. The BER format also allows chained data objects. The Distinguished Encoding Rules (DER), which are a subset of the BER, indicate among other things how the length parameter of the data object is coded (using one, two or three bytes).

**Big endian** → *endianness*

**Binary-compatible program code** → *Program code* that can be executed directly by a → *microprocessor* without interpretation.

**Biometrics** In the broad sense, biometrics is the science of measuring the bodily features of living organisms. In the information technology context, it refers to the automated identification of individuals using specific bodily features.

**Bit** A bit (binary digit) is a unit of data; one bit is the smallest representable amount of data.

**Blackbox test** A blackbox test<sup>10</sup> is based on the assumption that the party performing the test has no knowledge of the internal processes, functions, or mechanisms of the software under test.

**Blacklist** A database list of all cards or devices that are no longer allowed to be used in a particular → *application*. (*hotlist*, *greylist*, *whitelist*)

<sup>9</sup> See also Section 6.1, ‘Data Structures’, on page 109

<sup>10</sup> See also Section 15.4.2.3, ‘Blackbox test’, on page 649

**Bluetooth** A wireless network technology [Bluetooth] for short-range communication (<100 m) in the 2.4-GHz band, with a maximum gross data transmission rate of around 1 Mbit/s or up to 2.1 Mbit/s in an enhanced version. Ericsson, as the initiator of this technology, chose the name in memory of the Danish king Harald II, who lived approximately 1000 years ago and was nicknamed ‘Bluetooth’. His major achievement was merging many separate regions into a unified kingdom.

**Bond-out chip** A microcontroller mounted in a multipin ceramic package in order to provide free access to all of the memory buses inside the chip, so that the mask-programmed → *ROM* otherwise used for program storage can be replaced by memory external to the chip. A bond-out chip allows software to be tested on the target hardware without using a → *ROM mask*.

**Boot loader** A small, simple program whose only purpose is to load other, larger programs into memory, such as via a serial interface, and then run them from memory.<sup>11</sup> A boot loader is typically used to load the operational program code into a new chip or a new piece of electronic equipment. In many cases, the boot loading process can be performed only once.

**BPSK (binary phase shift keying)** A modulation method in which a sinusoidal signal is modulated by switching its phase between +180° and –180°. This yields two states, which can be used to encode one bit.

**Bring principle** → *push technology*

**Browser** A program used to view → *hypertext* documents, navigate between such documents, and run → *program code* embedded in hypertext documents.<sup>12</sup> Simple browsers that need little memory or processing power are often called microbrowsers. A micro-browser may run as an → *application* under a → *smart card operating system*, such as the WIB browser from SmartTrust [SmartTrust] or the S@T browser from SIM Alliance [SIMalliance], or it may be integrated into the software of a mobile telephone as a WAP browser or HTML browser. The functionality of browsers can be extended by downloadable software components called plugins.

**Brute-force attack** A computationally intensive form of attack on a cryptographic system based on computing all possible values of a key.<sup>13</sup>

**BSI (German Federal Office for Information Security)** The Bundesamt für Sicherheit in der Informationstechnik [BSI] was founded in 1991 as the successor to the Zentralstelle für das Chiffrierwissen. It is responsible for assessing the security risks of IT applications, testing and evaluating the security of IT systems, approving IT systems for use in government organizations, and assisting law enforcement bodies and organizations responsible for the protection of the German constitution. The BSI also advises manufacturers, operators and users with regard to IT security issues, and it often specifies the general constraints for cryptographic applications in Germany.

**Buffering** A typical form of attack on magnetic-stripe cards in which the data on the magnetic stripe is first read and stored (buffered). After this data has been modified with the

<sup>11</sup> See also Section 14.1, ‘Tasks and Roles in the Production Process’, on page 567

<sup>12</sup> See also Section 19.7, ‘Microbrowsers’, on page 857

<sup>13</sup> See also Section 7.1.1.1, ‘DES algorithm’, on page 138

aid of a terminal emulator program (such as changing the state of the retry counter), it is written back to the magnetic stripe.

**Bug fix** In the software development context, a bug fix consists of supplementary → *program code* used to remedy a known defect (bug). Unlike a → *workaround*, a bug fix eliminates the defect.

**Burst** → *signal burst*

**Bytecode** This term has several meanings and must be understood in context. One widely used meaning is related to → *Java*, in which bytecode is intermediate code produced from source code by a Java compiler for use by a → *Java virtual machine* (JVM). This bytecode is standardized by the Sun Corporation and executed by the → *interpreter* of a Java virtual machine. The term bytecode is also used in the context of microbrowsers (*Browser*), where it refers to the result of the translation of a → *hypertext* document by a bytecode converter. The bytecode produced by the converter is interpreted by a microbrowser.

**CAD (card acceptance device)** In electronic payment systems, ‘CAD’ is often used to refer to a smart card terminal in place of the ISO-compliant abbreviation → *IFD* (interface device).

**CAMEL (Customized Applications for Mobile Enhanced Logic)** An optional supplementary feature of → *GSM* used to support intelligent network (IN) functionality. With CAMEL, for example, a dialing number can be modified during call setup on the network. This provides a simple means to implement applications such as international → *roaming* with prepaid cards or to allow standard service numbers to be used worldwide.

**CAP file (card application file)** A file format used to exchange data between the Java offcard virtual machine and the Java oncard virtual machine.

**Card** A general designation for a thin, rectangular piece of material with rounded corners and standardized dimensions. A card can have various card components, including a semiconductor chip (*chip card, smart card*).

**Card acceptor** An entity with which cards can be used for a certain type of transaction (such as payment). A typical example is a merchant who accepts credit cards for making purchases.

**Card body** A plastic card forming an intermediate product in the production of smart cards. It is further processed in subsequent production steps and receives additional functional components, such as an implanted chip.

**Card component** A functional unit of a → *card*, such as a → *signature strip, embossing*, a → *magnetic stripe*, a chip (*memory card, processor card*), or a keypad (*system on card*).

**Card issuer** An entity responsible for issuing cards. In the case of mono-application cards, the card issuer is usually also the → *application operator*, but this is not necessarily the case.

**Card modeling language (CML)** An abstract, operating-system independent → *application* description language for smart cards.

**Card owner** A natural or legal person who has legal control over a card and the right of disposition with respect to the card. In the case of a credit or debit card, the bank issuing the card is often the card owner, while the customer who uses the card is simply the → *cardholder*.

**Card possessor** A person who possesses a card; usually the card user.

**Card producer** An entity that produces → *card bodies* and embeds → *modules* in them.

**Card reader** A device with a relatively simple electrical and mechanical construction that can accept → *smart cards* and make electrical contact with them.<sup>14</sup> Unlike a → *terminal*, a card reader does not have a display or keypad. Despite the name, most card readers can also write data to the card. The name originates from the era of magnetic-stripe cards, in which readers originally could in fact only read data.

**Card user** A person who uses a → *card*; usually but not necessarily the → *cardholder*.

**Cardholder verification method (CVM)** A method used for cardholder → *identification*.

This usually consists of PIN verification, although more advanced systems may use biometric user identification.

**Cardholder** A person who has the legal right to use a card. The cardholder need not necessarily be the same as the → *card owner*.

**Cardlet** → *applet*

**CAT (card application toolkit)** The CAT<sup>15</sup> forms the generic basis for all application toolkits for smart cards used in mobile telecommunication systems. It is a modular extension of the basic functionality of mobile phone cards that enables them to control the mobile telephone. → *SIM Application Toolkit*

**CAVE (cellular authentication, voice privacy and encryption)** CAVE is used for encryption and authentication in → *CDMA 2000* mobile telecommunication systems, in particular with the → *R-UIM*. CAVE is based on the Cellular Message Encryption Algorithm (CMEA), which is a symmetric → *cryptographic algorithm*. It has a variable block size of two to six bytes and a key length of 64 bits.

**Cavity** A recess in a → *card body* for the implanted → *module*, usually produced by milling.

**CCITT (Comité Consultatif International Télégraphique et Téléphonique)** The CCITT was originally an international committee for telephone and telegraph services, with headquarters in Geneva. With the assumption of additional responsibilities, its name was changed to → *ITU*.

**CCS (cryptographic checksum)** A cryptographically generated checksum for data, which is used to allow manipulation of the data during storage to be detected. A CCS used to protect data during transmission is called a message authentication code (*MAC*).

**CDMA 2000 (Code Division Multiple Access 2000)** A third-generation (3G) mobile telecommunication system operating in the 2 000-MHz frequency band with features

<sup>14</sup> See also Chapter 17, ‘Smart Card Terminals’, on page 735

<sup>15</sup> See also Section 19.4.4.6, ‘SIM Application Toolkit’, on page 833

similar to those of → *UMTS*. The smart card specified for use in CDMA mobile equipment, which is called the → *R-UIM*, is optional.

**CDMA (code division multiple access)** A multiple access method for the concurrent transmission of data from several transmitters to a single receiver within a frequency band. For this purpose, the narrowband radio signal is mapped (spread) into a wideband radio signal in the transmitter by using a transmitter-specific mapping scheme. If this mapping scheme is known, the receiver can recover the original narrowband signal from the received wideband radio signal. CDMA is used in → *UMTS* for the air interface between the mobile station and the base station. In the case of wideband code division multiple access (WCDMA), separate frequency bands are used for → *uplink* and → *downlink*, for which reason WCDMA is often called frequency division/code division multiple access (FD/CDMA). With time division/code division multiple access (TC/CDMA), the uplink and downlink are separated by using different time slots.

**Cell** In the mobile telecommunication realm, the smallest geographical division of a network.

**Cellular technology** Technology of an analog or digital mobile telecommunication system (such as → *GSM* or → *UMTS*) that is organized in cells. The transceiver stations of the network, usually called base stations, are normally located at the approximate centers of the cells.

**CEN (Comité Européen de Normalisation)** CEN [CEN] is a European standardization organization based in Brussels. It is composed of all national European standardization organizations and is the official institution of the European Union for generating European standards.

**CEPS (Common Electronic Purse Specifications)** An → *electronic purse* specification with emphasis on international → *interoperability*, including all components necessary for operating an electronic purse system. The first version was published in 1999 by CEPSCO [CEPSCO]. It is based on many of the principles of EN 1546, the European standard for electronic purses.

**CEPT (Conférence Européenne des Postes et Télécommunications)** A European standardization organization for national telecommunication companies.

**Certificate** A digitally signed public key with associated management data that is issued by a trustworthy entity so that other parties can recognize it as authentic (*PKI*).<sup>16</sup> The best known and most widely used specification for the structure of certificates is the → *X.509* standard.

**Certificate revocation list (CRL)** A list, maintained by a → *directory service*, of all certificates in a → *PKI* that are blocked and no longer accepted.

**Certification** A term sometimes used, not entirely correctly, as equivalent to → *evaluation*.

**Certification authority (CA)** In a public key infrastructure (*PKI*), an entity that guarantees the authenticity of public keys for → *digital signatures*.<sup>17</sup> For this purpose, the certification authority signs the user's public key with its private key and makes the

<sup>16</sup> See also Section 7.6, 'Certificates', on page 178

<sup>17</sup> See also Section 15.5, 'Evaluation of Hardware and Software', on page 659

signed public key available as needed in the form of a → *certificate* held in a directory (*directory service*). The certification authority can itself generate the key pairs (public and private keys) necessary for this process. For organizational reasons, certification authorities often have a hierarchical structure in which the highest certification authority is called the top-level CA or root CA.

**Challenge–response method** The method most often used for authentication in the smart card realm.<sup>18</sup> It is based on a secret key for a cryptographic algorithm, which acts as a shared secret of the communicating parties. One of the parties sends the other party a random number (the challenge). The second party encrypts the random number using a cryptographic algorithm and sends the result (the response) back to the challenger. The challenger then applies the reverse function of the cryptographic algorithm to the encrypted version of the random number it has received and compares the result with the originally sent random number. If they match, the challenger knows that the other party also knows the secret key, and from this it concludes that the other party is authentic.

**Chinese remainder theorem (CRT)** A technique used to accelerate the → *RSA* algorithm. As it requires knowing both prime numbers  $p$  and  $q$ , it can only be only used for decryption or signing.

**Chip card** A general term for a card, usually plastic, containing one or more semiconductor chips. A chip card can be either a → *memory card* or a → *microprocessor card*. In English-speaking countries, the term → *smart card* is generally used instead.

**Chip module** A carrier and package for a → *die*, with a set of contacts on its top surface. The short form ‘module’ is frequently used to refer to chip modules.

**Chip on tape (COT)** A packaging method in which → *chip modules* are arranged in adjacent pairs on a thin, flexible tape that is typically 35 mm wide.

**Chip size** The surface area of a chip, usually measured in square millimeters. Chip prices are largely dependent on the chip size. With the embedding methods currently used, the chip sizes of smart card microcontrollers can range from 3 to 25 mm<sup>2</sup>.

**CHV (cardholder verification)** → *PIN*

**CICC (contactless integrated chip card)** The standard ISO designation for a chip card that uses electromagnetic fields to transmit power and data without any contact with the card. The chip may be a memory chip (*memory card*) or a microcontroller chip (*processor card*).

**Circuit-switched** A form of data transmission for which a direct connection (a circuit) is established between two parties. The charge for this is usually based on the duration of the connection instead of the amount of data transferred as with a → *packet-switched* connection. Analog landline telephone connections are a typical example of circuit-switched data transmission.

**Class** In the context of → *object-oriented programming*, a class is a sort of abstract set of instructions for constructing an → *object*, or in other words, for constructing the → *attributes* and → *methods* of an object and its relationships to other objects.

<sup>18</sup> See also Section 11.6, ‘Commands for Authenticating Devices’, on page 374

**Class file** A file holding a compiled Java program (one that has been translated into → *bytecode*) along with supplementary information. After being loaded, the class file is executed by the → *Java Card virtual machine*.

**Cleanroom VM** → *Java Card virtual machine*

**Clearing** In an electronic payment system, the process of settling accounts between parties that accept electronic payments (usually merchants) and their bank.

**Clearing system** A computer-based → *background system* that performs centralized clearing in an electronic payment system.

**Client** A program that accesses data or other programs located on a central entity, which is called a → *server* and is usually located on a different computer than the client.

**Clone** → *cloning*

**Cloning** The process of making a complete copy of the → *ROM* and → *EEPROM* of a microcontroller. Cloning is a typical form of attack on smart card systems. It yields a → *microcontroller* whose memory contents are identical to those of the original device.

**Closed application** A smart card → *application* that is only available to the application operator and cannot be used for general purposes.

**Closed purse** An instance of a → *closed application* for an electronic purse system. A closed purse can be used only within the limits defined by the application operator, not for general payment transactions.

**CMM (capability maturity model)** An internationally used model for determining the maturity of software development. The maturity is determined using a standardized set of criteria and has five levels. The lowest maturity level (Level 1) designates a more or less chaotic development process, while the highest maturity level (Level 5) designates an orderly and continually self-improving development process.

**CoD (clear on deselect)** A property of a variable in Java Card that initiates clearing of the content of the variable when the applet is successfully deselected.

**Code density** A metric of the command set of a processor and the quality of the compiler. Code density is an important factor in systems with memory restrictions. It is rarely stated as an absolute value, but instead used to compare different processors and toolchains.

**CODEC (compressor/decompressor or coder/decoder)** A hardware chip or algorithm intended to be used for data compression and decompression or data encryption and decryption.

**Coercivity** Designates the strength of the magnetic field (measured in oersteds) necessary to write data to the → *magnetic stripe*. With low-coercivity ('loco') magnetic stripes, the required magnetic field strength is 300 to 650 oersted, while with high-coercivity ('hico') magnetic stripes the required magnetic field strength is 2750 to 4000 oersted.<sup>19</sup>

**Coil on chip** Designates a coil antenna located directly on a chip for a → *contactless smart card*.

<sup>19</sup> See also Section 2.2, 'Magnetic-stripe Cards', on page 16

**Cold reset** → *reset*

**Collision** A collision occurs when two or more contactless cards located within the active range of a terminal send data to the terminal at the same time, with the result that the received data cannot be decoded or unambiguously associated with a particular card.

**CombiCard** A registered trademark of ADE [ADE]; designates a → *dual-interface card*.

**Command** In the realm of → *smart card operating systems*, an instruction to the smart card to perform a specific action.<sup>20</sup> The result of a command is a → *response* returned by the smart card, which always contains status information and may contain data related to the executed command. Commands are transferred to the smart card in → *command APDUs*, while responses are transferred in → *response APDUs*.

**Command APDU** A → *command* sent from a terminal to a smart card, consisting of a command header and an optional command body.<sup>21</sup> The command header consists of a class byte, an instruction byte, and the two parameter bytes P1 and P2 (*APDU*).

**Commit buffer** In the Java Card context, an intermediate storage area used to implement → *atomic operations*.

**Common Criteria** (CC) The Common Criteria for Information Technology Security Evaluation<sup>22</sup> [CC] are a set of criteria for the development and → *evaluation* of information technology systems. They result from the refinement and harmonization of originally national and international criteria catalogs such as → *TCSEC* and → *ITSEC*. The Common Criteria were first published in 1996 by the → *NIST* and subsequently standardized internationally as ISO 15408. The Common Criteria define seven hierarchical levels of trustworthiness called → *evaluation assurance levels* (EALs).

**Compatibility** In general, the ability of different types of hardware, software and data to work together without any special adaptations, which does not mean that the various items are identical.

**Compiler** A program that translates a program written in a language such as Basic or C into machine language that can be executed directly by a microprocessor. After the compilation process, it is usually necessary to link the program code using a → *linker* program.

**Completion** The process of completing the → *operating system* by loading the EEPROM portion. This makes it possible to retrofit changes and adaptations without requiring the generation of a new → *ROM mask*. The same data is written to each smart card in the completion process, which means that it is similar in principle to → *initialization*.

**Contact field** An area with six or eight contacts located on the front of a → *smart card*, which forms the electrical interface between the → *terminal* and the → *microcontroller* in the smart card.<sup>23</sup> All electrical signal pass via these contacts.

**Contactless card** Abbreviated term for a type of → *smart card* with which power and data are transmitted using electromagnetic fields without any contact with the card (*CICC*).

<sup>20</sup> See also Chapter 11, ‘Smart Card Commands’, on page 353

<sup>21</sup> See also Section 8.3.1, ‘Command APDU structure’, on page 221

<sup>22</sup> See also Section 15.5, ‘Evaluation of Hardware and Software’, on page 659

<sup>23</sup> See also Section 3.2, ‘Contact Field’, on page 36

**Convention** In the smart card context, ‘convention’ refers to the order of the data bits and the coding of the signal levels for data transmission using the →  $T = 0$  or →  $T = 1$  protocol.

**CoR (clear on reset)** A property of a variable in Java Card that initiates clearing of the content of the variable when the smart card is reset.

**Core foil** Another term for → *interior foil*

**Core voltage** The operating voltage used directly on the chip of a microprocessor or microcontroller. If the core voltage is lower than the external voltage applied to the device, the external voltage must be suitably reduced by a voltage converter integrated in the chip. Low core voltages are necessary to compensate for reduced breakdown voltages due to small structure widths and to reduce the magnitude of capacitive currents at high clock rates. A microcontroller fabricated in 0.13- $\mu\text{m}$  technology typically has a core voltage of 1.8 V.

**COS (card operating system)** The common designation for a → *smart card operating system*. It often forms part of the product name of the operating system (e.g. → STARCOS, → TCOS, → SECCOS).

**CP8** Brand name of a multiapplication smart card operating system (*multiapplication smart card*) produced by Bull, which was available in several versions.

**CPU (central processing unit)** → *microprocessor*

**CRC (cyclic redundancy check)** A simple, widely used type of → *error detection code* for protecting data.<sup>24</sup> The initial value and divisor polynomial of the CRC method must be specified before it can be used.

**Credential** A data set that confirms the legitimation of an entity to perform a particular action. Some examples of credentials are a → *passport*, a secret → *key*, or the result of an → *authentication*.

**Credit card** A credit card<sup>25</sup> is a card, with or without a chip, that has an associated expenditure limit and with which payment occurs after the goods or services are received. This type of payment is called ‘buy now, pay later’. Embossed credit cards are typical examples of this type of card.

**Cryptoalgorithm** A condensed form of → *cryptographic algorithm*.

**Cryptocard** → *processor card*

**Cryptographic algorithm** A defined computational procedure with at least one secret parameter, the → *key*, that can be used to encrypt or decrypt data.<sup>26</sup> There are symmetric cryptographic algorithms (such as the → *DES* algorithm) that use the same key for encryption and decryption, as well as asymmetric cryptographic algorithms (such as the → *RSA* algorithm) that use a public key for encryption and a private (secret) key for decryption.

<sup>24</sup> See also Section 6.5.2, ‘CRC checksums’, on page 125

<sup>25</sup> See also Section 18.1.1.1, ‘Credit cards’, on page 748

<sup>26</sup> See also Section 7.1, ‘Cryptology’, on page 133

**Cryptoprocessor** In the smart card realm, a supplementary numerical processing unit in a microcontroller, optimized for the rapid computation of → *secret-key algorithms* such as → *DES* or → *AES* and/or public-key algorithms such as → *RSA*, → *DSA*, or → *ECC*.

**CT-API (Chipcard Terminal – Application Programming Interface)** CT-API<sup>27</sup> is an application-independent interface specification for connecting → *MKT*-compliant terminals with PCs. It is published by Teletrust Deutschland [Teletrust].

**Customer card** A → *smart card* used by a customer in an electronic payment system to make purchases using merchant terminals.

**D-AMPS** → *AMPS*

**Data protection** The technical and legal protection of data regarding the personal and material circumstances of individuals that can be associated with a particular person or persons.

**DEA (Data Encryption Algorithm)** Another name for → *DES*

**Deactivation sequence** Specifies the sequence of events for deactivating the electrical signals of a → *smart card microcontroller* in the process of powering down a → *smart card*. It does not say anything about the sequence of events for mechanical decontacting. The objective of the deactivation sequence is to protect the smart card microcontroller, which is sensitive to charges and voltages on its contacts. By contrast, the → *activation sequence* specifies the sequence of events for activating the signals on the contacts.

**Deanonymization** Reversing the process of → *anonymization*, such as by evaluating additional information in order to restore the link between the data and the original person.

**Debit card** A debit card<sup>28</sup> is a card with or without a chip that allows the cardholder to make expenditures up to a certain limit and with which payment occurs at the same time as the goods or services are received. For this purpose, the debit card is linked to a bank account so the purchase amount can be transferred directly at the time of payment. This form of payment is called ‘pay now’. A typical example of a debit card is the Eurocheque card.

**Debugging** Searching for and eliminating defects, with the objective of detecting and correcting as many defects in a software program as possible. Debugging is normally performed by software developers during → *implementation* and is not the same as testing (*test*).

**DECT (Digital Enhanced Cordless Telecommunication)** An → *ETSI* specification for cordless telephones operating in the 1.9-GHz band using → *cellular technology* and digital data transmission. Although the DECT specification has provisions for using smart cards in mobile stations, this is specified as optional, with the result that they are not used in practice.

**Defragmentation** The process of rearranging data storage in memory so that data originally located in several noncontiguous regions ultimately occupies a single contiguous memory region.<sup>29</sup> The essential parts of a defragmentation process must be performed

<sup>27</sup> See also Section 8.7.3, ‘MKT’, on page 241

<sup>28</sup> See also Section 18.1.1.2, ‘Debit cards’, on page 749

<sup>29</sup> See also Section 13.7.5, ‘Free memory management mechanisms’, on page 468

as atomic operations, as otherwise memory inconsistency can occur if the process is terminated prematurely.

**Delamination** The undesirable separation of foils that have been bonded together (laminated) using heat and pressure. Delamination of a card can for example be caused by using a nonthermoplastic ink (as typically used for offset printing) to print overly large areas between the → *core foil* and the → *overlay foils*.

**DEMA (differential electromagnetic analysis)** → *SEMA*

**Depersonalization** The process of undoing the electrical → *personalization* of a smart card.

If the → *smart card operating system* allows depersonalization, it may be performed using a special → *command* after prior authentication. One use for depersonalization is to restore incorrectly personalized cards to their original state so they can be reused.

**DER (Distinguished Encoding Rules)** → *BER*

**DES (Data Encryption Standard)** The DES algorithm<sup>30</sup> is the best known and mostly widely used symmetric → *cryptographic algorithm*. It was developed by IBM in collaboration with the → *NBS* and published in 1977 as a US standard (FIPS 46) called ‘Data Encryption Algorithm’ (DEA). The official successor to the DES is the → *AES*.<sup>31</sup>

**DESFire (Data Encryption Standard Fast Innovative Reliable and Secure)** The full name is Mifare DESFire (*Mifare*), which refers to a widely used, inexpensive memory chip made by NXP Semiconductors [NXP] and used in contactless communication systems. DESFire performs encryption and authentication using the → *triple DES* cryptographic algorithm and provides on-chip support for a variety of applications.

**Design** In the context of software development, the process of devising a software architecture based on the requirements defined in the → *analysis* phase. In simplified terms, the result of the design process is a description of how the requirements will be implemented in the software. In a sequential software development process, the subsequent phase is → *implementation*.

**Deterministic** Designates a method or algorithm that always produces the same result with a given set of initial conditions. It is the opposite of → *probabilistic*.

**DF (dedicated file)** A DF<sup>32</sup> is a directory in a smart card file system. The root directory (→ *MF*) is a special type of DF.

**DF name** A DF name<sup>33</sup> is a DF attribute with a length of 1 to 16 bytes, similar to the file identifier (→ *FID*). It is used to select the DF, and it may contain a registered application identifier (→ *AID*), which has a length of 5–16 bytes and makes the DF internationally unique.

**Die, dice** A die (plural dice) is a small, flat piece of crystalline silicon holding a single semiconductor integrated circuit (such as a microcontroller).

**Differential cryptanalysis** A computational method for determining the value of a secret key by using plaintext–ciphertext pairs having certain differences but the same key. The

<sup>30</sup> See also Section 7.1.1.1, ‘DES algorithm’, on page 138

<sup>31</sup> See also Section 7.1.1.2, ‘AES algorithm’, on page 140

<sup>32</sup> See also Section 12.3.2, ‘Dedicated file’, on page 424

<sup>33</sup> See also Section 12.5.3, ‘DF name’, on page 429

manner in which these differences propagate with further DES cycles is analyzed to determine the key. This method was published by Eli Biham and Adi Shamir in 1990.

**Differential fault analysis (DFA)** The operating principle of differential fault analysis<sup>34</sup> was first described in 1996 in a publication by Dan Boneh, Richard A. DeMillo and Richard J. Lipton [Boneh 96], all of whom were employees of Bellcore (*Bellcore attack*) at the time. The method is based on intentionally introducing scattered errors into a cryptographic computation in order to determine the secret key. Only public-key algorithms were mentioned in the original description of the method, but within a few months this form of attack was rapidly extended [Anderson 96a] with the result that all cryptographic algorithms are vulnerable to this form of attack if they do not employ protective measures.

**Digital** Refers to systems in which signals can assume only a limited number of values.

**Digital fingerprint** A commonly used designation for the hash value of a message (e.g. generated using → *SHA-1*).

**Digital signature** Digital signatures are used to establish the authenticity of electronic messages and documents. They are usually based on asymmetric cryptographic algorithms, such as the RSA algorithm. The validity of digital signatures is regulated by legislation in many countries, such as the Signaturgesetz (*Signature Act*) in Germany. Digital signatures are sometimes called ‘electronic signatures’.

**Digital watermark** A feature added to an imagery or audio file, ideally invisible or inaudible, that cannot be removed and is used to protect intellectual property rights. An analysis program can be used as necessary to check imagery or audio files for the presence of digital watermarks. Steganographic methods (*steganography*) are often used to generate digital watermarks.

**Directory service** A service that makes lists of certain data, stored in a database, available to requesters. Typical examples of such lists are → *certificate revocation lists* in a → *PKI*, which identify all certificates that are no longer valid or accepted.

**Disturbance test** (recovery test, tear-safe test) A test used to verify the preservation of data and program consistency when electrical power to a smart card is interrupted at an arbitrarily selected time. This consistency is implemented using the → *roll forward* and → *roll back* functions of the operating system.

**Divisor** A commonly used equivalent of clock rate conversion factor (CRCF),<sup>35</sup> which defines the length of the bit interval for data transmission in terms of the number of clock periods.

**Downlink** A connection from a higher-level system (such as a base station) to a lower-level system (such as a mobile station); the opposite of → *uplink*.

**Download** The process of transferring data from a higher-level system (such as a background system or host system) to a lower-level system (such as a terminal); the opposite of → *upload*.

<sup>34</sup> See also Section 16.5.1, ‘Attacks on the hardware’, on page 684

<sup>35</sup> See also Section 9.3, ‘ISO Transmission Protocols’, on page 254

**DPA (differential power analysis)** A form of attack<sup>36</sup> on → *smart cards* that represents an improvement on simple power analysis (*SPA*). It involves first making repeated measurements of the current consumption of a microcontroller for certain operations using known data with high time resolution and eliminating random noise by averaging. Following this, the current consumption is measured using unknown data, and deductions regarding the unknown data are made by analyzing the differences between the results with known and unknown data.

**DRAM (dynamic random-access memory)** A type of → *RAM* having a dynamic structure, which requires a continuous supply of power and periodic refreshing in order to retain its contents. DRAM cells are effectively capacitors. DRAM occupies less space on the chip than → *SRAM* and is thus less expensive, but SRAM has shorter access times.

**DSA (Digital Signature Algorithm)** An algorithm published in 1991 by the → *NIST* for use as a signature algorithm. In cryptographic terms, it is a variant of the El Gamal algorithm. The security of DSA is based on the discrete logarithm problem instead of the factorization problem as with → *RSA*. The intention with DSA was to create an algorithm that could not be used for encryption, but as early as 1993 it was shown that it can also be used to encrypt data.

**Dual-band mobile telephone** A mobile telephone that can operate in two different frequency bands, such as 900 and 1800 MHz.

**Dual-interface card** Designation for a → *smart card* having both contactless and contact interfaces for data transmission to and from the card.

**Dual-mode mobile telephone** A mobile telephone that can operate in two different mobile telecommunication systems, such as GSM and AMPS.

**Dual-rail bus** A bus in a smart card microcontroller on which all signals are transmitted in both directions.<sup>37</sup> This form of bus is used to defend against → *DPA* and → *SPA* attacks, since it prevents the determination of the data transmitted on the bus by measuring the differential current consumption of the microcontroller.

**Dual-slot mobile telephone** Designation for a mobile telephone with a second, externally accessible contact unit, usually for an ID-1 → *smart card*, in addition to the contact unit for the user card (such as a SIM). A dual-slot mobile telephone can for example be used with a smart card electronic purse to make payments via the mobile telecommunication system.

**Dual-slot solution** A smart card application based on using the second contact unit of a → *dual-slot mobile telephone*.

**Duplication** Transferring genuine data to a second card with the objective of producing one or more identical (cloned) cards. Generally synonymous with → *cloning*.

**Dynamic STK (dynamic SIM Application Toolkit)** Outdated term for a microbrowser (*browser*) that complies with the → *SIM Alliance* specification [SIMalliance].

---

<sup>36</sup> See also Section 16.5.1, ‘Attacks on the hardware’, on page 684

<sup>37</sup> See also Section 16.5.1, ‘Attacks on the hardware’, on page 684

**EAP** The Extensible Authentication Protocol (EAP), which is specified in RFC 3748, defines various mechanisms for the → *authentication* of two communication parties. It is very flexible and is used for authentication in applications such as PPP links, LANs and WANs. The term ‘EAP-SIM’ is used when a smart card is used to perform authentication with EAP. The EAP-SIM function is usually a → *smart card application* in a → *SIM* or → *USIM*.

**EAP-SIM** → *EAP*

**ECBS (European Committee for Banking Standards)** A European organization [ECBS] founded in 1992 with the task of developing technical solutions and standards for the infrastructure of → *interoperable* trans-European financial transaction systems.

**ECC (elliptic curve cryptosystem)** Designation for a cryptographic system (usually a cryptographic algorithm) based on elliptic curves.<sup>38</sup>

**e-Commerce (electronic commerce)** Refers to all forms of service, trade and associated financial transactions using public networks (primarily the Internet). The term → *m-commerce* is used when mobile terminals are used for e-commerce.

**EDGE (Enhanced Data Rates for GSM and TDMA Evolution)** An evolutionary GSM phase. The EDGE specification enables GSM mobile telephones to connect to base stations with a higher data transmission rate by using a different modulation scheme on the air interface, without any changes to the existing network infrastructure.

**EEPROM (electrically erasable programmable read-only memory)** A type of non-volatile memory used in → *smart cards*.<sup>39</sup> EEPROMs are partitioned into memory pages, and the page size is called the → *granularity*. The contents of a memory page can usually be changed bytewise but only erased as a block, and there is a physical upper limit to the number of write/erase cycles. The typical write/erase cycle time of EEPROM is 2.5 ms per page.

**EF (elementary file)** The actual data storage entity in a smart card file system. EFs are classified as working (for use by the terminal) or internal (for use by the smart card operating system), and they have an internal structure (transparent, linear fixed, linear variable, cyclic, BER-TLV, etc.).

**Electronic check** An → *electronic purse* variant with fixed, nondivisible monetary amounts, which is classified as ‘pay before’.

**Electronic purse (e-purse)** A card with a chip that must be loaded with an amount of money before it can be used to make purchases,<sup>40</sup> which is classified as ‘pay before’. Typical examples of electronic purses include the German GeldKarte, the Austrian Quick purse, Visa Cash, Proton, and Mondex. Some electronic purse systems support → *purse-to-purse transactions*.

**Embossing** Part of the physical personalization of a card, consisting of raised characters stamped into the plastic card body.<sup>41</sup>

<sup>38</sup> See also Section 7.1.2.4, ‘Elliptic curves as asymmetric cryptographic algorithms’, on page 152

<sup>39</sup> See also Section 5.3.3, ‘EEPROM’, on page 85

<sup>40</sup> See also Section 18.1.1.3, ‘Electronic purses’, on page 749

<sup>41</sup> See also Section 3.5.9, ‘Embossing’, on page 46

**Emulator** A hardware device that imitates the operation of another device or system (the target system). An emulator implemented in software is called a → *simulator*. Emulators are frequently used in the development of software for target systems that do not yet exist. A smart card emulator consists of circuitry that fully imitates the electrical and logical properties of a real smart card. As the majority of the functionality is implemented in hardware, emulators are usually faster (closer to real time) than simulators.

**EMV (Europay, MasterCard, Visa)** → *EMV specification*

**EMV specification** The EMV specification [EMV] is a joint specification for payment cards with chips and associated terminals, originally generated by Europay, MasterCard, Visa and American Express. It has achieved the status of a worldwide → *industry standard* for → *credit cards*, → *debit cards*, and → *electronic purses*.

**Endianness** Designates the order of the bytes in a byte string. Big-endian means that the most significant byte is at the beginning of the byte string, which consequently means that the least significant byte is at the end of the string. Little-endian designates the opposite order, with the least significant byte first and the most significant byte last.

**End-to-end connection** A direct communication link between two parties using the communication channels of one or more other entities that do not alter the information content of the actual data exchange. If communication between the two originating parties is cryptographically protected, the term → *tunneling* is used. A typical example of an end-to-end connection is direct communication between an application provider and a SIM compliant with TS 43.048.

**Enrollment** The process of originally acquiring the biometric data of a cardholder and entering it into the corresponding smart card. The data stored in the smart card forms the basis for subsequent biometric user identification.

**Envelope stuffing** The process of automatically folding letters, which may have attached smart cards, and inserting them into envelopes.

**EPROM (electrically programmable read-only memory)** A type of nonvolatile memory formerly used in smart cards, which has been supplanted by → *EEPROM*. An EPROM can only be erased by ultraviolet light, which means that in smart cards it can only be used for write-once, read-multiple (WORM) data storage.

**Error correction code (ECC)** A checksum calculated from a set of data to enable the detection and correction of errors in the data with a certain probability of successful error detection and correction.<sup>42</sup> A typical example of an ECC is the Reed–Solomon checksum.<sup>43</sup>

**Error counter** A counter that registers unsuccessful attempts to use a particular secret (PIN or key) and determines whether it can still be used. If the error counter reaches its maximum value, the secret is blocked and can no longer be used. The error counter is usually reset to zero when the action is successful.

<sup>42</sup> See also Section 6.5, ‘Error Detection and Correction Codes’, on page 122

<sup>43</sup> See also Section 6.5.3, ‘Reed–Solomon codes’, on page 127

**Error detection code (EDC)** A checksum calculated from a set of data to enable the detection of errors in the data with a certain probability of successful detection.<sup>44</sup> Typical examples of error detection codes are the XOR checksum and → *CRC checksum*.<sup>45</sup>

**ETSI (European telecommunication standard)** Designates standards issued by → *ETSI*, which are primarily related to European telecommunication systems.

**ETSI (European Telecommunication Standards Institute)** The standardization organization of the European telecommunication companies, responsible for defining standards for European telecommunication systems. The most important ETSI standards in the smart card realm are the families of standards for GSM (such as TS 51.011 for the SIM) and UMTS (such as TS 102 221 for the USIM application). Meetings of ETSI expert groups are usually held in a wide variety of scenic or popular locations in Europe and throughout the world, for which reason some people are convinced that the abbreviation ‘ETSI’ stands for ‘European travel and sightseeing institute’.

**etu (elementary time unit)** The duration of one bit in smart card data transmission. The length of the etu is not defined in absolute terms, but instead in terms of the frequency of the clock signal applied to the card and a divisor value. For example, the duration of the etu is 372 clock cycles for transmitting the → *ATR*, which corresponds to a data transmission rate of 9600 bit/s with a clock frequency of 3.5712 MHz.

**Eurosmart** An organization founded in 1994 to represent the interests of European smart card producers, with headquarters in Brussels. It is responsible for promoting and standardizing (*standard*) → *smart cards* and smart card systems, acting as a forum for exchanging market data and technical data, and forging links to national and international standardization committees.

**Evaluation** The unbiased, objective, repeatable, and reproducible assessment of an information technology system (hardware and/or software) by a trustworthy entity in accordance with the specifications of a criteria catalog.<sup>46</sup> The information technology system to be evaluated is called the → *target of evaluation*. The → *Common Criteria* (CC) are now used almost exclusively worldwide for the evaluation of → *smart cards* and smart card systems.

**Evaluation Assurance Level (EAL)** Designates a set of seven levels of trustworthiness of a → *target of evaluation* in the context of an evaluation in accordance with the → *Common Criteria*.<sup>47</sup> Level 0 is the lowest level of trustworthiness, while Level 7 is the highest.

**f1, f2, f3, f4, f5** (function 1 . . . function 5) Designations of cryptographic functions used in → *UMTS* for authenticating the network and the → *USIM* and for establishing cryptographically secured data transmission over the air interface. The kernel of these security functions is a symmetric → *cryptographic algorithm* that can be parameterized by supplementary linked input values. The USIM specification proposes the → *Milenage algorithm*, whose kernel is based on → *AES*, as an example algorithm for f1–f5.

<sup>44</sup> See also Section 6.5, ‘Error Detection and Correction Codes’, on page 122

<sup>45</sup> See also Section 6.5.2, ‘CRC checksum’, on page 125

<sup>46</sup> See also Section 15.5, ‘Evaluation of Hardware and Software’, on page 659

<sup>47</sup> See also Section 15.5, ‘Evaluation of Hardware and Software’, on page 659

**F2F (face-to-face) → face**

**Fab** Abbreviated term for a semiconductor fabrication plant.

**Face** The face of a semiconductor chip is the surface with the functional structures produced using semiconductor fabrication processes. Consequently, the expression ‘face-to-face contact’ means that two chips with suitably configured functional structures are placed together such that they are electrically connected to each other.

**FAT (file allocation table)** A table used in a certain type of file management system in which the data storage region to be managed is divided into sections called clusters. Data describing the occupancy and addresses of these clusters is stored and managed in the file allocation table.

**Fault tree analysis** A test method in which every execution path in the program code is traversed in order to search for possible faults.<sup>48</sup>

**FD/CDMA (frequency division / code division multiple access) → CDMA**

**FDMA (frequency division multiple access)** A → *multiple access method* for concurrent data transmission from several transmitters to a single receiver using several different frequency bands. Each transmitter is allocated a particular frequency band in the total available frequency spectrum, within which it can transmit exclusively. FDMA is used in mobile telephone systems such as the German C-Netz for the air interface between mobile stations and base stations.

**FeliCa (Felicity Card)** A Sony → *industry standard* for → *contactless smart cards*. FeliCa cards operate at the typical contactless smart card frequency of 13.56 MHz with a transmission rate of 212 kbit/s. However, FeliCa is not compatible with the ISO/IEC 14443 standard for contactless smart cards.

**FIB (focused ion beam)** A device that generates a focused beam of ions used to remove material from a semiconductor device or deposit material on a semiconductor device.

**FID (file identifier)** An FID<sup>49</sup> is a two-byte file identifier that can be used to select a file. The MF and each DF and EF have an FID. The FID of the MF is always ‘3F00’.

**File body → file header**

**File header** Files in smart cards are usually divided into two parts: a file header that holds information about the → *file structure* and the → *access conditions*, and a file body that holds the → *user data* and is linked to the file header by a pointer.

**File structure** The file structure designates the externally visible structure of an → *EF*.<sup>50</sup> File structures allow → *user data* to be stored in a compact, logically structured form. The standard file structures defined by ISO/IEC 7816-4 are transparent, linear fixed, linear variable, cyclic, and BER-TLV.

<sup>48</sup> See also Section 16.1.1, ‘Classification of attacks’, on page 669

<sup>49</sup> See also Section 12.5.1, ‘File identifier’, on page 426

<sup>50</sup> See also Chapter 12, ‘Smart Card File Management’, on page 421

**File type** The file type determines how a file is handled by the file management functions in the smart card<sup>51</sup> – i.e. whether it is a directory file (*MF*, *DF*, *ADF*) or a file that holds → *user data* (EF).

**FIPS (Federal Information Processing Standard)** Designates standards issued by the → *NIST*.

**Firewall** An entity (hardware or software) that implements a security barrier between individual → *applications* or other entities. For example, a firewall can separate two applications in a smart card so they cannot access each other's data across the firewall.

**Flash** A type of nonvolatile memory, more precisely designated flash EEPROM, used in → *smart cards* in place of mask-programmed → *ROM*.<sup>52</sup> Flash EEPROM is closely related to → *EEPROM* in its structure and operation, and it also has a physical upper limit to the number of write/erase cycles. Flash memory can be written bytewise but can be erased only pagewise or sectorwise. The write process takes only a few microseconds, but the erase process takes several milliseconds. NOR flash memory allows fully random access, while physically smaller NAND flash only allows blockwise access. Consequently, NAND flash is primarily suitable for mass storage and less suitable for holding program code.

**Flash EEPROM** Outmoded long form of → *Flash*.

**Flash-light attack** → *light attack*

**Floor limit** Defines the level above which a purchase must be authorized by a third party. Authorization is not required below the floor limit, but it must always be obtained above the floor limit, as otherwise payment may not be possible and is not guaranteed.

**Footprint** More precisely: memory footprint; designates the amount of memory needed for a particular purpose.

**Forensics** As a subdiscipline of forensics, the task of IT forensics is to investigate security-related incidents in connection with information technology systems by means of identification, analysis and evaluation of evidence located in computers.

**Form factor** A commonly used expression for the size of a → *smart card*.<sup>53</sup> The standard form factors are → *ID-1*, → *ID-000*, → *mini-UICC*, and → *Visa Mini*.

**Foundry** A semiconductor fabrication facility that operates on a contract basis and produces semiconductor devices developed by other parties.

**FPLMTS (Future Public Land Mobile Telecommunication Service)** → *IMT-2000*

**FRAM (ferroelectric random-access memory)** A type of nonvolatile memory that has been used only rarely in → *smart cards* up to now. Data storage in this type of memory is based on the properties of a ferromagnetic substance located between a control gate and a floating gate. FRAM cells typically have a write time of 100 ns per memory page and do not require a special erase voltage. However, the number of read cycles is limited, and FRAM fabrication involves processes that are difficult to master.

<sup>51</sup> See also Section 12.3, 'File Types', on page 423

<sup>52</sup> See also Section 5.3.4, 'Flash memory', on page 90

<sup>53</sup> See also Section 3.1, 'Card Formats', on page 29

**Frame** A sequence of data bits and optional error detection bits bounded by frame delimiters. Frames for contactless data transmission with smart cards are defined in ISO/IEC 14433.

**FTL (flash translation layer)** A software layer above the actual memory hardware; used to manage defective memory blocks in NAND → *flash*.

**Full duplex** Designates a type of data transmission in which the two communicating devices can send and receive concurrently. The other two types are → *half-duplex* and → *simplex*.

**Garbage collection** A function that collects memory no longer used by an → *application* and makes it available as free memory.<sup>54</sup> Garbage collection was formerly implemented as an interrupt-driven function. In modern computer systems, garbage collection is a low-priority thread that constantly searches memory for regions that are no longer needed and returns them to the free memory pool.

**GeldKarte** Brand name of an electronic purse system launched in Germany in 1996. It designates the application in a → *multiapplication smart card* as well as the → *smart card* itself. The smart card operating system used for GeldKarte or debit card functionality is → *SECCOS*.

**Glitch** A very short voltage dropout or voltage spike.

**Glob top** Designation for an encapsulation material, usually epoxy resin, used to protect a chip and its bonding wires in a → *module*.

**Global Platform (GP)** An international association founded in 1999 by various smart card companies to standardize technologies for → *multiapplication smart cards*. The most important specification published by Global Platform is the Global Platform (GP) specification. Among other things, this specification encompasses downloading smart card applications, securing the application → *life cycle*, and linking smart card applications to the smart card operating system. The GP specification is the de facto worldwide standard for → *multiapplication smart cards* and application management.

**Good case** The case in which a logical decision yields a favorable or intended result.

**GPRS (General Packet Radio System)** A → *GSM* extension, standardized by → *ETSI*, for achieving higher data transmission rates with mobile telephones. It provides a packet-switched connection by bundling existing time slots. A mobile telephone with GPRS technology is constantly connected to the network for data transfer and thus always available for data transmission. The data transmission rate is dynamically adapted to the currently required capacity, so only the capacity actually needed is used. This makes GPRS very suitable for discontinuous data transfers.

**Granularity** A frequently used alternative term for the page size of → *EEPROM*. For example, an EEPROM with a granularity of 32 has a page size of 32 bytes. This is independent of the size of the sector, which contains several pages and typically has a size of 2 to 8 kilobytes and can be erased as a block.

<sup>54</sup> See also Section 13.7.5, ‘Free memory management mechanisms’, on page 468

**Greybox test** A mixed form of test intermediate between a → *whitebox test* and → *black-box test*.<sup>55</sup> With a greybox test, the party performing the test knows some but not all of the internal processes, functions and mechanisms of the software under test.

**Greylist** A database list of all → *smart cards* or devices that are under observation. (*black-list, hotlist, whitelist*)

**GSM (Global System for Mobile Communication)** A digital, cellular, interoperable, international second-generation terrestrial (→2G) mobile telecommunication system. The frequency bands allocated to this system are 900 MHz (GSM 900), 1800 MHz (GSM 1800), and 1900 MHz (GSM 1900). The GSM system is defined by a family of specifications published by → *ETSI*. The major network operators and manufacturers are allied in the → *GSM Association*. Originally, GSM was only intended to be used in certain central European countries as a successor to national analog mobile telephone systems, but it has developed into an international standard for mobile telecommunication systems. The designated successor to GSM is → *UMTS*.

**GSM Association** The GSM Association [GSM Association] is an international body for the coordination of mobile telecommunication systems, with offices in Dublin and London. It was founded in Copenhagen in 1987, and it is responsible for the development and use of the → *GSM* standards. The GSM Association represents more than 700 network operators, manufacturers, and suppliers in the GSM sector.

**Guilloche patterns** Decorative patterns consisting of very fine interwoven lines, usually circular or oval, such as are found on many types of banknotes and share certificates. Due to their fine structures, these patterns can only be reproduced by high-quality printing techniques, so they are difficult to copy.

**HAL (hardware abstraction layer)** An intermediate layer in an operating system that conceals all hardware-specific features of the target platform from the rest of the operating system. The objective of this is to simplify the porting of the operating system, since changing the hardware platform only requires modifications within the HAL.

**Half duplex** A data transmission method in which the two communicating parties cannot send and receive data concurrently. A → *full duplex* connection is required for concurrent data transmission and reception.

**Handle** An intelligent pointer that, unlike pointers that reference data objects directly, points to a data object by means of a named reference. This allows the data object in memory to be relocated by an external party without invalidating the reference.

**Handover** In a mobile telecommunication network, the interruption-free transfer of a mobile telephone from one cell to the next. Handovers are always initiated by the network in GS systems.

**Happy-day scenario** Another term for → *good case*.

**Hard mask** This means that most of the → *program code* is held in ROM (*ROM mask*), which saves space compared with a → *soft mask* because ROM cells are significantly

<sup>55</sup> See also Section 15.4.2.5, ‘Greybox test’, on page 653

smaller than EEPROM or flash cells. However, it entails the full duration of the production process for customer-specific semiconductor devices, so the production lead time with a hard mask is significantly longer than with a soft mask. Hard masks are normally used to produce large quantities of smart card chips with largely identical functionality. The opposite of a hard mask is a soft mask, which means that major functions are stored in EEPROM or flash memory.

**Hash function** An algorithm for compressing data using a → *one-way function* so that it is not possible to compute the original data by reversing the process. A hash function produces a fixed-length result from an input of any arbitrary length, and it is designed so that any change to the input data has a very high probability of affecting the computed hash value (output value). → *SHA-1* is a typical example of a hash algorithm. The result of a hash function is a hash value, which is often also called a digital fingerprint.

**HBCI (Home Banking Computer Interface)** A standard defined by the German banking industry for the implementation of home banking in Germany, optionally with smart card support.

**Hologram** A photographic image produced using holographic techniques. It is a three-dimensional image of the photographed object. The object in the hologram can thus be seen from different angles, depending on the viewing angle of the observer. The holograms normally used with smart cards are embossed holograms, which produce reasonably satisfactory three-dimensional images under normal lighting conditions.

**Home net** The mobile telecommunication network of the operator with which a mobile telephone user has a contractual relationship (subscription).

**Home zone** In a mobile telecommunication system, a → *location-based service* with which calls are charged at a significantly lower rate (normally the landline rate) inside a certain region (usually the immediate vicinity of the subscriber's residence). As a result, the subscriber does not need to have a connection to the regular telephone network.

**Horizontal prototype** → *prototype*

**Hotlist** A database list of → *smart cards* and devices that probably have been manipulated and must not be accepted under any circumstances. (*blacklist*, *greylist*, *whitelist*)

**HSCSD (high-speed circuit-switched data)** An extension to the GSM standard for boosting the data transmission rate over the air interface up to a theoretical value of 76 800 bit/s ( $8 \times 9600$  bit/s) for uplink or downlink by supplementary use of existing time slots. Existing GSM networks can be extended to support HSCSD at a relatively low cost by upgrading the base stations and using special mobile telephones. The drawback is that the demand for transmission channels can increase by as much as a factor of eight.

**HSM (hardware security module / host security module)** Another name for → *security module*.

**HTML (Hypertext Markup Language)** A logical markup language conceptually based on → *XML* and used for → *hypertext* documents on the World Wide Web (→ *WWW*) (→ *WML*).

**HTTP (Hypertext Transfer Protocol)** A stateless protocol typically used to transfer → *HTML* pages on the Internet. The underlying transport protocol is usually → *TCP/IP*.

**Hybrid card** Designation for a card with two different card technologies. Typical examples are cards with a → *magnetic stripe* as well as a chip and → *smart cards* with optical storage on the card surface.

**Hypertext** Unlike normal text, hypertext contains cross-references (hyperlinks) to other locations in the text or to other documents. These hyperlinks can be invoked by a suitable user action, such as clicking the link. As opposed to normal linearly structured text, such as text in books, hypertext allows any desired interlinking of documents to be achieved by using cross-references. Typical examples of markup languages for hypertext documents are *HTML* and → *WML*.

**ICAO (International Civil Aviation Organization)** A United Nations organization [ICAO] that is responsible for many standards in the aviation industry. It was founded in 1944 and is based in Montreal, Canada.

**ICC (integrated chip card)** Standard → *ISO* designation for a card with a chip, usually in → *ID-1* or → *ID-000* format. The chip may be a memory chip or a microcontroller chip.

**ID-000** A → *smart card* format specified in TS 51.000, which is the most widely used format in the mobile telecommunication sector.<sup>56</sup> It is also called ‘plug-in card’ format. An ID-000 card measures 15 by 25 mm with a thickness of 0.76 mm. It can be produced by stamping from a standard → *ID-1* card. A smart card in → *mini-UICC* format can also be produced by stamping from an ID-000 card.

**ID-1** A format specified in ISO 7810 as the standard format for → *smart cards*.<sup>57</sup> It measures 54 by 85.6 mm with a thickness of 0.76 mm. However, the → *ID-000* format is now used exclusively in the mobile telecommunication sector.

**ID (identification document)** A document issued by a government organization or private organization that establishes the identity of its holder, with a claim to binding force. A special form of ID is a → *passport*, which is primarily intended to facilitate international travel.

**ID card** A card, with or without a chip, used to establish the identity of a person.<sup>58</sup>

**Identification** A process that results in the recognition of an entity.<sup>59</sup> Identification can be achieved by means of authentication.

**IEC (International Electrotechnical Commission)** An organization founded in 1906 for the purpose of generating international standards for electrical and electronics technology. It is based in Geneva, Switzerland [IEC].

**IFD (interface device)** The standard ISO designation for a smart card terminal.

**I<sup>2</sup>C interface** An interface specified by Philips that employs two signal lines between the interconnected devices, such as an → *NFC* chip and a security chip in a → *smart card*.

<sup>56</sup> See also Section 3.1, ‘Card Formats’, on page 29

<sup>57</sup> See also Section 3.1, ‘Card Formats’, on page 29

<sup>58</sup> See also Chapter 22, ‘Smart Cards for Identification and Passports’, on page 893

<sup>59</sup> See also Section 7.8, ‘Identification of Persons’, on page 187

**Implanter** A production machine for → *smart cards* whose function is to insert → *modules* in the cavities of → *card bodies*.

**Implementation** In the context of software development, the process of producing a program based on the software architecture defined in the → *design* phase. Implementation also includes → *debugging* but not testing (→ *test*), which occurs in the subsequent and final phase of sequential software development.

**IMSI catcher** A device that intercepts GSM conversations by establishing its own cell. An IMSI catcher interposes itself between a mobile station and a base station. It purports to be a base station with respect to the mobile station and a mobile station with respect to the base station.

**IMT-2000 (International Mobile Telecommunication 2000)** A concept developed by the → *ITU* for third-generation(3G) mobile telecommunication systems operating in the 2000-MHz frequency band. IMT-2000 was developed in 1995 as a successor to the Future Public Land Mobile Telecommunication Service (FPLMTS), a mobile telecommunication concept initiated by the ITU in 1985 that failed to achieve international acceptance in its original form. Typical implementations of IMT-2000 are → *UMTS* and → *CDMA 2000*.

**Indexed TAN** → *transaction number*

**Individualization** → *personalization*

**Industry standard** A standard or specification that is typically generated and published by a company or a commercial organization instead of an international standardization organization.

**Initialization** The process of loading the fixed and person-independent data of an → *application* into EEPROM.<sup>60</sup> This process is also called ‘prepersonalization’.

**Initializer** An entity that performs → *initialization*.

**Instrumentation** Introducing special → *program code* in a program to allow the processes and calls of the program to be analyzed for test purposes.

**Intelligent memory card** A memory card with additional logic circuitry for supplementary security functions that monitor memory accesses.

**Internal foil** A foil located inside the stack of foils laminated together to produce a → *card body*; synonymous with ‘core foil’. An internal foil is usually laminated between two → *overlay foils* to form the card body. The internal foil often forms the substrate for security features or electrical components, such as the antenna coil of a contactless smart card.

**Internet smart card** A → *smart card* with a → *smart card operating system* that supports typical Internet technologies such as the → *TCP/IP protocol*, secure communication using → *SSL* and → *TSL*, and an HTTP server (smart card web server (SCWS)). It often also has a USB interface and operating system support for multitasking and multithreading. Internet smart cards are sometimes called ‘network cards’ because they can act as independent devices on the Internet.

<sup>60</sup> See also Section 14.7.3, ‘Initializing the application’, on page 612

**Interoperability** In the smart card world, an attribute of a system or solution that is not tailored to a particular smart card → *application* or the equipment of a particular manufacturer. As a rule, → *open smart card operating systems* are interoperable. The opposite of an interoperable solution is a → *proprietary* solution. An example of an interoperable smart card is the → *SIM*, which can be used in all types of → *GSM* mobile telephones without compatibility problems.

**Interpreter** A program that translates the instructions of a programming language such as → *Basic* or → *Java* into machine code that can be executed by a microprocessor, with each instruction being executed immediately after it has been translated. Interpreted programs always run more slowly than compiled → *program code* because the translation takes place at run time. However, programs with significantly greater hardware independence can be produced using an interpreted language.

**Interpreter-based operating system** A → *smart card operating system* that runs → *applications* whose program code is not in the machine language of the target processor. Interpreter-based operating systems accordingly have an → *interpreter* or a → *compiler* to assist in translating the programs into the machine language of the target processor.

**ISDN (Integrated Services Digital Network)** Designation for an internationally standardized digital telephone network that supports voice communication as well as data transmission. An ISDN connection consists of two base channels, each with a transmission rate of 64 kbit/s, and a control channel with a transmission rate of 16 kbit/s.

**ISO (International Organization for Standardization)** ISO was founded in 1947 and is based in Geneva, Switzerland. Its function is to support the generation of international standards in order to promote the free exchange of goods and services. The first ISO standard was published in 1951 and deals with temperature effects on length measurements.

**ISO file system** Designates a smart card file system compliant with ISO/IEC 7816-4.

**ISO protocols** In the → *smart card* domain, the → *T = 0* and → *T = 1* transmission protocols, which for a decade were the only protocols specified by the ISO/IEC 7816 standard.

**ITSEC (Information Technique System Evaluation Criteria)** A catalog of criteria for the development and → *evaluation* of the security of information technology systems in Europe, published in 1991. The → *Common Criteria* emerged as the result of further refinement of the ITSEC and harmonization with various sets of national criteria.

**ITU (International Telecommunication Union)** The ITU [ITU] is an international organization dedicated to the coordination, standardization, and development of international telephone services, based in Geneva. It is the successor to the → *CCITT*.

**Jative card** A → *Java card* without any function for loading → *applets*, which means that it has the same characteristics as a → *native card* with respect to the outside world. Jative cards are used predominantly in situations with very restrictive memory size requirements, where they have the advantage of providing the functional attributes of regular Java cards without their elaborate functions for loading applets. They enable card producers to match the small memory → *footprints* of → *native smart card operating systems* while working with a → *Java operating system*.

**Java** A hardware-independent, object-oriented programming language (→ *object-oriented programming*) developed by the Sun Corporation, which is widely used in the Internet world. Java source code is translated by a compiler into standardized → *bytecode*, which is usually interpreted by a virtual machine running on a platform consisting of the target hardware (Intel, Motorola, etc.) and operating system (Windows, Mac OS, Unix, etc.). There are also special microprocessors (such as picoJava) that can execute Java bytecode directly.

**Java Card API** An API that enables Java applications in a Java card to access a large number of methods in order to implement the required functionality.

**Java Card Forum (JCF)** Java Card Forum [JCF] is an international organization that was founded by several smart card companies in 1997 to promote → *Java Card* technology and develop related specifications.

**Java Card runtime environment (JCRE)** The Java Card runtime environment essentially consists of the → *Java Card virtual machine* (JCVM) and the → *Java Card API*.

**Java card** A → *smart card* with a → *microcontroller* that hosts a → *Java Card virtual machine* and a → *Java Card runtime environment*.<sup>61</sup> Java cards are → *multiapplication smart cards* that can manage and run programs written in Java. Strictly speaking, Java Card is not a true → *smart card operating system*, in part because its original specification does not include file management functions. However, in practice Java Card is regarded as the archetype of an → *open smart card operating system*.

**Java Card virtual machine (JCVM)** A Java Card → *virtual machine* is a simulated → *microprocessor* (usually implemented in software) whose function is to execute Java → *bytecode* and manage Java classes and objects. The Java Card virtual machine also assures → *application separation* by means of firewalls and enables shared use of data. In principle, a Java Card virtual machine can be regarded as a sort of → *interpreter*. A Java Card VM implemented using publicly accessible information, i.e. without using additional information subject to a licensing agreement with Sun, is called a ‘cleanroom VM’. Cleanroom implementations of the Java VM are generally considered to be free of any obligation to pay license fees to Sun.

**Java Development Kit (JDK)** A set of software tools that facilitate the development of programs written in → *Java*.

**JCOP** An operating system for Java smart cards that is used worldwide. It was originally developed by IBM and subsequently taken over by NXP.

**Kerckhoff's principle** A principle named after August Kerckhoff (1835–1903), which states that the entire security of a cryptographic algorithm should be based exclusively on the confidentiality of its key, rather than the confidentiality of the algorithm.<sup>62</sup>

**Kernel** The central part of an → *operating system*, which provides basic system functions to the overlying layers of the operating system.

<sup>61</sup> See also Section 13.16.2, ‘Java Card’, on page 499

<sup>62</sup> See also Section 7.1, ‘Cryptology’, on page 133

**Key** In the case of a → *cryptographic algorithm*, the parameter that individualizes the encryption or decryption process. When a symmetric cryptographic algorithm is used to ensure security, the key must be secret, but the public key of an asymmetric cryptographic algorithm may be generally known.

**Key management** A collective term for all management functions associated with generating, distributing, storing, updating, revoking, and accessing cryptographic keys.

**Kinegram** A kinegram shows different images when viewed at different angles.<sup>63</sup> In terms of its structure, a kinegram is form of → *hologram*. A kinegram can display related images that appear to shift suddenly, or it can display different images at different viewing angles. Kinegrams are similar to holograms, which show three-dimensional images, but not identical to them.

**Lamination** The process of bonding thin sheets of material by subjecting them to heat and pressure. Cards are often laminated from several plastic foils.

**Laser cutter** A device that uses a high-energy laser beam for drilling and cutting with a precision of a fraction of a micrometer, preferably on semiconductor chips. Laser cutters can also be used to carry out → *light attacks* on semiconductor devices.

**Laser engraving** A process for blackening special plastic layers by charring them with a laser beam. This is also colloquially called ‘lasing’.

**Lead time** In semiconductor fabrication, the elapsed time between the provision of the mask (→ *ROM mask*) and the availability of the first samples.

**Lead-frame module** A type of low-cost module with contacts stamped from a sheet of copper alloy electroplated with gold and held together by a plastic mold body. A chip is placed on the lead-frame module by a pick-and-place robot and wire-bonded to the backs of the leads. After this, the chip is covered by a glob of opaque epoxy resin for protection. Lead-frame technology is currently one of the least expensive ways to produce chip modules, with no penalties in terms of the mechanical robustness of the modules.

**Life cycle** The aggregate of the phases in the life of a → *smart card*, beginning with the production of the chip and the card, progressing through → *personalization* and use, and terminating with the logical or physical end of the card’s life. The individual phases of the smart card life cycle are used to define specific security measures and functionalities. An example of life cycle management for cards is the → *Global Platform* specification.

**Life cycle model** A life cycle model (also called a process model) specifies in abstract form the organizational context, work processes, and activities of a development process, along with the associated conditions and results. Its objective is to achieve a general-purpose, uniform working method for development projects. Some examples of life cycle models are the waterfall model, the V model, and the cyclical development model.

**Light attack** A form of attack, also known as optical fault induction attack, in which a → *microcontroller* is exposed to brief, high-intensity light pulses focused on a particular

<sup>63</sup> See also Section 3.5.7, ‘Kinegram’, on page 45

point.<sup>64</sup> A → *laser cutter* can be used for this purpose. The intense energy can cause the microcontroller to malfunction, which can be utilized for attacks on the microcontroller.

**Linker** A program that converts the symbolic memory addresses of compiled or assembled → *program code* into absolute or relative memory addresses.

**Little endian** → *endianness*

**Load agent** An entity that loads electronic money into an electronic purse; functionally the opposite of a service provider.

**Loader** A program that can be used to load other programs, for example via a serial interface (→ *boot loader*).

**Location-based services** Value-added services for mobile telephone subscribers that are based on knowledge of the subscriber's current geographic position, such as local weather forecasts, city maps dynamically adapted to the user's current location, and integrated location data for emergency services calls.

**Logical channel** Logical channels allow data to be exchanged concurrently and independently with several → *applications* in a → *smart card*.<sup>65</sup> Although communication with the smart card still takes place via the single serial interface of the card, the logical channels allow → *APDUs* to be sent directly to specific applications in the smart card.

**M/Chip** Designation of an → *EMV*-compliant implementation of a chip-based debit/credit card issued by MasterCard. The M/Chip Select version uses both symmetric and asymmetric cryptographic algorithms and is a superset of M/Chip, which is a simplified version that uses only symmetric cryptographic algorithms.

**M2M (machine to machine)** A collective term for applications in which smart card microcontrollers are used in connection with automated equipment, telemetry equipment, and the broad field of applications in the automotive sector. The functionality and safety requirements on the microcontrollers usually remain unchanged in such applications, but the operating temperature range is significantly larger.

**MAC (message authentication code)** A cryptographic checksum for data that allows manipulation of the data during transmission to be detected. An equivalent checksum used to protect stored data is called a → *cryptographic checksum* (CCS).

**Magnetic card** A commonly used short form of → *magnetic-stripe card*.

**Magnetic stripe** A plastic stripe with a surface layer consisting of a magnetic metal oxide, which is laminated over the full length of a card.<sup>66</sup> Magnetic stripes have two or three data tracks and are classified as low-coercivity, medium-coercivity or high-coercivity, depending on the strength of the magnetic field necessary to erase the data they hold (→ *coercivity*).

**Magnetic-stripe card** A card with a magnetic stripe on which data can be written and subsequently read. The magnetic stripe usually has three data tracks with different data

<sup>64</sup> See also Section 16.5.1, 'Attacks on the hardware', on page 684

<sup>65</sup> See also Section 8.5, 'Logical Channels', on page 233

<sup>66</sup> See also Section 2.2, 'Magnetic-stripe Cards', on page 16

recording densities. Tracks 1 and 2 are used only for reading after the card has been issued, but data may also be written to track 3 during normal use. The magnetic substance in the stripe may have either a high-coercivity characteristic or a low-coercivity characteristic.

**Management data** Data used only for the management of → *user data*, which is otherwise of no significance to an → *application*

**Maosco** → *Multos*

**Mask** An abbreviated form of → *ROM mask*.

**m-Commerce** (mobile commerce) A collective term for all types of services, commercial transactions and associated payment transactions using mobile equipment such as mobile telephones or PDAs. If fixed equipment is used, the designation → *e-commerce* is used instead.

**Memory card** A card with a chip having simple logic circuitry and memory that can be read and/or written. Memory cards may also have supplementary security units, which for example support authentication.

**Memory footprint** Designates the logical memory structure of a computer system

**Memory protection** A function of a multitasking system in which programs running in parallel are located in closed, protected memory spaces and must use special mechanisms to exchange data with each other.

**Merchant card** In an electronic payment system, a → *smart card* located in a merchant terminal that acts as a security module.

**Method** In the context of → *object-oriented programming*, a function provided by the → *class* of an → *object* that can be used to manipulate the → *attributes* of the object.

**MexE (mobile station execution environment)** A framework for integrating a Java virtual machine (JVM) in a mobile telephone. It can be used to load and run Java programs in the mobile telephone, which allows value-added services to be implemented directly in the mobile telephone instead of in the SIM.

**MF (master file)** A special type of DF in a smart card file system.<sup>67</sup> It is the root directory of the file tree and is automatically selected after the smart card is reset.

**Microbrowser** → *browser*

**Microcontroller** A microcontroller<sup>68</sup> integrates a → *microprocessor*, volatile memory (→ *RAM*), nonvolatile memory (→ *ROM*, → *EEPROM* or → *flash*), and suitable interfaces for external communication in a single chip. It is thus a self-contained, fully functional computer on a single chip. In addition to smart cards, microcontrollers are widely used in control systems.

**Microprocessor** The primary functional unit of a → *microcontroller*. The microprocessor resolves the machine instructions of the program code into microinstructions and

<sup>67</sup> See also Section 12.3.2, ‘Dedicated file’, on page 424

<sup>68</sup> See also Chapter 5, ‘Smart Card Microcontrollers’, on page 73

executes the microinstructions. A microprocessor contains all the registers necessary for instruction processing, control logic, and a processing unit. The processing unit of a microprocessor is sometimes simply called the processor. The term ‘central processing unit’ (CPU) is often used as a synonym for ‘microprocessor’.

**Midlet** A program written in the → *Java* programming language and executed by the virtual machine of a computer. Midlets typically run in mobile equipment such as mobile telephones.

**Mifare** A widely used → *standard* for → *contactless cards*, originating from NXP Semiconductors [[NXP]]. Mifare [Mifare] is compatible with ISO/IEC 14443 Type A.

**Milenage algorithm** A symmetric algorithm suggested as an example algorithm for the f1–f5 (→ *fI*) functions of → *USIM*. The kernel of the Milenage algorithm is based on the → *AES*.

**Mini-UICC** Designation of a → *smart card* format used in the mobile telecommunication sector.<sup>69</sup> This format, which was originally known as 3FF (third form factor), measures 12 by 15 mm with a thickness of 0.76 mm. A mini-UICC card can be produced by stamping from an → *ID-1* or → *ID-000* card.

**MKT (Multifunktionales Kartenterminal)** The MKT specification<sup>70</sup> is widely used in Germany as a specification for multipurpose smart card → *terminals* and their connection to PCs in accordance with the → *CT-API* interface specification. It supports → *memory cards* as well as → *processor cards*. The MKT specification is published by Teletrust Deutschland [Teletrust].

**MMU (memory management unit)** A memory protection device that supports virtual addressing.

**Module** → *chip module*

**Module producer** An entity that packages dice in → *modules* and electrically connects the dice to the module → *contacts* (usually by means of → *bonding*).

**Mondex** A smart card → *electronic purse* system that supports → *purse-to-purse transactions* [Mondex].

**Mono-application smart card** A smart card containing only one → *application*.

**Monofunctional smart card** A → *processor card* whose → *operating system* supports only one → *application*, and which may be optimized for this application. Monofunctional smart cards often provide only limited support for management functions such as file creation and deletion.

**Monolayer card** A card composed of only one layer of plastic foil, as opposed to a → *multilayer card*.

**MoU (Memorandum of Understanding)** The common legal basis for all GSM network operators. The organization behind the MoU is the → *GSM Association*.

<sup>69</sup> See also Section 3.1, ‘Card Formats’, on page 29

<sup>70</sup> See also Section 8.7.3, ‘MKT’, on page 241

**MPU (memory protection unit)** A device that provide additional access protection for memory regions of a smart card microcontroller. It is used to isolate individual applications.

**Multiapplication smart card** A smart card containing several → *applications*, such as a bank card with a supplementary electronic ticket application.

**Multiprint sheet** In printing, a set of relatively small items (such as cards) printed on a single large sheet, which is separated into individual items after printing. This allows the printing process to be optimized, since many items can be printed in one pass on a large sheet instead of using several separate passes. For example, a typical multiprint sheet for card printing holds 42 cards on a large plastic sheet.

**Multifunctional card (MFC)** A → *processor card* that supports multiple → *applications* and has suitable management functions for loading and deleting applications and files.

**Multilayer card** A card composed of several layers, consisting of overlay foils (outer layers) and one or more core foils (internal layers), as opposed to a → *monolayer card*.

**Multiple access methods** Methods used in radio systems and information technology systems to enable as many users as possible to utilize a limited frequency bandwidth for concurrent or quasi-concurrent transmission. The four commonly used methods are frequency division multiple access (→ *FDMA*), time division multiple access (→ *TDMA*), code division multiple access (→ *CDMA*), and space division multiple access (→ *SDMA*).

**Multiproject wafer** A semiconductor wafer on which different types of microcontroller chips with different ROM masks can be produced.

**Multitasking** A computer system that supports multitasking allows several programs to run quasi-concurrently. Each of the concurrently running programs is usually located in a separate address space that is protected against access by other programs, and it can exchange data with other programs only by means of special mechanisms. Multitasking is not the same as multithreading, in which a single program performs several different tasks quasi-concurrently. A computer system may support both multitasking and multithreading.

**Multithreading** A computer system that supports multithreading allows a single program to perform several different tasks quasi-concurrently. The individual threads of a program normally use a common address space. Multithreading is not the same as multitasking, in which several different programs run concurrently, each with its own separate address space. A computer system may support both multitasking and multithreading.

**Multos (Multiapplication Operating System)** Brand name of an open, multiapplication → *smart card operating system* (→ *open smart card operating system*). The publisher of the specifications, license issuer and operator of the certification services required for Multos is the Maosco Consortium [Maosco].

**Name space** A set of names in which all of the names are unique.

**NAND flash** → *flash*

**Native card** A smart card with a → *native smart card operating system*.

**Native code** A program whose instructions are in the specific machine language of the microprocessor that executes the program.

**Native smart card operating system** A → *smart card operating system* that runs in the machine language of the target processor. Program-based → *applications* that run under a native operating system also employ the machine language of the target processor. Native operating systems thus do not have an → *interpreter* or a → *compiler* to translate programs into the machine language of the target processor, as opposed to → *interpreter-based operating systems*.

**NBS (National Bureau of Standards)** The name of the → *NIST* prior to 1988.

**NCSC (National Computer Security Center)** The NCSC [NCSC] is a subagency of the US National Security Agency (→ *NSA*). It is responsible for evaluating security products and publishing criteria for secure computer systems, such as the TCSEC.

**Negative file** → *blacklist*

**Network card** → *Internet smart card*

**NFC (near field communication)** A contactless (RF) technology for use in the range up to approximately 20 cm, originally developed by Philips and Sony and now standardized by ISO 18092 and other standards.<sup>71</sup> The NFC Forum [NFC], founded in 2004, is an association of companies with an interest in NFC whose mission is to promote and standardize NFC technology. NFC operates in the 13.56-MHz frequency band with low transmit power and supports data transfer rates of 106, 212, and 424 kbit/s. Communication is always → *half duplex*, with communication link setup times much shorter than with Bluetooth or WLAN. NFC is exclusively intended to support communication between two devices. One of the two devices must actively transmit, while the other device may operate passively (passive mode) as in → *RFID* systems. However, it is also possible for both devices to participate actively in the communication process (active mode).

**Nibble** The four most significant or four least significant bits of a byte; also called a half-byte.

**NIST (National Institute of Standards and Technology)** The NIST [NIST] is a division of the US Department of Commerce, responsible for generating US standards for information technology. The NIST, which was called the NBS until 1988, publishes the FIPS standards.

**Noiseless** A property of a → *cryptographic algorithm* that always requires the same amount of time to encrypt or decrypt data, regardless of the → *key*, plaintext, and ciphertext involved. If a cryptographic algorithm is not noiseless, the size of the key space can be markedly reduced by analyzing the processing time behavior of the algorithm. This allows the key to be determined significantly faster than by systematically searching the key space.

**Nonrepudiation** A usually cryptographic method used to ensure that the recipient of a message cannot falsely deny receipt of the message, thus enabling the sender of the message

<sup>71</sup> See also Section 10.11, ‘Near Field Communication (NFC)’, on page 348

to prove that the intended recipient actually received the message. This is similar to a registered letter with return receipt in a conventional postal system.

**Nonvolatile memory** A type of memory (such as → *EEPROM* or → *flash*) that retains its content even in the absence of power.

**NOR flash** → *flash*

**NPU (numeric processing unit)** → *cryptoprocessor*

**NSA (National Security Agency)** The NSA [NSA] is the US government organization officially responsible for cryptography and communication security. It reports directly to the Department of Defense, and one of its functions is to monitor and decode foreign communications. Developing new cryptographic algorithms and restricting the use of existing algorithms also fall under the authority of this agency.

**Null PIN** A standard, known → *PIN* code used for all newly issued → *smart cards*, which nevertheless does not allow access to the actual user functions. It is thus a type of → *trivial PIN*. The first time the card is used, the null PIN must be changed to a user-selected PIN by using one of the standard mechanisms (usually CHANGE CHV); the null PIN value is not allowed for the new PIN. The purpose of a null PIN is to enable the card user to see whether the card is still in its original issued state when the user receives the card or has been illicitly used already while underway. The name ‘null PIN’ comes from the fact that the value ‘0000’ is often used for this type of PIN.

**Numbering** The process of marking a → *chip card* with a number by embossing or printing. This is typically done in the production of anonymous telephone cards in order to assign them unique, visible numbers for identification purposes.

**Object** In the context of → *object-oriented programming*, a software construct built according to the instructions defined by a → *class* and containing data, which means it has → *attributes* that can read and altered by using the → *methods* defined for the class.

**Object-oriented programming** A form of programming based on storing all the data of a software application in → *objects*, which also provide → *methods* that can be used to read or modify the data. The objects are defined by → *classes*. A key aspect of object-oriented programming is that it focuses on the data to be processed instead of the processes as with → *procedural programming*. Typical object-oriented programming languages in the smart card environment are C++ and → *Java*.

**OCF (Open Card Framework)** The Open Card Framework [OCF] specification describes a Java-based, platform-independent interface for integrating smart cards into any desired application on a PC. It presumes the availability of a suitable driver for each type of terminal to be used with the PC concerned and that the smart cards used in the system are OCF-compatible.

**Offcard application** → *application*

**Oncard application** → *application*

**Oncard matching** Refers to the ability of a → *smart card* to compare biometric data measured either oncard or offcard with reference data stored in the smart card for the purpose of user → *identification*.

**One-time password** A password that is sent to the user via a particular communication channel (such as e-mail) and that can be used only once, after which it is no longer valid.

**One-to-one** Designates a relationship between two objects such that each one references the other (i.e. A implies B and B implies A).

**One-way function** A mathematical function that is easily computed but whose inverse function requires a very large amount of computational effort.

**Open application** An → *application* in a smart card that is available to various service providers (such as merchants) without requiring any explicit legal relationship between the various parties.

**Open platform** → *open smart card operating system*

**Open Platform (OP)** A smart card operating system interface for managing smart card applications, originally specified by Visa International and formerly known as Visa Open Platform (VOP). The Open Platform specification has since been renamed → *Global Platform* and is now published by the Global Platform association.

**Open purse** An instance of an open → *application* for an electronic purse system. It can be used for general payment transactions with various service providers.

**Open smart card operating system** A → *smart card operating system* that allows third-party → *applications* and programs to be loaded and run in a secure environment in the associated → *smart card* without the involvement of the → *operating system producer*. The best-known open smart card operating systems are → *Multos* and → *Java Card*. Open smart card operating systems are usually → *interoperable* and not → *proprietary*. Another name for an open smart card operating system is ‘open platform’. This should not be confused with → *Open Platform*, which is an interface for managing applications in smart cards.

**Operating system (OS)** An operating system<sup>72</sup> comprises all of the programs of a digital computer system that, in combination with the hardware features of the computer system, form the basis for its possible operational modes, in particular monitoring and controlling the execution of programs.

**Operating system producer** An entity that programs and tests an → *operating system*.

**Optical memory card** A card in which data is stored by ‘burning’ it into a reflective surface layer (similar to a CD or DVD).

**OTA (over the air)** In → *GSM* and → *UMTS* systems, OTA refers to the possibility of establishing an → *end-to-end* connection between the background system and the → *SIM* or → *USIM* via the air interface between the base station and the mobile station. This can be used for purposes such as sending commands directly and transparently from the background system to the SIM. OTA also forms the basis for all → *value-added services* in the SIM because these services can exchange data directly and transparently with higher-level systems over the air interface. The Short Message Service (*SMS*) is often used as the → *bearer* service for OTA.

<sup>72</sup> See also Chapter 13, ‘Smart Card Operating Systems’, on page 441

**Overlay** Foils on the front and back surfaces of a → *card body*, which protect the → *internal foil*. Overlays are often transparent, and in this form they provide scratch protection for the graphic design on the underlying foils.

**Package** A → *name space* in Java Card and the smallest entity in the Java language. A package can have classes and interfaces.

**Packet-switched** With a packet-switched connection, the sender divides the data to be exchanged into packages, which are then sent individually to the recipient, possibly via separate paths. The recipient reassembles the packages to recover the original data. Charges for a packet-switched connection usually depend on the amount of data exchanged, rather than the duration of the connection as with a → *circuit-switched connection*. Typical examples of packet-switched connections are X.25 and → *GPRS*.

**Padding** Extending a data string with filler data in order to bring it to a particular length.<sup>73</sup> The length of the padded data string usually has to be an integral multiple of a certain block size (such as 8 bytes) so that it can be processed by a routine such as a cryptographic algorithm.

**Page-oriented** Designates a type of memory in which sets of bytes can only be written or erased as a group. In → *smart card microcontrollers*, only EEPROM and flash EEPROM are page-oriented, with typical page sizes of 8, 32, 128, or 256 bytes. However, there are now microcontrollers with page sizes that are variable within a certain range, such as 1 to 128 bytes, instead of fixed.

**Parallel data transmission** A data transmission method in which several data bits (e.g. 8, 16 or 32) are transmitted simultaneously over a corresponding number of data lines.

**Parity bit** Probably the best-known type of error detection code (EDC) is a parity bit appended to the byte to be protected. Before a parity bit can be calculated, it is necessary to specify whether even or odd parity is to be used. With even parity, the value of the parity bit is chosen such that the total number of bits with a value of 1 in the combination of the data byte and the parity bit is an even number. With odd parity, the total number of bits with a value of 1 must be odd. With a single parity bit, one incorrect bit per byte can be reliably detected. However, it is not possible to correct a bit error, since the parity bit does not provide any information about the location of the incorrect bit.

**Passivation** A protective layer on top of a semiconductor chip that screens it against oxidation and other chemical processes. The passivation layer must be partially or fully removed before the semiconductor device can be physically manipulated.

**Passport** An official document issued by a national government to a citizen of the country concerned. Like an → *ID document*, it establishes the identity of its holder with a claim to binding force, and in particular it authorizes the holder to travel to foreign countries and (usually) to return to the territory of the country that issued the passport.

**Patch** In software development, a small program, sometimes written in machine code, that modifies or extends the functionality of an existing program. Patches are commonly

<sup>73</sup> See also Section 7.1.3, 'Padding', on page 154

used for fast, simple correction of program defects. They can be implemented either as → *workarounds* or as → *bug fixes*.

**Patent** A document granting an inventor the right to the exclusive exploitation of an invention for a limited period in one or more countries. The maximum term of a patent is usually 20 years.

**Pay before** Refers to a type of money flow with cards used for payment transactions where the real money flows out of the cardholder's account before the goods or services are actually purchased. A typical example of a pay-before card is an → *electronic purse* card, which the user must load with electronic money before making purchases. In the telecommunication sector, this type of payment is called → *prepaid*.

**Pay later** Refers to a type of money flow with cards used for payment transactions where the real money flows out of the cardholder's account only some time after the goods or services are actually purchased. A typical example of a pay-later card is a credit card, with which several weeks may elapse between the time of purchase and the time when the money is transferred from the purchaser's account to the merchant's account. In the telecommunication sector, this type of payment is called → *postpaid*.

**Pay now** Refers to a type of money flow with cards used for payment transactions where the real money flows out of the cardholder's account at the same time as the goods or services are purchased. A typical example of a pay-now card is a debit card, such as the Eurocheque card, which allows money to be transferred from the purchaser's account to the merchant's account when the purchase is made.

**PC/SC (Personal Computer / Smart Card)** The PC/SC specification [PC/SC] describes an interface for integrating smart cards into any desired application, independent of the platform and programming language used.<sup>74</sup> This presumes that a suitable driver is available for the terminal used with the PC and the smart card is PC/SC-compatible.

**PCD (proximity coupling device)** A terminal that communicates with a contactless card (→ *PICC*).

**Persistence** The ability of an object to survive after the execution time of a process; the opposite of → *transience*. Persistent objects continue to exist after the end of a session or after a sudden loss of power, without any consequent loss of data or data inconsistency.

**Personalization** The process of associating a card with a person.<sup>75</sup> This can be achieved by means of physical personalization (such as embossing or laser engraving) or by means of electronic personalization (loading personal data in the memory of the smart card). A better term for this is 'individualization', since electronic personalization does not always involve loading personal data in the chip (for example, in the production of anonymous → *prepaid SIMs*).

**Personalizer** An entity that performs personalization.

**Phase 1, Phase 2, Phase 2+** These phases designate the successive evolutionary stages in the development of the GSM system. The basic services (including voice transmission,

<sup>74</sup> See also Section 8.7.1, 'PC/SC', on page 237

<sup>75</sup> See also Section 14.8.2, 'Personalization (individualization)', on page 619

call forwarding, → *roaming* and → *SMS*) were implemented in Phase 1. In Phase 2, which began in 1966, the Phase 1 services were augmented by value-added services such as conference calls, call handoff and calling number conveyance, as well as GSM in the 1800-MHz band. This was followed by Phase 2+, in which these services were augmented by (among other things) the → *SIM Application Toolkit* functions, high-speed circuit-switched data (HSCSD), and the General Packet Radio System (→ *GPRS*).

**Phishing** A neologism formed from ‘password’ and ‘fishing’ that refers to the technique of using false information to obtain personal passwords with deceitful intent.

**PICC (proximity integrated circuit card)** A contactless smart card with a range of approximately 10 cm.

**PIN pad** In the original sense, a keypad with special mechanical and cryptographic protection for use with a terminal. In common usage, an entire terminal is often called a PIN pad.

**PIN (personal identification number)** A secret number, usually consisting of four digits, used for the → *identification* of a person.<sup>76</sup> In the telecommunication sector, the equivalent of the PIN is usually called CHV (cardholder verification).

**Pipelining** A process in which data reception and writing data to nonvolatile memory take place concurrently. This process is used for loading data in smart cards during production.<sup>77</sup>

**PKCS #1 ... PKCS #15 (Public Key Cryptography Standard 1 ... 15)** RSA Inc. [RSA] standards for public-key cryptography that focus on the use of asymmetric cryptographic algorithms such as the → *RSA* algorithm.

**PKI (public key infrastructure)** Designates all of the facilities and systems necessary for performing and managing data storage and data exchange based on asymmetric cryptographic methods, including a → *certification authority*, a → *registration authority*, a → *directory service* for blacklists (→ *certificate revocation list*), a → *time-stamp* service, and → *signature cards*.

**Plastic card** A card made from plastic, typically in ID-1 format, with a semiconductor chip or security features for protection against unauthorized copying.

**PLMN (public land mobile network)** Customary designation for a terrestrial mobile telecommunication network.

**Plug-in** Alternative name for a smart card in → *ID-000* format.<sup>78</sup>

**Plugins** → *browser*

**Polling** Periodic program-driven querying of an input channel in order to detect an incoming message. Depending on the repetition rate of the queries, polling can require significant processing capacity, for which reason it is usually avoided in favor of hardware-based querying using interrupts. An example of polling is periodic querying of the SIM in a mobile telephone to enable the use of proactive commands.

<sup>76</sup> See also Section 7.8.2, ‘Testing a secret number’, on page 188

<sup>77</sup> See also Section 14.7.5, ‘Accelerating data transfer to the smart card’, on page 616

<sup>78</sup> See also Section 3.1, ‘Card Formats’, on page 29

**POS (point of sale)** The location where a particular item or service is sold.

**POS terminal** A POS (point of sale) terminal is a magnetic card and/or ICC card reader with a display, keypad and microcontroller. POS terminals are used to pay for and/or authorize retail sales.

**Postpaid** Refers to a type of money flow for cards used in the telecommunication sector where the real money of the cardholder flows only after the service (usually a telephone call or data transfer) has been received. Postpaid cards are comparable to credit cards in terms of their payment function. In payment systems, this type of payment is called → *pay later*.

**Power-on reset** → *reset*

**Prepaid** Refers to a type of money flow for cards used in the telecommunication sector where the real money of the cardholder flows before the service (usually a telephone call or data transfer) is received. Postpaid cards are comparable to electronic purses in terms of their payment function. In payment systems, this type of payment is called → *pay before*.

**Prepaid SIM** A prepaid and usually reloadable SIM. All billing and reloading functions are usually provided by the background system, so they have no effect on data objects or functions in the SIM. The opposite of this is a → *postpaid SIM*.

**Prepersonalization** Another term for → *initialization*.

**Proactivity** A transaction mechanism for smart cards that allows a smart card to independently initiate actions in the terminal.<sup>79</sup> This circumvents the rigid master–slave relationship between the terminal and the smart card. Proactivity is realized by periodic polling of the smart card by the terminal, with the polling interval being configurable in advance by the smart card. Proactivity originated with SIMs, and it is still predominantly used to allow SIMS to effectively assume control over certain functions of the mobile equipment in accordance with TS 51.014.

**Probabilistic** Designates a process or an algorithm that yields varying results with identical input conditions; the opposite of → *deterministic*.

**Procedural programming** A programming method based on formulating a program as a sequence of instructions to be executed by a → *microprocessor*. For the sake of simplification, the program flow can be broken down into functions and the necessary data can be held in variables. A key aspect of procedural programming is that it focuses on the processes of the program instead of the data to be processed, as with → *object-oriented programming*. Typical programming languages used for procedural programming are Basic and C.

**Process model** Another term for → *life cycle model*.

**Processor** → *microprocessor*

**Processor card** A card with a chip, which may be a → *microcontroller* with a CPU, volatile memory (→ *RAM*), and nonvolatile memory (→ *ROM* and/or → *EEPROM*). A processor card may also contain a numeric coprocessor (→ *cryptoprocessor*) for rapid

<sup>79</sup> See also Section 19.4.4.6, ‘SIM Application Toolkit’, on page 833

execution of public-key cryptographic algorithms. Such cards are sometimes called cryptocards or cryptocontroller cards.

**Profile** A document that specifies the → *smart card operating system* and → *smart card application* for a specific customer, including all of the applets, files, and access conditions.

**Program code** Designates a program in a form that can be executed by an → *interpreter* or directly by a → *microprocessor* (→ *native code*).

**Proprietary** An adjective used in the smart card world, often in a deprecatory sense, to designate a company-specific solution whose specifications are not fully public or belong to a single company. The opposite of a proprietary solution is an open solution, which can also be used by third parties. However, these terms are used in a far from unambiguous manner. In objective terms, many ‘open’ smart card operating systems are rather proprietary and dependent on a particular company. An example of a proprietary smart card → *application* would be an electronic purse system for use in a specific region that does not comply with relevant specifications and has been developed by a particular company as a specific solution.

**Protection profile (PP)** In the context of → *evaluation*, an implementation-independent set of security requirements (→ *security target*) adapted to a particular application area for a specific → *target of evaluation*.<sup>80</sup>

**Proton** Brand name of an electronic purse system with approximately 40 million issued cards. The Proton specification also defines a multiapplication → *smart card operating system*.

**Prototype** A (software) prototype is an executable model of the ultimate product with reduced functionality. It is used to experimentally investigate specific properties of the ultimate product. A horizontal prototype implements only one or more specific layers of the software, while a vertical prototype implements a specific portion of the software across all of the layers.

**Pseudonymization** The process of modifying person-specific data using an assignment rule such that it is afterward not possible to associate the data with the original persons without knowing the assignment rule. The term is based on the fact that in the simplest case, the original name of each person is replaced by a unique pseudonym. A separate assignment table (the assignment rule) can be used to restore the links between the pseudonyms and the original names. A stronger process that prevents the recovery of the relationship is → *anonymization*.

**PSTN (public switched telephone network)** The regular public landline telephone network.

**PTS (protocol type selection)** An outdated term for ‘protocol parameter selection’ (PPS).

**Public-key algorithm** → *cryptographic algorithm*

**PUK (personal unblocking key)** A special → *PIN* for resetting the → *error counter* of a PIN after it has reached its maximum value. A PUK is usually longer than a PIN (e.g. 8 digits), since users do not need the PUK unless they have forgotten the PIN, at which

<sup>80</sup> See also Section 15.5.1, ‘Common Criteria’, on page 661

time they can look for the PUK in their documents. When the PUK is used successfully, a new PIN is specified at the same time because the old PIN is evidently no longer known to the user.

**Pull principle → pull technology**

**Pull technology** Information transfer in which a lower-level system (such as a mobile telephone) fetches information from a higher-level system (such as a server). It is the opposite of → *push technology*.

**Purse holder** A person or entity that possesses a → *smart card* with an electronic purse.

**Purse provider** The organization responsible for the overall functionality and security of an electronic purse system. In most cases, the purse provider also issues the electronic money and is responsible for the redemption of the electronic money.

**Purse-to-purse transaction** Transfer of electronic monetary units from one electronic purse directly to another without the intervention of a third, higher-level system. Normally, such capability requires the purse system to operate anonymously and the electronic purses to use a single common key for this function.

**Push technology** Information transfer in which a higher-level system (such as a server) sends information to a lower-level system (such as a mobile telephone). It is the opposite of → *pull technology*.

**PVC (polyvinyl chloride)** The most popular material for producing bank cards and payment cards. PVC is easy to emboss, which is an advantage in such applications, but it suffers from a relatively low melting temperature.

**Quick** Brand name of an electronic purse system introduced throughout Austria in 1995. The essential components of the Quick system are based on EN 1546, which is the European standard for inter-sector electronic purse systems.

**Quota** The memory management mechanism used in smart cards for third-party applications is called the quota mechanism, named after corresponding functionality in Unix systems. It enables the card issuer to specify the maximum amounts of volatile and nonvolatile memory that can be allocated to each downloaded application and to adjust these figures as necessary.

**RAM (random-access memory)** A type of volatile memory used in smart cards as working memory.<sup>81</sup> RAM loses its contents in the absence of power. → *SRAM* and → *DRAM* are types of RAM with special characteristics.

**Random number generator (RNG)** A method for generating a series of random numbers. It may consist of a pseudorandom number generator (PRNG), which is usually implemented in software and generates a deterministic sequence of random numbers. A random → *seed number* (initial value) must be used to generate different sequences of random numbers. Nondeterministic random number generators exploit a physical phenomenon that yields random values, such as the noise signal of a current source, and use software to process the measured values. Such random number generators are often called true random number generators (TRNGs).

<sup>81</sup> See also Section 5.3.5, ‘RAM’, on page 92

**Reader** A short form of → *card reader*.

**Record** A record (data set) consists of a specific number of data elements, similar to a string.

**Redlist** → *hotlist*

**Registration authority** An entity of a → *PKI* that receives certification requests from users and forwards them to the → *certification authority* after verifying the authenticity of the requester. The registration authority is thus the entity that creates a unique association between certificates and persons.

**Relay attack** A form of attack in which the communication distance of contactless smart cards is increased to several times its normal range so that transactions can be performed on cards without the knowledge or awareness of the rightful users.<sup>82</sup>

**Remote applet management (RAM)** Management (creation, deletion, etc.) of → *applets* in a smart card by a → *background system*. For example, in various → *GSM* systems applets can be loaded in a → *SIM* or deleted from the SIM via the air interface.

**Remote file management (RFM)** Management (creation, deletion, writing, reading, modifying access conditions, etc.) of files in a → *smart card* by a → *background system*. For example, various → *GSM* systems allow new files to be created in a → *SIM* and data to be written to these files, all via the air interface.

**Reset** Restoring a computer (in this case, a smart card) to a defined initial state. A cold reset, or power-on reset, is initiated by switching the supply voltage off and then on again. A warm reset is initiated by a signal on the reset lead of the smart card without altering the supply voltage.

**Response** → *command*

**Response APDU** A response APDU<sup>83</sup> is the → *response* of a → *smart card* to a → *command APDU* sent by the → *terminal*. It consists of optional response data and a mandatory portion containing the single-byte status words SW1 and SW2. (→ *APDU*)

**Response data** In the smart card context, the data generated for the card issuer.

**Reticule** → *ROM mask*

**RFID (radio frequency identification)** A technology used to identify objects over a distance of up to a few meters using radio signals (→ *identification*). For this purpose, the objects concerned must be fitted with RFID chips.

**RMI (remote method invocation)** A protocol-independent mechanism used to invoke → *methods* on other computers and transfer data to them.

**Roaming** Accessibility of a mobile telephone in a network other than its → *home net*.

**Roll back** Functionality of an operating system for maintaining data consistency in the event of an error or abnormal termination. When a roll back occurs, the data of an improperly executed or aborted operation is replaced by the original data. This process

<sup>82</sup> See also Section 16.5.3, ‘Attacks on applications’, on page 727

<sup>83</sup> See also Section 8.3.2, ‘Response APDU structure’, on page 224

can be initiated automatically or on demand, and in smart card operating systems it is often implemented using → *atomic operations*. Another strategy for maintaining data consistency in case of errors or abnormal termination is → *roll forward*.

**Roll forward** Functionality of an operating system for maintaining data consistency in the event of an error or abnormal termination. When a roll forward occurs in response to an improperly executed or aborted operation, the available but inconsistent data of the operation is fed back into the operation in such a way that on completion the data is again consistent. This process can be initiated automatically or on demand, but it is rarely implemented in smart card operating systems due to their stringent security requirements. Another strategy for maintaining data consistency is → *roll back*.

**ROM (read-only memory)** A type of nonvolatile memory used in smart cards.<sup>84</sup> It is mainly used to store programs and static data, since ROM contents cannot be altered.

**ROM mask** In ordinary language, this term is used in a highly context-specific manner.<sup>85</sup> The original meaning is an exposure mask used in semiconductor fabrication to produce the ROM. However, the term ‘mask’ is only used when the mask image is not reduced in scale when exposing the → *wafer*. If the structures are reduced in scale for imaging onto the wafer, the mask is called a reticule. The term ‘mask’ is also used in connection with → *smart card microcontrollers* to refer to the data content of the ROM, and in some cases it is even synonymous with the entire → *smart card operating system* (→ *soft mask*, → *hard mask*).

**ROMed application** A smart card application that is located in the mask-programmed ROM of the → *smart card microcontroller* instead of the EEPROM and is thus unalterable.

**Root CA** → *certification authority*

**Round-trip engineering** A software development method in which design and implementation activities are performed concurrently and thus influence each other. The mutual consistency of the software architecture and program code is maintained automatically by software. This process is based on a formal modeling language such as UML, from which at least the basic structure of the program is generated using automatic program code generation. The insights and improvements obtained by refining and testing this program code flow back into the program model via a reverse engineering process. Usable software with source code that is consistent with the software architecture can be produced relatively quickly with this method by cycling through the code generation and reverse engineering loop several times.

**RSA (Rivest, Shamir, Adleman)** The best known and most widely used asymmetric → *cryptographic algorithm* was published in 1978 by Ronald L. Rivest, Adi Shamir and Leonard Adleman, and its name comes from the initials of the surnames of its authors. Its simple operating principle is based on the arithmetic of large integers, and its security is based on the factorization problem.

**R-UIM (removable user identity module)** An optional, interchangeable security module for use in mobile telephones in the → *CDMA 2000* mobile telecommunication

<sup>84</sup> See also Section 5.3.1, ‘ROM’, on page 84

<sup>85</sup> See also Section 14.3.3, ‘Chip fabrication in semiconductor plants’, on page 575

system.<sup>86</sup> The functionality of the R-UIM is similar to that of the → *SIM*, although the → *CAVE* (cellular authentication, voice privacy and encryption) cryptographic algorithm is used for numerous cryptographically secured functions in the R-UIM. A UIM Application Toolkit (*UATK*) based on the → *SIM Application Toolkit* is also specified for the R-UIM.

**Rule-based programming** A programming method based on the formulation of general rules that can subsequently be employed to solve problems. A computer can then use these rules to solve problems independently. A key aspect of rule-based programming is that it focuses on general rules instead of processes as with → *procedural programming* or the data to be processed as with → *object-oriented programming*. Typical rule-based programming languages are Lisp and Prolog.

**S<sup>2</sup>C** Designates a two-wire bus with two lines (SigIn and SigOut) that supports full-duplex communication between two parties. It was originally proposed by Philips (NXP) for communication between the → *USIM* and the → *NFC* controller in mobile telephones with NFC functionality.

**S@T browser** → *browser*

**Salt** A random sequence used to extend a password in order to hinder dictionary attacks on stored passwords.

**SAM (secure application module)** → *security module*

**Sandbox** → *virtual machine*

**SCOPE (Smart Card Open Platform Environment)** An initiative for the specification of a hardware abstraction layer (→ *HAL*) for smart card microcontrollers. It failed to achieve general acceptance.

**Scrambling** Intentional jumbling of the address, data and control buses on a microcontroller chip so that it is not possible to recognize the purpose of individual bus lines without inside information. With static scrambling, the buses of a batch of chips are all scrambled in the same way, while with dynamic scrambling, the buses are scrambled differently on each chip or even during each session.

**Scrap rate** The proportion of reject cards in card production.

**Scratch card** A card in standard → *ID-1* format but thinner than usual, with a printed number concealed beneath an opaque seal layer that can be scratched off. The seal layer allows the integrity of the card to be verified visually before it is used. A scratch card is similar to a PIN letter in this respect. Scratch cards are often used as vouchers for distributing one-time passwords used to reload → *prepaid SIMs*.

**Script** An interpreted program, primarily used to implement a simple, short application or automate a recurrent process.

**SDMA (space division multiple access)** A multiple access method for concurrent data transmission from a transmitter to several receivers using a single frequency. For this

<sup>86</sup> See also Section 19.4.4, ‘The SIM’, on page 811

purpose, the transmitter uses directionally selective antennas aimed at the individual receivers. Due to the complexity of this method, it can only be implemented in base stations in mobile telecommunication systems, for instance by using array antennas (adaptive antennas).

**SECCOS (Security Card Operating System)** A multiapplication → *smart card operating system* used in German Eurocheque cards with chips and → *GeldKarte* cards.

**Secret-key algorithm** → *cryptographic algorithm*

**Sector** → *granularity*

**Secure messaging** All methods, protocols and cryptographic algorithms used to protect data transmissions to and from → *smart cards* against manipulation and interception.<sup>87</sup>

**Secure signature creation device (SSCD)** A signature generation device, such as a → *smart card*, that is protected against attacks. This term originates from a specific → *protection profile* defined in the → *Common Criteria*.

**Security environment (SE)** In a smart card, a logical container holding a set of fully defined security measures used by security-related → *commands* or used for → *secure messaging*. Security environments are well suited to protecting the various → *life cycle* phases of a smart card. For example, in the simplest case different security environments can be defined for the card personalization phase and the subsequent card use phase, so that different file → *access conditions* can be specified for the different phases of the smart card life cycle. With this arrangement, write access can be allowed to all files during personalization, while the access conditions during normal use can be specified as appropriate for the → *application* concerned.

**Security module** A component that is secured by mechanical and information technology measures and used to hold secret data and execute cryptographic algorithms. It is also called a secure application module (SAM), hardware security module (HSM), or host security module (HSM).

**Security requirements** In the context of an → *evaluation*, an implementation-independent set of requirements (→ *security target*) for a specific → *target of evaluation*, usually grouped into a → *security profile*.<sup>88</sup>

**Security target (ST)** In the context of an → *evaluation*, security targets describe the mechanisms of the → *target of evaluation* that are to be evaluated. They thus form a sort of set of requirements for the evaluation. Security targets for specific types of targets of evaluation and application areas can be described by → *protection profiles*.

**Seed number** A → *random number* used as the initial value of a pseudorandom number generator.<sup>89</sup>

**SEMA (simple electromagnetic analysis)** A technique similar to simple power analysis (→ *SPA*), used to make deductions regarding the internal processes of a microcontroller by analyzing its electromagnetic radiation during operation. A more sophisticated version of this method is differential electromagnetic analysis (→ *DEMA*).

<sup>87</sup> See also Section 8.4, ‘Secure Data Transmission’, on page 225

<sup>88</sup> See also Section 15.5.1, ‘Common Criteria’, on page 661

<sup>89</sup> See also Section 7.3, ‘Random Numbers’, on page 159

**Sequence control** A technique for specifying a mandatory sequence of activities. For example, the proper → *command* sequence for mutual → *authentication* of the → *smart card* and the → *background system* can be enforced by using sequence control in the smart card. For this purpose, the → *smart card operating system* has a → *state machine* with specified states and state transitions that define the command sequence to be used.

**Serial data transmission** A type of data transmission in which individual data bits are sent sequentially over a data line; the opposite of → *parallel data transmission*.

**Server** A central program that provides data and/or programs to one or more → *clients*.

**Service provider** An entity that provides goods or services in exchange for payment. In a smart card system, an entity that provides services that are used and paid for by a user. In an electronic purse system, a service provider is an entity that receives money from the electronic purse of a purse holder in exchange for goods or services.

**Session** The period between the activation and deactivation sequences of a smart card, during which all of the necessary data is exchanged and the related information technology mechanisms are executed.

**Session key** A cryptographic key (also called a dynamic key) that is valid during only one → *session*.

**SET (Secure Electronic Transaction)** A financial transaction protocol for secure credit card payments via the Internet, published by Visa and MasterCard in 1996. Although SET supports credit cards with chips (Chip SET), this is optional because SET can be implemented entirely in software on a PC. SET has not achieved widespread acceptance in the Internet payments sector.

**SHA-1 (Secure Hash Algorithm 1)** A → *hash function* that produces a 160-bit hash value; originally published in 1995. The SHA-1 algorithm should not be used for new products because a weakness was published in 2006. Improved versions of SHA-1, published in 2002 by the → *NIST*, are SHA-224 with a 224-bit hash value, (SHA-256) with a 256-bit hash value, and SHA-384 with a 384-bit hash value.

**Shall, should, may** These three auxiliary verbs are often used in international standards. Their meanings in this context are precisely defined and differ in part from more relaxed usage in common language. ‘Shall’ designates something that must be implemented as described. ‘Should’, although it may appear to be a recommendation, actually means that the described item must be implemented or provided if at all possible. Only items designated by ‘may’ are truly optional.

**Shared secrets** A principle according to which no single person knows everything about a particular system. The intentional distribution of knowledge avoids making individual persons subject to attack and prevents individuals from acquiring excessive power over a system due to their knowledge. Distributing knowledge over several persons is a commonly used technique in the development of security components.

**Short FID** A five-bit EF identifier that can have a value of 1 to 31.<sup>90</sup> It is used with a write or read command (such as READ BINARY) for implicit selection of an EF in a smart card.

<sup>90</sup> See also Section 12.5.2, ‘Short file identifier’, on page 428

**Shrink** Reduce the surface area of a semiconductor chip by employing a semiconductor technology with a smaller structure width. A smaller chip surface area allows a larger number of chips to be located on a → *wafer*. This in turn reduces the price of the individual devices, which is roughly proportional to the amount of space occupied by the chip on the wafer.

**Shutter** A mechanical device in a terminal that severs any wires leading out of the terminal from the card.<sup>91</sup> This is intended to prevent manipulation of communication. If the wires cannot be cut, the inserted smart card will not be electrically activated.

**Side channel attack** A form of attack that exploits generally undesirable physical characteristics of the hardware used in a smart card. It involves measuring the timing characteristics, current consumption, electromagnetic radiation, or some other characteristic in order to obtain additional information about the data or the program processes in the microcontroller, after which this information is correlated with known information. This form of attack is strongly dependent on the type of microcontroller concerned, and it can be simulated only at considerable effort and expense.

**Signal burst** A cohesive data packet transmitted between a base station and a mobile station over the air interface; often simply called a burst.

**Signature Act** In general, legislation that governs the use of → *digital signatures*.<sup>92</sup> In Germany, this means the Signaturgesetz (SigG), which is known in full as Gesetz über Rahmenbedingungen für elektronische Signaturen. It defines general provisions for the use of digital signatures in Germany, which are given more concrete form in the → *Signaturverordnung*.

**Signature card** A → *smart card* whose principal function is to secure the storage and use of secret keys for → *digital signatures*.<sup>93</sup>

**Signature strip** A writable strip on a card that the → *cardholder* signs, preferably in the same way as a normal document. Signature strips are often designed such that an attempt to glue another strip on top or erase the original signature can be detected.

**Signaturverordnung (SigV)** The Signaturverordnung (Digital Signature Ordinance) of 22 October 2001 translates the general provisions of the Signaturgesetz (→ *Signature Act*) into more concrete terms to the extent necessary to enable the generation of catalogs of specific measures that serve as recommendations for the practical use of digital signatures. For example, it describes the methods necessary for generating signature keys and identification data, as well as the necessary security concepts and → *Common Criteria* evaluation levels for signature components.

**SIM (subscriber identity module)** The standard designation for a GSM-specific smart card.<sup>94</sup> The SIM is a mandatory, interchangeable security module used in GSM mobile telephones. SIMs are now made exclusively as plug-in cards in → *ID-000* format. The SIM bears the identity of the subscriber, and its primary function is to secure the

<sup>91</sup> See also Section 17.5, ‘Security’, on page 746

<sup>92</sup> See also Section 23.1.2, ‘The legal framework in Germany’, on page 900

<sup>93</sup> See also Section 23.1.7, ‘Signature cards’, on page 906

<sup>94</sup> See also Section 19.4.4, ‘SIM’, on page 811

authenticity of the mobile station with respect to the network. Additional functions include executing programs with protection against manipulation, user identification (using a PIN), and storing data such as telephone numbers. The equivalent of the SIM in → UMTS systems is the → USIM.

**SIM Alliance** A consortium founded in 1999 by Gemplus, G&D, ORGA, and Schlumberger to allow services developed for WAP to be used with mobile telephones lacking WAP capability. For this purpose, the SIM must have a browser with SIM Alliance capability (S@T browser), and the mobile telephone must support → Phase 2+ of → GSM. This gives the SIM sufficient control over the mobile telephone via the → SIM Application Toolkit to display Internet content and support Internet functions. In the meantime, the SIM Alliance has also established a position as a worldwide source of marketing data for → SIMs and → USIMs.

**SIM Application Toolkit (SAT, STK)** A modular extension to the TS 11.11 specification, standardized by TS 51.011, that allows the SIM to assume an active role in controlling the mobile telephone. For example, with the SIM Application Toolkit a SIM can output data and imagery to the display, request keypad entries, and send and receive messages over the air interface. The SIM Application Toolkit forms the basis for most value-added services in mobile telephones. The equivalent of the SIM Application Toolkit in UMTS is the → USIM Application Toolkit (USAT), while the equivalent for the → R-UIM is the UIM Application Toolkit (→ UATK). The generic basis of all future application toolkits for smart cards used in mobile telecommunication systems is the Card Application Toolkit (→ CAT) defined by the → EP SCP expert group.

**SIM lock** A technique that binds a specific mobile telephone to a specific → smart card (→ SIM).<sup>95</sup> It involves either having the mobile telephone read certain data from the SIM and compare it with data stored in the mobile telephone, or having the SIM read unique data from the mobile telephone and compare it with stored data. If the two sets of data match, the mobile telephone can be used. It is usually possible to disable the SIM lock by a transaction over the air interface or by entering a secret key using the keypad of the mobile telephone, after which other SIMs can also be used. The SIM lock function is used to bind a mobile telephone subsidized by a network operator to a particular SIM and its payment mode (→ prepaid) for a certain period.

**SIM toolkit** A short form of → SIM Application Toolkit.

**SIMEG (Subscriber Identity Module Expert Group)** An expert group under the aegis of → ETSI, founded in 1988, that developed the specification for the interface between the smart card and the mobile telephone (GSM 11.11; now TS 11.11). The name was changed to → SMG9 in 1994.

**Simplex** A data transmission method in which there is a distinct sender and a distinct receiver; in other words, the data flows in one direction only. The other two methods are → half duplex and → full duplex.

**Simulator** A software emulation of the operation of a device (target system). By contrast, a hardware emulation is called an → emulator. Simulators are often used in the

<sup>95</sup> See also Section 19.4.4.12, 'Operating principle of SIM lock', on page 845

development of software for target systems that do not yet exist. For instance, a smart card simulator consists of software that fully emulates a real smart card at the logical level. Simulators are generally slower than emulators, and in many cases they cannot emulate the target system in real time.

**Single sign-on (SSO)** A technique in which several different user IDs for different applications are replaced by a single, common user ID. This is implemented using software that sends the appropriate user name (→ *identification*) and password to the associated identification authority after the user has been successfully identified at a central location. This eliminates the need for the user to remember many different passwords.

**Skimming** A typical form of attack on magnetic-stripe cards. It involves illicitly reading the magnetic-stripe data from a card not belonging to the attacker and copying it to the magnetic stripe of a blank card, which can then be used in the same way as the original card (with regard to its magnetic stripe). This term is also used now to refer to copying a smart card.

**Smart card** Another term for → *processor card*. It designates a chip card that is ‘smart’, which means it does not properly apply to → *memory cards*.

**Smart card application** → *application*

**Smart card microcontroller** Smart card → *microcontrollers*<sup>96</sup> are specifically adapted to the needs and requirements of smart cards. These enhancements primarily involve chip security aspects (such as shield layers and detectors), chip size, and special functional units that fulfill requirements specific to smart cards (such as a UART for communication).

**Smart card operating system** A smart card operating system,<sup>97</sup> also called a card operating system (COS), is a specialized type of → *operating system* tailored to the needs of smart cards and encompassing all of the routines in a → *smart card microcontroller* necessary for using and managing smart card → *applications*. For this purpose, the data, files, → *commands*, processes, states, mechanisms, algorithms, and routines needed by one or more smart card applications must be supported in a suitable manner. In terms of their functions, modern smart card operating systems are typically oriented toward the ISO/IEC 7816 family of standards. If a smart card operating system allows several applications to be run concurrently, it is said to be multiapplication capable. The trend in the development of smart card operating systems is toward → *open smart card operating systems*. Some typical smart card operating systems are → *Multos*, → *Java Card*, → *JCOP*, → *TCOS*, and → *STARCOS*.

**Smart label** A thin data storage medium that can communicate using contactless data transmission. The simplest types of smart labels (many of which do not contain chips) are read-only devices. More sophisticated types of smart labels also allow data to be written to the label and/or processed in the label, in which regard they resemble → *smart cards*.

<sup>96</sup> See also Chapter 5, ‘Smart Card Microcontrollers’, on page 73

<sup>97</sup> See also Chapter 13, ‘Smart Card Operating Systems’, on page 441

**Smart object** A smart card microcontroller packaged in a form other than a typical card form. Some examples of smart objects are USB plugs, tokens, and rings equipped with a smart card microcontroller.

**Smartcard** A registered trademark of the Canadian company Groupmark [Groupmark].

**SMG9 (Special Mobile Group 9)** An expert group operating under the aegis of → ETSI that developed specifications for the interface between smart cards and mobile telephones, such as GSM 11.11 (now TS 11.11), GSM 11.14 (now TS 11.14), and so on. It was composed of representatives of card producers, mobile telephone manufacturers, and network operators. The group was previously known as → SIMEG. In 2000, SMG9 was dissolved and its tasks were divided between two new expert groups. The 3GPP T3 expert group is responsible for the application-specific interface between the mobile telephone and the SIM or USIM, while the ETSI Project Smart Card Platform (EP SCP) expert group looks after all general topics related to smart cards in the telecommunication sector.

**SMS (short message service)** The → GSM text messaging service, which can be used to send text messages with a maximum length of 160 characters. SMS messages are sent via the signaling channel instead of the data channel, which means they can also be sent and received during an active telephone conversation. SMS is used not only for conveying text messages to subscribers, but also as a → bearer service for sending data to the mobile telephone (→ WAP) or the → SIM (→ OTA).

**Sniffer** Software for analyzing communication processes during data transmission. In contrast to a → tracer, a sniffer is a pure software tool with no supplementary hardware, for which reason it has distinct limitations, particularly with regard to analyzing the electrical aspects of data transmission.

**Soft mask** Designates an arrangement in which a portion of the program code of a smart card with a → ROM-based → operating system is located in → EEPROM or → flash memory.<sup>98</sup> Routines stored in EEPROM or flash can easily be modified by writing new data and are thus ‘soft’. Here the term ‘mask’ is somewhat misleading, since it is not necessary to produce a semiconductor fabrication mask for program code stored in EEPROM or flash. Soft masks were traditionally not used for large production volumes, but as a result of the replacement of EEPROM by less expensive flash memory, smart cards with soft masks are now produced in large volume as well. The opposite of a soft mask is a → hard mask, which means that all of the essential software functions are held in ROM.

**Soft SIM** → virtual smart card

**SPA (simple power analysis)** A form of attack on smart cards that involves measuring the current consumption of the microcontroller with high time resolution. Conclusions about the internal processes and the data processed in the microcontroller can be drawn from this information. A similar but significantly stronger form of attack is differential power analysis (→ DPA).

**SPA/DPA-resistant** A property of a cryptographic algorithm that does not allow the secret key being used to be determined using → SPA or → DPA.<sup>99</sup>

<sup>98</sup> See also Section 13.5.3, ‘Soft and hard masks’, on page 456

<sup>99</sup> See also Section 16.5.1, ‘Attacks on the hardware’, on page 684

**Specification** An unambiguous, complete and nonredundant description of a software item. Its content must be free of anything subject to interpretation, and it must be formulated such that all types of readers with various roles (developers, testers, QA personnel, etc.) can understand it within an acceptable length of time.

**SRAM (static random-access memory)** SRAM ( $\rightarrow$  RAM) needs a constant supply of power to retain its contents, but it does not have to be refreshed regularly like  $\rightarrow$  DRAM.<sup>100</sup> SRAM has shorter access time than DRAM, but it occupies more space on the chip and is thus more expensive.

**SSL (Secure Socket Layer)** SSL<sup>101</sup> is a transparent protocol between the transport layer and the application layer that provides authenticated and encrypted data transmission. It is primarily used on the Internet. There is also an enhanced version of SSL called Transport Layer Security (TSL). Both versions are in common use.

**Stack** A data structure in which the most recently entered item is the first to be retrieved (last in, first out – LIFO). Probably the best-known stack is the program stack, which is used to hold return addresses when subroutines are called.

**Standard** A document containing technical descriptions and precise criteria used as rules and definitions of characteristics and features in order to ensure that materials, products, processes, and/or services can be used for their intended purposes. In this book, the term ‘standard’ is generally used in connection with a national or international standardization organization such as ISO, CEN, ANSI, or ETSI.

**STARCOS** Brand name of a multiapplication smart card operating system ( $\rightarrow$  *multiapplication smart card*) produced by G&D [GD], which has been released in several versions since 1991.

**State machine** In information technology, a state machine is a software component that specifies a sequence of events by means of a defined state diagram with specific states and state transitions.

**Steganography** The purpose of steganography is to conceal messages within other messages such that they cannot be recognized by a naive observer (human or machine). For example, a message could be coded and hidden in an image file in such a way that it only marginally alters the image, so the changes to the image are practically imperceptible. In connection with images, this is called a  $\rightarrow$  *digital watermark*. With a suitable analysis program, the message (such as a copyright message) hidden in the image file can subsequently be recovered and made visible.

**Super smart card** A smart card with integrated complex card components such as a display and keypad. A more common equivalent term is  $\rightarrow$  *system on card*.

**SWP (Single Wire Protocol)** A full-duplex serial transmission protocol that requires only one signal lead and is used for communication between a SIM or USIM and a contactless front end (CLF).<sup>102</sup>

<sup>100</sup> See also Section 5.3.5, ‘RAM’, on page 92

<sup>101</sup> See also Section 8.4, ‘Secure Data Transmission’, on page 225

<sup>102</sup> See also Section 9.6, ‘Single-Wire Protocol’, on page 278

**Symmetric cryptographic algorithm** → *cryptographic algorithm*

**Synchronous data transmission** Data transmission in which data transfer is linked to a predefined timing reference. The timing reference may be derived from the clock signal applied to the chip. It is the opposite of → *asynchronous data transmission*.

**System on card** In the smart card realm, a designation for a smart card with supplementary card components as well as a chip module. Commonly used supplementary components include displays, power sources (batteries and solar cells), keypads, antennas, sensors for biometric user identification (such as fingerprint readers), and loudspeakers. These card components may be controlled by the chip in the module, but this is not mandatory. Another designation for such cards is ‘super smart card’, although the latter term is now relatively uncommon.

**T = 0** A protocol for data transmission between a terminal and a smart card.<sup>103</sup> The T = 0 protocol was the first internationally standardized transmission protocol for smart cards. It is a byte-oriented, asynchronous half-duplex protocol that is designed for minimum memory usage and maximum simplicity. This protocol is used worldwide with → *SIM* cards and is thus the most widely used of all smart card transmission protocols. The T = 0 protocol is specified in the ISO/IEC 7816-3 standard. Compatible specifications are included in the TS 11.11, TS 102 221, and EMV specifications.

**T = 1** A protocol for data transmission between a terminal and smart card.<sup>104</sup> It is a block-oriented, asynchronous half-duplex protocol with separation between the data transport and application layers. The T = 1 protocol is specified in the ISO/IEC 7816-3 standard. Compatible specifications are included in the TS 102 221 and EMV specifications.

**T = CL** A general designation for contactless data transmission protocols,<sup>105</sup> such as the protocols specified in ISO/IEC 14443.

**T3** → *SMG9*

**Tag** A data object identifier, primarily used in → *ASN.1* encoding.<sup>106</sup>

**Tape out** The point in the development process when the chip design is completed and the resulting design data is issued for mask generation (→ *ROM mask*). This is an important milestone in the chip production process. The term originates from the time when the mask data was usually output to magnetic tape.

**Tape test** A common designation for an ink adhesion test, which consists of applying a piece of adhesive tape to the surface under test, pulling it free, and examining the tape to see whether any ink adheres to it.

**Target of evaluation (TOE)** The information technology system to be assessed in an → *evaluation*.<sup>107</sup> For example, the target of evaluation could be a → *smart card microcontroller* with integrated software that must fulfill certain → *security targets*.

<sup>103</sup> See also Section 9.3.1, ‘The T = 0 transmission protocol’, on page 255

<sup>104</sup> See also Section 9.3.2, ‘The T = 1 transmission protocol’, on page 260

<sup>105</sup> See also Chapter 10, ‘Contactless Data Transmission’, on page 283

<sup>106</sup> See also Section 6.1, ‘Data Structures’, on page 109

<sup>107</sup> See also Section 15.5, ‘Evaluation of Hardware and Software’, on page 659

**TCOS (Telesec Chipcard Operating System)** A → *smart card operating system* produced by T-Systems.

**TCP/IP** The TCP (Transmission Control Protocol) and IP (Internet Protocol) are the main protocols of the TCP/IP architecture and form part of the basis for Internet communication. TCP is a reliable circuit-switched protocol that enables error-free communication between two computers, and it belongs to the transport layer. In terms of the layer model, IP is located in the layer below TCP and belongs to the Internet layer. One of the essential functions of IP is unique addressing and grouping of computers in a network, which in turn forms the basis for routing data packets.

**TCSEC (Trusted Computer System Evaluation Criteria)** A criteria catalog for the development and → *evaluation* of the security of information technology systems in the US domain, published in 1983 by the National Computer Security Center (→ NCSC). The successor to the nationally oriented TCSEC is the internationally applicable → *Common Criteria*.

**TD/CDMA (time division/code division multiple access)** → *CDMA*

**TDES** Another name for → *triple DES*.

**TDMA (time division multiple access)** A multiple access method for the quasi-concurrent transmission of data from multiple transmitters to a receiver using a single frequency. For this purpose, each transmitter is allocated a specific time slot for its exclusive use, which requires very precise synchronization. TDMA is used together with FDMA in GSM systems for the air interface between mobile stations and base stations.

**Terminal** As a counterpart to a smart card, a terminal<sup>108</sup> is a device that supplies electrical power to a smart card and enables data exchange with the card, and which may also have a keypad and a display. The standard ISO designation for a smart card terminal is ‘interface device’ (IFD), while in the payment sector the standard designation for a terminal is ‘card accepting device’ (CAD). Terminals are sometimes called → *card readers*.

**Test** A systematic procedure used to check whether an already debugged program functions properly and fulfills the requirements determined in the → *analysis* phase. The primary objective is not searching for defects, but instead testing the required functions. Testing is thus not the same as → *debugging*.

**TETRA (Terrestrial Trunked Radio; formerly Trans-European Trunked Radio)** Digital trunked radio system defined by an → *ETSI* specification that operates in the 380–420-MHz band in TDMA mode. A security module usually called TETRA SIM, similar to the → *SIM* used in the → *GSM* system, is specified for use in TETRA. However, the TETRA SIM is optional and can be implemented in software in the mobile station if desired.

**TETRA SIM** → *TETRA*

**Thread** → *multithreading*

**Three-factor authentication** Three-factor authentication combines all three forms of authentication (authentication by possession, by knowledge, and by a biometric feature).

<sup>108</sup> See also Chapter 17, ‘Smart Card Terminals’, on page 735

This means that three different types of authentication must be performed with positive results in order to achieve successful authentication.

**Time stamp** An attestation with a digital signature, issued by an authority, that confirms that certain digital data was provided to the authority at a certain time (→ *PKI*).<sup>109</sup>

**Timing attack** A form of → *attack* on a → *cryptographic algorithm* that exploits the dependence of the cryptographic algorithm on the secret key in order to determine the value of the key. The countermeasure is to use noise-free cryptographic algorithms whose processing time is independent of the key.

**TLV format** A term commonly used to designate → *ASN.1* BER-coded data objects,<sup>110</sup> in which a value is uniquely described by means of a prefixed tag and a length value.

**Toolchain** The entire set of tools used in software development, starting with program generation and source code management and extending over the → *compiler* and → *linker* to the → *simulator* and → *emulator*. In some cases, the tools used for requirements management and change management are also included in the toolchain.

**Top-level CA** → *certification authority*

**TPDU** → *APDU*

**Tracer** A combination of software and associated hardware used to examine communication processes in data transmission. In contrast to a → *sniffer*, a tracer requires hardware as well as software.

**Transaction** A sequence of → *commands* sent to a smart card in order to perform a specific task. A typical example of a transaction is the command sequence for loading an electronic purse.

**Transaction number (TAN)** In contrast to a PIN, a TAN is valid for only one transaction and thus can be used only once. The user normally receives a group of TANs consisting of a list of five-digit numbers printed on paper, which usually must be used for individual transactions or sessions in the order listed. With the indexed transaction number (indexed TAN) variant, the background system sends the user a number that identifies which TAN must be used for the current transaction. The TAN technique provides a certain amount of protection if a potential attacker knows only one transaction number or only a small set of transaction numbers.

**Transfer card** A → *smart card* that is used to transmit data between two entities. For this purpose, it has a large data memory and, as a rule, keys for an authentication process that checks whether the entity concerned is allowed to write or read the data to be transferred.

**Transient** A property of an object that exists only for the duration of the execution of a process; the opposite of → *persistent*

**Transmission protocol** In the smart card world, the mechanisms used to send and receive data between the terminal and the smart card.<sup>111</sup> A transmission protocol describes in

<sup>109</sup> See also Section 23.1.6, ‘Trust center’, on page 905

<sup>110</sup> See also Section 6.1, ‘Data Structures’, on page 109

<sup>111</sup> See also Chapter 8, ‘Communication with Smart Cards’, on page 201

detail the OSI protocol layers that are used, the data exchange process in the absence of errors, error detection mechanisms, and error response mechanisms.

**Transport protocol** An alternative term for → *transmission protocol*.<sup>112</sup>

**Trap door** A mechanism intentionally included in a computer program or an algorithm to allow security functions or protective mechanisms to be bypassed in order to gain access to data or programs.

**Triple DES** A modified form of DES encryption, also designated TDES and 3DES, in which the DES algorithm is used three times in succession with alternating encryption and decryption.<sup>113</sup> If the same key is used for all three DES operations, triple DES is equivalent to normal DES encryption, but if two or three different keys are used, triple DES is considerably stronger than simple DES.

**Triple-band mobile telephone** A mobile telephone that can operate in three frequency bands, such as GSM 900, GSM 1800, and GSM 1900.

**Triple-interface card** Designates a → *smart card* with three different interfaces for data transfer to and from the card. An example is a smart card with an ISO interface (→ *ISO protocol*), a → *USB* interface, and an → *SWP* interface.

**Trivial PIN** A PIN that can easily be guessed, such as 1234.<sup>114</sup> (*null PIN*)

**Trojan horse** In the historical sense, the wooden horse that enabled Odysseus to gain access to the heavily defended city of Troy by deceitful means.<sup>115</sup> In modern usage, a Trojan horse is a program that apparently performs a particular function but can also perform hidden functions. Trojan horses are intentionally introduced into computer systems or commercial software, and unlike viruses they cannot reproduce themselves.

**Trust center (TC)** One of the two essential components of a public key infrastructure (→ *PKI*) (the other being the → *signature cards*), which generates, distributes and manages certificates. Depending on the system implementation, the trust center can also assume the responsibilities of the → *certification authority*, → *registration authority*, → *directory service*, and → *time-stamp* service.

**Trusted third party (TTP)** An entity that is regarded as trustworthy by two or more other entities. A typical activity of a trusted third party is to issue → *certificates*.

**TSL → SSL**

**Tunnel** A cryptographically secured → *end-to-end connection* between two entities that employs the communication channels of one or more other entities that do not modify the information content of the data actually exchanged.

**Two-factor authentication** Two-factor authentication combines two of the three different forms of authentication (authentication by possession, by knowledge, or by a biometric feature). This means that two different types of authentication must be performed with positive results in order to achieve successful → *authentication*.

<sup>112</sup> See also Chapter 9, ‘Data Transmission with Contact Cards’, on page 243

<sup>113</sup> See also Section 7.1.1.7, ‘Multiple encryption’, on page 144

<sup>114</sup> See also Section 7.8.2, ‘Testing a secret number’, on page 188

<sup>115</sup> See also Section 16.5.2, ‘Attacks on the operating system’, on page 712

**UART (universal asynchronous receiver/transmitter)** A general-purpose component for transmitting and receiving data, which operates asynchronously and independently of the → *microprocessor*. With a UART, the microprocessor does not have to handle communication at the bit and byte level. This leads to a simplification of the communication protocols, and it can also be used to achieve higher data transmission rates than with a pure software implementation using the microprocessor.

**UATK (R-UIM application toolkit)** A modular extension to the → *R-UIM*, similar to the → *SIM Application Toolkit*, that enables the R-UIM to assume an active role in controlling the mobile telephone.

**UCS (Universal Character Set)** A character coding scheme that represents an extension of → *ASCII* and → *Unicode*. It is specified in the ISO/IEC 10646 standard. UCS uses 32-bit character codes, although only half of the possible address space is used ( $2^{32}/2=2\,147\,483\,648$ ). This address space is large enough to encode all characters of all languages in the world. UCS is defined such that → *Unicode* is a subset of UCS and the coding of the first 128 characters is the same as the ASCII code.

**UICC (universal integrated chip card)** A → *smart card* with a → *smart card operating system* that is compliant with ISO/IEC 7816 and optimized for smart card → *applications* in the telecommunication sector. The UICC is standardized by the TS 102 221 specification published by → *ETSI*. The UICC forms the basis for the → *USIM*. It is normally implemented as a plug-in card in → *ID-000* or → *mini-UICC* format.

**UIM (user identity module)** An outdated term for → *USIM*.

**UML (Unified Modeling Language)** A graphically based general-purpose modeling language used to abstractly describe static and dynamic aspects of object-oriented programs. The basic notation and semantics of UML were defined in the 1990s by Grady Booch, James Rumbaugh, and Ivar Jacobson. UML is independent of any particular → *life cycle model* for software development. The Object Management Group (OMG) [OMG] is responsible for extending and enhancing UML.

**UMTS (Universal Mobile Telecommunication System)** The European successor to → *GSM* and a member of the → *IMT-2000* family. It is a digital, cellular, third-generation (→ *3G*) mobile telecommunication system that is neither country-specific nor operator-specific. The designated operating frequency of this mobile telecommunication system is in the 2000-MHz band. UMTS is defined by a set of standards generated under the authority of the → *3GPP* and published by → *ETSI*. It is the next major evolutionary step after → *GSM*. The key changes with respect to *GSM* are a new air interface using → *CDMA* technology and significantly higher data transmission rates.

**Unicode** Unicode<sup>116</sup> [Unicode] is an extension of the well-known ASCII character code. In contrast to the seven-bit ASCII code, Unicode uses 16-bit character codes. This enables it to support the character sets of the world's most commonly used languages. The first 256 characters of Unicode are the same as ISO 8859-1 ASCII.

**Uplink** A connection from a lower-level system (such as a mobile telephone) to a higher-level system (such as a base station); the opposite of → *downlink*.

<sup>116</sup> See also Section 6.2.3, ‘Sixteen-bit code’, on page 116

**Upload** The process of transferring data from a lower-level system (such as a terminal) to a higher-level system (such as a background system or host system); the opposite of → *download*.

**URL (uniform resource locator)** A unique alphanumeric address on the → *WWW*.

**USB (Universal Serial Bus)** A serial bus with two differential data lines, used to connect a computer to external devices, which supports a data transmission rate of 1.5 Mbit/s (Low Speed mode), 12 Mbit/s (Full Speed mode), or 480 Mbit/s (High Speed mode).<sup>117</sup>

**User data** All data directly necessary for an → *application*.

**User** A person who uses a → *smart card*; not necessarily the same as the → *cardholder*.

**USIM (universal subscriber identity module)** Customary designation of the smart card → *application* for → *UMTS*, which is located on a → *UICC*.<sup>118</sup> However, in practice USIM refers to the UMTS smart card instead of the application, even though this is not entirely correct. The USIM is the bearer of the subscriber identity, and its main function is to secure the authenticity of the mobile telephone with respect to the network and the other way round. Additional functions include running programs with protection against manipulation, user identification (using a PIN), and storing data such as telephone numbers. The USIM is standardized by TS 31.102, which is published by → *ETSI*. The equivalent of the USIM in the → *GSM* system is the → *SIM*.

**USIM Application Toolkit (USAT)** The USIM Application Toolkit allows a USIM card to assume an active role in the control of a mobile telephone. For example, it can enable the USIM to output messages or images to the display, request input from the keyboard, or send and receive messages over the air interface. The USIM Application Toolkit forms the basis for most value-added services in mobile telephones. The equivalent of the USIM Application Toolkit in the GSM system is the → *SIM Application Toolkit*.

**Validation** Evaluation of the assessment of measures with regard to their effectiveness.

**Value-added service (VAS)** A supplementary → *application* in a → *smart card*, in addition to the main application. This is usually only possible with → *multiapplication smart cards*.

**Vertical prototype** → *prototype*

**Virgin card** A card not yet fitted with a chip or personalized (visually and/or electrically).

A virgin card is essentially a printed, standard → *card body* resulting from the mass production process for cards.

**Virtual machine (VM)** A software simulation of a → *microprocessor*, which usually has its own opcodes for machine instructions and a simulated address space. This enables the generation of hardware-independent software. The virtual address space of a VM may be many times larger than the physical address space available in hardware. In the → *Java* realm, the closed environment of the virtual machine is often called the ‘sandbox’.

**Virtual merchant card** → *virtual smart card*

<sup>117</sup> See also Section 9.4, ‘USB Transmission Protocol’, on page 272

<sup>118</sup> See also Section 19.5, ‘The UMTS System’, on page 848

**Virtual smart card** A software simulation of a smart card in another system, such as a → *security module* or a mobile telephone. A virtual merchant card is a specific instance of a virtual smart card in which the smart card simulation runs in a merchant terminal.

**Visa** An official confirmation that a person is allowed to enter, leave, or remain in a foreign country. The conditions for obtaining visas vary from one country to the next.

**Visa Cash** A Visa Inc. brand name for a variety of electronic purse systems using smart cards, which differ in their technical features.

**Visa Easy Entry (VEE)** A method for easy migration from magnetic-stripe → *credit cards* to credit cards with → *microprocessor* chips. For this purpose, the name of the → *cardholder* and all of the magnetic-stripe data is stored in an → *EF* in a DF reserved for Visa. When a purchase is made using the credit card, the → *terminal* reads the data necessary for the → *transaction* from the → *chip* instead of the → *magnetic stripe*. The advantage of this method is that it only requires upgrading the → *POS* terminal to include a smart card contact unit, while the entire → *background system* can remain unchanged.

**Visa Mini** A card format specified by Visa, based on the → *ID-1* format.<sup>119</sup> It measures 40 by 65.6 mm with a thickness of 0.76 mm.

**Volatile memory** A type of memory (such as RAM) that retains its contents only as long as power is applied.

**VOP** → *OP*

**Wafer** A thin disk of silicon that serves as a substrate for the fabrication of semiconductor devices (chips). Typical wafer diameters are 150 mm (6 inch), 200 mm (8 inch), and 300 mm (12 inch).

**WAP (Wireless Application Protocol)** WAP [WAP] designates a set of specifications for connecting mobile equipment to the Internet via a mobile telecommunication system. The WAP standards were generated by the WAP Forum, which was established in June 1999 and later renamed Open Mobile Alliance [OMA].

**Warm reset** → *reset*

**WCDMA (wideband code division multiple access)** → *CDMA*

**Wear leveling** Designates methods used to increase the service life of nonvolatile memory (such as → *EEPROM* and → *flash memory*) with a limited number of write/erase cycles.<sup>120</sup> The basic strategies are to distribute the wear due to write/erase cycles as uniformly as possible over the total available amount of nonvolatile memory and to mark defective memory blocks and prevent them from being used.

**White plastic** Blank cards that have not yet been personalized and are used with deceitful intent. The term originates from the typical appearance of blank cards used to produce test cards. However, it now refers equally well to printed cards with a wide variety of → *card components*, such as blank credit cards with magnetic stripes and holograms that have not yet been embossed or smart cards with freely programmable microcontrollers.

<sup>119</sup> See also Section 3.1, ‘Card Formats’, on page 29

<sup>120</sup> See also Section 13.6.3, ‘Flash memory management’, on page 461

**Whitebox test** Designates a test in which it assumed that the party performing the test has full knowledge of all internal processes and data of the software under test. Also called ‘glassbox test’.

**Whitelist** A database list of all smart cards or devices that are allowed to be used in a particular → *application*. (→ *blacklist, hotlist, greylist*)

**WIB browser** → *browser*

**WIM (Wireless Application Protocol identity module)** A security module for mobile equipment that supports the Wireless Application Protocol (→ *WAP*).<sup>121</sup> The specification describes a smart card → *application* compatible with → *PKCS#15*. The main functions of a WIM are generating and verifying → *digital signatures* and encrypting data. A WIM may be either a separate, physical smart card or one of several applications in a multiapplication smart card. It is typically an application in a → *SIM* or → *USIM*.

**Windows for Smart Cards (WfSC, WSC)** An → *smart card operating system* developed by Microsoft [Microsoft] that supports multiple → *applications* and downloadable programs. One of the unusual features of Windows for Smart Cards is that it uses a → *FAT*-based file system.

**WML (Wireless Markup Language)** A logical markup language based on → *XML* that is used to generate applications for → *WAP*. WML is very similar to → *HTML*. WML applications held on a WML site in a WAP server are translated on the fly by a converter into compact WML bytecode, which is transmitted via the wireless network to the mobile telephone where it is interpreted by a microbrowser (→ *browser*).

**Workaround** In the context of software development, code that circumvents a known defect. A workaround avoids the negative impact of a defect on the rest of the program, but it does not eliminate the defect. For example, defects found in a ROM-based → *smart card operating system* are typically corrected by workarounds stored in EEPROM. It is entirely possible for the functionality of the operating system to be degraded as a consequence of using a workaround.

**WWW, W3 (World Wide Web)** Part of the worldwide Internet, primarily characterized by the capability of using hyperlinks to link various documents as desired and the inclusion of multimedia objects in documents.

**X.509** The X.509 standard published by the → *ITU* defines the structure and coding of → *certificates*. It is the most widely used worldwide standard for certificate structures (→ *PKI*).

**XML (Extended Markup Language)** A logical markup language that is both a successor to and an extension of → *HTML*. XML can be used to define new language elements, which means that other markup languages, such as HTML and WML, can be defined using XML. XLM is a subset of the powerful Standard Generalized Markup Language (SGML), which is specified by an ISO standard.

**ZKA (Zentraler Kreditausschuss)** The coordinating body for the electronic payment transactions of German banks. The ZKA is composed of the following associations:

<sup>121</sup> See also Section 19.6, ‘The WIM’, on page 854

Deutscher Sparkassen- und Giroverband (DSGV), Bundesverband der Deutschen Volksbanken und Raiffeisenbanken (BVR), Bundesverbund deutscher Banken (BdB), and Verbund öffentlicher Banken (VÖB). The four banking associations assume the chairmanship of the ZKA in annual rotation.

## 25.2 RELATED READING

The *Smart Card Handbook* focuses primarily on smart cards and their applications. However, there are many other disciplines that strongly affect smart cards and their environment, each of which has its own particular areas of interest and specialist literature. The aim of the authors of the *Smart Card Handbook* is to maintain the focus of this book within the smart card domain, rather than dealing extensively with other related fields, which would anyhow exceed the scope of this book. For readers who wish to increase their knowledge of these related subjects, we have prepared the following short list of related reading (see Table 25.1).

**Table 25.1** Recommended literature in fields related to smart cards

Subject	Reference
Smart card applications	[Rankl 06]
Operating systems	[Tanenbaum 02]
Smart card production	[Haghiri 02]
The Java programming language	[Arnold 00]
Cryptography	[Menezes 97], [Schneier 96]
RFID	[Finkenzeller 06]
SPA/DPA analysis	[Mangard 07]
Security of components and systems	[Anderson 01]
Software development	[Balzert 98]
Software development for Java Card	[Chen 00]

## 25.3 BIBLIOGRAPHY

The following publications are sorted first by the surname of the author and then in ascending order of publication date. ‘Internet’ is listed as the source of publications that appeared in newsgroups or discussion forums on the Internet.

- AIS 20 German Federal Office for Information Security [BSI], ‘Anwendungshinweise und Interpretationen zum Schema (AIS), Funktionalitätsklassen und Evaluationsmethodologie für deterministische Zufallszahlengeneratoren’ [‘Application Tips and Interpretations regarding the Scheme (AIS), Functionality Classes and Evaluation Methodology for Deterministic Random Number Generators’], Bonn 1999
- AIS 31 German Federal Office for Information Security [BSI], ‘Anwendungshinweise und Interpretationen zum Schema (AIS), Funktionalitätsklassen und Evaluationsmethodologie für physikalische Zufallszahlengeneratoren’

- [‘Application Tips and Interpretations regarding the Scheme (AIS), Functionality Classes and Evaluation Methodology for Physical Random Number Generators’], Bonn 2001
- Abelson 97 Hal Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, ‘The Risks of Key Recovery, Key Escrow, and Trusted Third-Party Encryption’, [www.crypto.com](http://www.crypto.com) 1997
- Anderson 01 Ross J. Anderson, *Security Engineering*, John Wiley & Sons, Ltd, Chichester 2001
- Anderson 92 Ross J. Anderson, *Automatic Teller Machines*, Internet December 1992
- Anderson 95 Ross J. Anderson, Roger M. Needham, ‘Programming Satan’s Computer’, *Computer Science Today*, [www.computersciencetoday.com](http://www.computersciencetoday.com) 1995
- Anderson 96a Ross J. Anderson, Markus G. Kuhn, ‘Improved Differential Fault Analysis’, Internet, November 1996
- Anderson 96b Ross J. Anderson, Markus G. Kuhn, ‘Tamper Resistance – a Cautionary Note’, USENIX Workshop, November 1996
- Arnold 00 Ken Arnold, James Gosling, David Holmes, *The Java Programming Language*, 3rd edn, Addison-Wesley, Boston 2000
- BIS 96 Bank for International Settlements, ‘Security of Electronic Money – Report by the Committee on Payment and Settlement Systems and the Group of Computer Experts of the Central Banks of the Group of Ten Countries’, Basel, August 1996
- BSI 05 German Federal Office for Information Security, ‘Untersuchung der Leistungsfähigkeit von biometrischen Verifikationssystemen BioP II’ [‘Study of the Performance Capabilities of Biometric Verification Systems (BioP II)’], Final Public Report, Version 2.0, 23 August 2008, Bonn
- Balzert 98 Helmut Balzert, *Lehrbuch der Software-Technik*, vol. 2, 2nd edn, Spektrum Akademischer Verlag, Heidelberg 1998
- Bar-El 04 Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, Claire Whelan, ‘The sorcerer’s apprentice guide to fault attacks’, Workshop on Fault Detection and Tolerance in Cryptography, Florence, Italy, 30 June, 2004
- Bellare 95 Mihir Bellare, Philip Rogaway, ‘Optimal Asymmetric Encryption – How to encrypt with RSA’, Internet 1995
- Bellare 96 Mihir Bellare, Philip Rogaway, ‘The Exact Security of Digital Signatures – How to Sign with RSA and Rabin’, Internet 1996
- Biham 91 Eli Biham, Adi Shamir, ‘Differential Cryptoanalysis of DES-like Cryptosystems’, *Journal of Cryptology*, vol. 4 no. 1, 1991
- Biham 93 Eli Biham, Adi Shamir, *Differential Cryptoanalysis of the Data Encryption Standard*, Springer, New York 1993
- Biham 96 Eli Biham, Adi Shamir, ‘A new cryptoanalytic attack on DES’, Internet 1996

- Boehm 81 Barry W. Boehm, *Software Engineering Economics*, Prentice Hall, Upper Saddle River, New Jersey 1981
- Bogdanov 07 Andrey Bogdanov, ‘Cryptanalysis of the KeeLoq block cipher’, Internet 2007
- Bögeholz 99 Harald Bögeholz, Dusan Zivadinovic, ‘Telefon-Zellen’, *c't* 18, 1999
- Boneh 96 Dan Boneh, Richard A. DeMillo, Richard J. Lipton, ‘On the Importance of Checking Computations’, Math and Cryptography Research Group, Bellcore 1996
- Bono 05 Steve Bono, Matthew Green, Adam Stubblefield, Ari Juels, Avi Rubin, Michael Szydlo, ‘Security Analysis of a Cryptographically-Enabled RFID Device’, The Johns Hopkins University Information Security Institute, Baltimore, 28 January 2005
- Bronstein 97 I.N. Bronstein, K.A. Semendjajew, *Taschenbuch der Mathematik*, 7th edn, B.G. Teubner Verlagsgesellschaft, Leipzig 1997
- Buchmann 96 Johannes Buchmann, ‘Faktorisierung großer Zahlen, *Spektrum der Wissenschaft*’, September 1996
- Chen 00 Zhiqun Chen, *Java Card Technology for Smart Cards*, Addison-Wesley, Boston 2000
- Datenschutz 96 Hamburg Data Protection Commissioner, ‘Anforderungen zur informationstechnischen Sicherheit bei Chipkarten’ [‘Requirements for Informational Security with Smart Cards’], Hamburg 1996
- Dhem 96 J.-F. Dhem, J.-J. Quisquater, R. Lecat, ‘Lossless Compression Algorithms for Smart Cards, A Progress Report’, UCL 1996
- Diffie 76 Whitfield Diffie, Martin E. Hellman, ‘New Directions in Cryptography’, Internet 1976
- Douin 04 J.-M. Douin, P. Paradinas, C. Pradel, ‘Open Benchmark for Java Card Technology’, paper presented at e-Smart 2004
- EC 98 Council of the European Communities,  
Council Regulation (EC) No. 2135/98 of 24 September 1998 Amending Regulation (EEC) No. 3821/85 on recording equipment in road transport and Directive 88/599/EEC concerning the application of Regulations (EEC) No. 3820/85 and (EEC) No. 3821/85
- EU 95 European Parliament and the European Council,  
Directive 95/46/EC of the European Parliament of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data, Official Journal L 281 of 23 November 1995
- Eberspächer 00 Jörg Eberspächer, Hans-Jörg Vögel, Christian Bettstetter, *GSM – Global System for Mobile Communication*, B.G. Teubner, Stuttgart 2000
- Fenton 96 Norman E. Fenton, Shari Lawrence Pfleeger, *Software Metrics*, Thomson Computer Press, London 1996
- Finkenzeller 06 Klaus Finkenzeller, *RFID Handbook*, 2nd edn, John Wiley & Sons, Ltd, Chichester 2003

- Franz 98 Michael Franz, ‘Java – Anmerkungen eines Wirth-Schülers’, *Informatik Spektrum*, Springer, Berlin 1998
- Gamma 04 Erich Gamma, Richard Helm, Ralph E. Johnson, *Design Patterns*, Addison-Wesley, New York 2002
- Gandolfi 01 Karine Gandolfi, Christophe Mourtel, Francis Oliver, ‘Electromagnetic Analysis, Concrete Results, Cryptographic Hardware and Embedded Systems’, CHES 2001, Springer, Berlin/Heidelberg 2001
- Garstka 03 Hansjürgen Garstka, ‘Informationelle Selbstbestimmung und Datenschutz’, in Christiane Schulzki-Haddouti, *Bürgerrechte im Netz*, Bundeszentrale für politische Bildung, Bonn 2003
- Gentz 97 Wolfgang Gentz, *Die elektronische Geldbörse in Deutschland*, graduation dissertation submitted to the Munich University of Technology, Munich 1997
- Gora 98 Walter Gora, *ASN.1 – Abstract Syntax Notation One*, 3rd edn, Fossil, Cologne 1998
- Gosling 95 James Gosling, Henry McGilton, ‘The Java Language Environment – A White Paper’, Sun Microsystems, USA 1995
- Guthery 02 Scott B. Guthery, Mary J. Cronin, *Mobile Application Development with SMS and the SIM Toolkit*, McGraw-Hill, New York 2002
- Gutmann 96 Peter Gutmann, ‘Secure Deletion of Data from Magnetic and Solid-State Memory’, USENIX Conference, San José, California 1996
- Gutmann 98a Peter Gutmann, ‘Software Generation of Practically Strong Random Numbers’, Internet 1998
- Gutmann 98b Peter Gutmann, ‘X.509 Style Guide’, Internet 1998
- Haghiri 02 Yahya Haghiri, Thomas Tarantino, *Smart Card Manufacturing: A Practical Guide*, John Wiley & Sons, Ltd, Chichester 2002
- Hancke 05 Gerhard P. Hancke, ‘A Practical Relay Attack on ISO 14443 Proximity Cards’, University of Cambridge Computer Laboratory, Internet, February 2005
- Hassler 02 Vesna Hassler, Martin Manninger, Mikhail Gordeev, Christoph Muller, *Java Card for E-Payment Applications*, Artech House, London 2002
- Hellmann 79 Martin E. Hellmann, ‘The Mathematics of Public-Key Cryptography’, *Scientific American*, August 1979
- Heydt-Benjamin 06 Thomas S. Heydt-Benjamin, Daniel V. Bailey, Kevin Fu, Ari Juels, Tom O’Hare, ‘Vulnerabilities in First-Generation RFID-enabled Credit Cards’, *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)*, Cambridge, Springer 2006
- Hillebrand 2002 Friedhelm Hillebrand (ed.), *GSM and UMTS*, John Wiley & Sons, Ltd, Chichester 2002
- Hilleringmann 04 Ulrich Hilleringmann, *Silizium-Halbleitertechnologie*, 4th edn, B.G. Teubner, Stuttgart 2004
- Horster 99 Patrik Horster, Dirk Fox (eds), *Datenschutz und Datensicherheit*, Vieweg, Braunschweig 1999

- Hunt 03 Andrew Hunt, David Thomas, *The Pragmatic Programmer*, Addison-Wesley 1999
- IC Protection 97 Common Criteria for IT Security Evaluation Protection Profile – Smart-card Integrated Circuit Protection Profile, Internet 1997
- Intel 98 Intel, ‘Understanding the Flash Translation Layer (FTL) Specification’, Application Note AP-684, December 1998
- Isselhorst 97 H. Isselhorst, ‘Betreiberorientierte Sicherheitsanforderungen für Chipkarten-Anwendungen’, *Card-Forum*, Lüneburg 1997
- Jacobs 05 Bart Jacobs, Ronny Wickers Schreur, ‘Biometric Passport’, Radboud University Nijmegen, Safe NL Workshop 2005
- Janke 02 Marcus Janke, Peter Laackmann, ‘Renaissance der physikalischen Angriffe’, *Card-Forum*, Lüneburg 2002
- Janke 03 Marcus Janke, Peter Laackmann, ‘Chipkarten unter Beschuss’, *Card-Forum*, Lüneburg 2003
- Jones 91 C. Jones, *Applied Software Measurement*, McGraw-Hill, New York 1991
- Jun 99 Benjamin Jun, Paul Kocher, ‘The Intel Random Number Generator’, Internet 1999
- Kaliski 93 Burton S. Kaliski Jr., ‘A Layman’s Guide to a Subset of ASN.1, BER and DER’, RSA Laboratories Technical Note, Internet 1993
- Kaliski 96 Burton S. Kaliski Jr., ‘Timing Attacks on Cryptosystems’, RSA Laboratories, Redwood City 1996
- Kfir 05 Z. Kfir, A. Wool, ‘Picking Virtual Pockets using Relay Attacks on Contactless Smartcard Systems’, Securecomm 05, 2005
- Knapp 97 Fritz Knapp, ‘Zur Sicherheit der ec-Karte PIN: Das Urteil des OLG Hamm’, *Karten*, Frankfurt, August 1997
- Knuth 97 Donald Ervin Knuth, *The Art of Computer Programming, vol. 2: Seminumerical Algorithms*, 3rd edn, Addison Wesley Longman, Reading, Massachusetts 1997
- Kocher 95 Paul C. Kocher, ‘Timing Attacks on Implementations of Diffie–Hellmann, RSA, DSS, and Other Systems’, Internet 1995
- Kocher 98 a Paul C. Kocher, Joshua Jaffe, Benjamin Jun, ‘Introduction to Differential Power Analysis and Related Attacks’, Internet 1998
- Kocher 98 b Paul C. Kocher, Joshua Jaffe, Benjamin Jun, ‘Differential Power Analysis: Leaking Secrets’, Internet 1998
- Krissler 08 Jan Krissler, Karsten Nohl, Henryk Plötz, ‘Chiptease’, *c·t* 8, 31 March 2008
- Kriterien 05 German Regulatory Authority for Telecommunication and Postal Services, ‘Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (Übersicht über geeignete Algorithmen)’ [‘Notification Regarding Electronic Signatures in Accordance with the Signature Act and the Signature Ordinance (Overview of Suitable Algorithms)’], 2 January 2005
- Kuhn 97 Markus G. Kuhn, ‘Probability Theory for Pickpockets – ec-PIN Guessing’, COAST Laboratory, Purdue University, West Lafayette, Indiana 1997

- Kuhn Markus G. Kuhn, ‘Attacks on Pay-TV Access Control Systems’, University of Cambridge, Internet, year unknown
- Kömmerling 99 Oliver Kömmerling, Markus G. Kuhn, ‘Design Principles for Tamper-Resistant Smartcard Processors’, USENIX Workshop on Smartcard Technology, Chicago USA, 10–11 May 1999
- Lamla 00 Michael Lamla, ‘Hardware Attacks on Smart Cards – Overview’, Eurosmart Security Conference, Marseille, 13–15 June 2000
- Lamport 81 Leslie Lamport, ‘Password Authentication with Insecure Communication’, *Communications of the ACM* 24, 11, San Diego, California 1981
- Leiberich 99 Otto Leiberich, ‘Vom diplomatischen Code zur Falltürfunktion’, *Spektrum der Wissenschaft*, June 1999
- Lender 96 Friedwart Lender, ‘Production, Personalisation and Mailing of Smart Cards – A Survey’, Smart Card Technologies and Applications Workshop, Berlin, November 1996
- Levy 99 Steven Levy, ‘The Open Secret’, *Wired*, April 1999
- Liggesmeyer 02 Peter Liggesmeyer, ‘Software-Qualität’, *Spektrum*, Heidelberg 2002
- Lindholm 97 Tim Lindholm, Frank Yellin, *The Java Virtual Machine Specification*, 2nd edn, Addison-Wesley, Reading 1997
- Maltoni 05 Davide Maltoni, Dario Maio, Anil K. Jain, Salil Prabhakar, *Handbook of Fingerprint Recognition*, 2nd edn, Springer Science + Business Media, USA 2005
- Mangard 07 Stefan Mangard, Elisabeth Oswald, Thomas Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, Springer, Berlin 2007
- Massey 88 James L. Massey, ‘An Introduction to Contemporary Cryptology’, *Proceedings of the IEEE*, vol. 76 no. 5, pp 533–549, IEEE 1988
- Massey 97 James L. Massey, *Cryptography, Fundamentals and Applications*, 1997
- McConnell 05 Steve McConnell, *Code Complete*, 2nd edn, Microsoft Press 2004
- Meister 95 Giesela Meister, Eric Johnson, ‘Schlüsselmanagement und Sicherheitssprotokolle gemäß ISO/SC 27 – Standards in Smart Card-Umgebungen’, in: Albert Glade, Helmut Reimer, Bruno Struif, *Digitale Signatur*, Vieweg, Braunschweig 1995
- Menezes 93 Alfred J. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishing, Boston 1993
- Menezes 97 Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton 1997
- Merkle 81 Ralph C. Merkle, Martin E. Hellman, ‘On the Security of Multiple Encryption’, Internet 1981
- Messerges 99 Thomas S. Messerges, Ezzy A. Dabbish, Robert H. Sloan, ‘Investigations of Power Analysis Attacks on Smartcards’, USENIX Workshop on Smartcard Technology, Chicago USA, 10–11 May 1999
- Meyer 82 Carl H. Meyer, Stephen M. Matyas, *Cryptography*, John Wiley & Sons, Inc., New York 1982

- Meyer 96 Carsten Meyer, ‘Nur Peanuts – Der Risikofaktor Magnetkarte’, *c’t*, July 1996
- Montenegro 99 Sergio Montenegro, *Sichere und fehlertolerante Steuerungen*, Carl Hanser, Munich/Vienna 1999
- Moore 02 Simon Moore, Ross Anderson, Paul Cunningham, Robert Mullins, George Tayler, ‘Improving Smart Card Security using Self-timed Circuits’, Internet, May 2002
- Myers 95 Glenford J. Myers, *The Art of Software Testing*, 5th edn, John Wiley & Sons, Inc., New York 1995
- Müller-Maguhn 97a Andy Müller-Maguhn, ‘“Sicherheit” von EC-Karten’, *Die Datenschleuder* 53, 1997
- Müller-Maguhn 97b Andy Müller-Maguhn, ‘EC-Karten Unsicherheit’, *Die Datenschleuder* 59, 1997
- NIST 01 NIST, ‘A Statistical Test Suite for Random and Pseudorandom Number Generator for Cryptographic Applications’, Special Publication 800-22
- Nebelung 96 Brigitte Nebelung, ‘Das Geldbörsen-Konzept der ec-Karte mit Chip’, de-bis Systemhaus, Bonn 1996
- Nechvatal 00 James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, James Foti, Morris Dworkin, Edward Roback, ‘Report on the Development of the Advanced Encryption Standard (AES)’, NIST, Internet 2000
- Odlyzko 95 Andrew. M. Odlyzko, ‘The future of integer factorization’, AT&T Bell Laboratories, 1995
- Otto 82 Siegfried Otto, ‘Echt oder falsch? Die maschinelle Echtheitserkennung’, *Betriebswirtschaftliche Blätter*, vol. 2, February 1982
- PP 06 Sun, ‘Java Card System Protection Profile Collection’, Version 1.1, Santa Clara, California, 2006
- Peyret 97 Patrice Peyret, ‘Which Smart Card technologies will you need to ride the Information Highway safely?’, Gemplus, 1997
- Pomerance 84 C. Pomerance, ‘The Quadratic Sieve Factoring Algorithm’, *Advances in Cryptology*, Proceedings of Eurocrypt 84
- Press 92 William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, *Numerical Recipes in C – The Art of Scientific Computing*, 2nd edn, Cambridge University Press, Cambridge 1992
- Rankl 06 Wolfgang Rankl, *Smart Card Applications – Design models for programming and using smart cards*, John Wiley & Sons, Ltd, Chichester 2007
- Reed 60 Irving S. Reed, Gustave Solomon, ‘Polynomial Codes over Certain Finite Fields’, 1960
- Rees 99 Jim Rees, Peter Honeyman, ‘Webcard: a Java Card Web Server’, CITI Technical Report 99-3, University of Michigan, Ann Arbor 1999
- Rescorla 01 Eric Rescorla, *SSL and TLS*, Addison-Wesley, Boston 2001
- Rivest 78 Ronald L. Rivest, Adi Shamir, Leonard Adleman, ‘Method for obtaining Digital Signatures and Public-Key Cryptosystems’, Internet 1976

- Robertson 96 James Robertson, Suzanne Robertson, *Complete System Analysis*, Dorset House, 1994
- Rother 98a Stefan Rother, ‘Prüfung von Chipkarten-Sicherheit, *Card-Forum*, Lüneburg 1998
- Rother 98b Stefan Rother, ‘Prüfung von Chipkarten-Sicherheit, *Tagungsband Chipkarten*, Vieweg Verlag, Braunschweig 1998
- SAS 07 ‘Security Accreditation Scheme – Methodology’, Version 3.4.0, GSM Association, September 2007
- Schaar 02 Peter Schaar, *Datenschutz im Internet*, C.H. Beck, Munich 2002
- Schartner 01 Peter Schartner, *Security Token*, it Verlag, Höhenkirchen 2001
- Scherzer 00 Helmut Scherzer, ‘Chipkarten-Betriebssysteme – Gefahrenpotentiale und Sicherheitsmechanismen’, Smart Card IT Security Forum, 14 March 2000
- Schindler 97 Werner Schindler, ‘Wie sicher ist die PIN?’, presentation at the Credit Card Criminality Conference, Heppenheim, October 1997
- Schneier 96 Bruce Schneier, *Applied Cryptography*, 2nd edn, John Wiley & Sons, Inc., New York 1996
- Schneier 99 Bruce Schneier, ‘Attack Trees – Modeling Security Threats’, *Dr. Dobb’s Journal*, December 1999
- Sedgewick 97 Robert Sedgewick, *Algorithms*, 3rd edn, Addison-Wesley, Bonn Munich Reading 1997
- Sharma 03 Ashok K. Sharma, *Advanced Semiconductor Memories, Architectures, Designs and Applications*, IEEE Press and John Wiley & Sons, Inc., 2003
- SigG 01 Gesetz über Rahmenbedingungen für elektronische Signaturen (‘Act Regarding General Conditions for Digital Signatures’), 22 May 2001
- Silverman 97 Robert D. Silverman, ‘Fast Generation of Random, Strong RSA Primes’, RSA Laboratories Crypto Byte, Internet 1997
- Simmons 92 Gustavus J. Simmons (ed.), *Contemporary Cryptology*, IEEE Press, New York 1992
- Simmons 93 Gustavus J. Simmons, ‘The Subliminal Channels in the U.S. Digital Signature Algorithm’, *Proceedings of the Symposium on the State and Progress of Research in Cryptography*, Rome 1993
- Skorobogatov 02 Sergei Skorobogatov, Ross Anderson, ‘Optical Fault Induction Attacks’, Internet, May 2002
- Sommerville 90 Ian Sommerville, *Software Engineering*, Addison-Wesley, Wokingham 1990
- Spillner 03 Andreas Spillner, Tilo Linz, *Basiswissen Softwaretest*, dpunkt.verlag, Heidelberg 2003
- Steele 01 Raymond Steele, Chin-Chun Lee, Peter Gould, *GSM, cdmaOne and 3G Systems*, John Wiley & Sons, Ltd, Chichester 2001
- Stix 96 Gary Stix, ‘Herausforderung “Komma eins”’, *Spektrum der Wissenschaft*, February 1996

Tanenbaum 02	Andrew S. Tanenbaum, <i>Moderne Betriebssysteme</i> , 3rd edn, Addison-Wesley Longman, Reading, Massachusetts 2002
Thaller 93	Georg Erwin Thaller, <i>Qualitätsoptimierung der Software-Entwicklung, Das Capability Maturity Model (CMM)</i> , Vieweg, Braunschweig 1993
V-Modell XT	Federal Republic of Germany, V-Modell XT, www.v-modell-xt.de, 2004
Vedder 97	Klaus Vedder, Franz Weikmann, ‘Smart Cards – Requirements, Properties and Applications’, ESAT-COSIC Course, Catholic University of Leuven, 1997
Walke 00a	Bernhard Walke, <i>Mobilfunknetze und ihre Protokolle, B. 1: Grundlagen, GSM, UMTS und andere zellulare Mobilfunknetze</i> , B.G. Teubner, Stuttgart 2000
Walke 00b	Bernhard Walke, <i>Mobilfunknetze und ihre Protokolle, B. 2: Bündelfunk, schnurlose Telefonsysteme, W-ATM, HIPERLAN, Satellitenfunk, UPT</i> , B.G. Teubner, Stuttgart 2000
Weikmann 92	Franz Weikmann, ‘Smart Card-Chips – Technik und weitere Perspektiven’, <i>Der GMD-Spiegel</i> 1, 1992, Gesellschaft für Mathematik und Datenverarbeitung, Sankt Augustin 1992
Weikmann 98	Franz Weikmann, Klaus Vedder, ‘Smart Cards Requirements, Properties and Applications’, <i>Tagungsband Chipkarten</i> , Vieweg, Braunschweig 1998
Wiener 93	Michael J. Wiener, ‘Efficient DES Key Search’, Crypto 93, Santa Barbara 1993
Woodhouse 01	David Woodhouse, ‘JFFS: The Journalling Flash File System’, Red Hat, Inc., Internet 2001
Yellin 96	Frank Yellin, ‘Low Level Security in Java’, Internet 1996
Zieschang 98	Thilo Zieschang, ‘Differentielle Fehleranalyse und Sicherheit von Chipkarten’, Internet 1998

## 25.4 DIRECTORY OF STANDARDS AND SPECIFICATIONS

This section contains an extensively annotated directory of international standards, industry standards and specifications relevant to cards with and without chips. The main focus of this directory is international standards, with less attention given to local and country-specific standards. It lists standards generated by standardization organizations such ANSI, CEN, ETSI and ISO, as well as quasi-standards that are relevant to smart cards, such as the EMV specification and Internet RFCs.

In addition to the annotated directory, Table 25.2 on the next page provides a summary of potentially helpful sources of standards and specifications related to specific topics. Industrial standards in particular are generally available free of charge via the Internet. Unfortunately, this is not true of many standards issued by standardization organizations.

The standards and specifications that are especially important for smart cards are marked by a diamond symbol (♦) in the list that follows. The standards and specifications are listed in order of the name of the issuing organization and their numerical designation, ignoring prefixes (such as ‘pr’) and status indications (such as ‘DIS’). The date listed is the initial publication date of the currently applicable version.

**Table 25.2** Summary of the most important Web servers for downloading standards and information related to smart cards

Entity	Web server	Remarks
ANSI	[ANSI]	—
CEN	[CEN]	—
DIN	[DIN]	—
EMV	[EMVCO]	The specifications can be downloaded from the Web server free of charge
ETSI	[ETSI]	All ETSI standards (including those for GSM and UMTS) can be downloaded from the Web server free of charge
FIPS	[NIST]	All FIPS standards can be downloaded from the Web server free of charge
Global Platform	[Global Platform]	The specifications can be downloaded from the Web server free of charge
IEEE	[IEEE]	—
ISO/IEC	[ISO]	—
ITU	[ITU]	—
Java Card Forum	[JCF]	The specifications can be downloaded from the Web server free of charge
RFC	[RFC]	The specifications can be downloaded from the Web server free of charge
RSA Inc.	[RSA]	The specifications can be downloaded from the Web server free of charge
SEIS	[SEIS]	The specifications can be downloaded from the Web server free of charge

Extensions to ISO and ISO/IEC standards are usually contained in one or more amendments (Amd.). Each time a standard is revised, which normally takes place every five years, any outstanding amendments are incorporated into the new version of the standard as appropriate.

New versions of CEN standards are identified in a similar manner. In the case of FIPS standards, the number of the revised edition forms part of the name of the standard (e.g., FIPS 140-2). The telecommunication standards generated by ETSI use a three-digit number to distinguish different versions. In the case of industry standards, the revision level is indicated by a year number or a version number, depending on the publisher.

3GPP TS 21.111 Version 8.1.0: 2007 USIM and IC card requirements

*A short standard that specifies the basic functional requirements of a security module (USIM) for a UMTS mobile telecommunication network. This standard is the UMTS equivalent of the GSM 02.17 standard.*

3GPP TS 23.038 Version 8.0.0: 2008 Alphabets and language-specific information

*Specifies the character coding used for SMS and USSD and the character sets used for UMTS.*

3GPP TS 23.040 Version 8.1.0: 2008 Technical realization of the Short Message Service (SMS)

3GPP TS 31.102 Version 8.1.0: 2008 Characteristics of the USIM Application

*◆ Specifies the logical characteristics of the USIM smart card application in terms of a description of the interface between the USIM and the*

---

*UMTS mobile telephone. Includes detailed descriptions of all files and their data objects, definitions of several somewhat less UMTS-specific commands, and a list of command sequences for typical processes. Together with TS 31.101, it is the UMTS equivalent of the GSM 11.11 standard.*

3GPP TS 31.111 Version 8.0.0: 2008 USIM Application Toolkit (USAT)  
*Defines and extensively describes the USIM Application Toolkit (USAT) for USIMs. The USAT describes an interface between the mobile equipment and the USIM that allows supplementary applications in the USIM to assume partial control of the telephone. It introduces proactive commands for the USIM and defines many new commands related to control of the telephone for functions such as display output, keypad polling, and sending text messages. The GSM equivalent of this standard is GSM 11.14.*

3GPP TS 31.120 Version 7.0.0: 2007 UICC-terminal interface; Physical, electrical and logical test specification

3GPP TS 31.121 Version 7.2.0: 2007 UICC-terminal interface; Universal Subscriber Identity Module (USIM) application test specification

3GPP TS 31.122 Version 3.1.0: 2007 Universal Subscriber Identity Module (USIM) conformance test specification  
*Specifies the test environment, test equipment, test hierarchy, and individual test cases for testing USIMs. The tests described address electrical and information technology aspects exclusively. Detailed specifications are provided for tests covering a wide range of topics, such as electrical power, data transmission, file management, commands, and typical processes in the UMTS application. This standard is a very good example of how USIM tests can be described, constructed and executed. It is the USIM equivalent of the GSM 11.17 standard for testing SIMs.*

3GPP TS 33.102 Version 7.1.0: 2006 3G Security; Security Architecture

*A key standard for the entire security architecture of a UMTS mobile telecommunication network with regard to network access, authentication, confidentiality, and data integrity. Includes complete descriptions, independent of any specific cryptographic algorithm, of network security functions, authentication protocols and encryption methods, as well as the generation of authentication vectors and the key derivation processes that are used.*

3GPP TS 42.019 Version 5.1.0: 2005 Subscriber Identity Module Application Programming Interface (SIM API); Stage 1

3GPP TS 51.011 Version 4.15.0: 2005 Specification of the Subscriber Identity Module – Mobile Equipment (SIM–ME) interface (Release 4), 2003

◆ *Specifies the physical and logical properties of the SIM by means of a description of the interface between the SIM and the GSM mobile telephone. Includes definitions of the ID-1 and plug-in card formats and specifies the general mechanical parameters of the card and the contacts. Specifies all general electrical parameters, as well as the structure and*

*content of the ATR and PTS. Also defines the possible data structures, security mechanisms, commands, and return codes. Lists all data elements and files necessary for a SIM, along with typical command sequences. This standard is the GSM equivalent of the TS 31.101 and TS 31.102 UMTS standards.*

3GPP TS 51.013 Version 5.6.1: 2005 Test specification for Subscriber Identity Module (SIM) Application Programming Interface (API) for Java Card

3GPP TS 51.014 Version 4.5.0: 2005 Specification of the SIM Application Toolkit for the Subscriber Identity Module – Mobile Equipment (SIM–ME) interface (Release 4), 2003

◆ *Defines and extensively describes the SIM Application Toolkit (SAT) for SIMs. The SAT describes an interface between the mobile equipment and the SIM for partial control of the mobile telephone by SIM-resident supplementary applications. This standard introduces proactive commands for the SIM and defines many new commands related to controlling the mobile telephone, such as display output, keypad polling, and sending text messages.*

3GPP TS 51.017 Version 4.2.0: 2005 Subscriber Identity Module (SIM) test specification

ANSI X9.84: 2003 Biometric Information Management and Security

*This very extensive standard specifies the basic architectural principles, use, management, and security requirements for biometric data for a wide range of biometric identification methods.*

CC PP 01 Common Criteria for Information Technology Security – Smart Card Security User Group Protection Profile, Internet, 2001

CEPS, Version 2.1.3: 2001 Joint Specification for Common Electronic Purse Cards

*CEPS is an important standard for electronic purses and forms the basis of most current and future European purse systems. It is aligned to EN 1546.*

CWA 14167 Security Requirements for Trustworthy Systems Managing Certificates for Electronic Signatures

-1: 2003 Part 1: System Security Requirements

-2: 2004 Part 2: Cryptographic module for CSP signing operations with backup – Protection profile (CMCSOB-PP)

-3: 2004 Part 3: Cryptographic module for CSP key generation services – Protection profile (CMCKG-PP)

-4: 2004 Part 4: Cryptographic module for CSP signing operations – Protection profile (CMCSO PP)

CWA 14169: 2004 Secure Signature-creation devices ‘EAL 4+’

CWA 14170: 2004 Security requirements for signature creation applications

CWA 14171: 2004 General guidelines for electronic signature verification

CWA 14365 Guide on the Use of Electronic Signatures

-1: 2004 Part 1: Legal and Technical Aspects

-2: 2004 Part 2: Protection Profile for Software Signature Creation Devices

---

CWA 14890	◆ Application Interface for smart cards used as Secure Signature Creation Devices
-1: 2004	Part 1: Basic requirements
-2	Part 2: Additional Services
EMV Version 4.1 : 2007	Integrated Circuit Card Specification for Payment Systems ◆ <i>This is the most important family of standards for smart cards for payment transactions. It is published jointly by EMVCo [EMV]. The family consists of four parts, called books, that specify the smart cards, the associated debit and credit transactions, and the associated terminals.</i>
EMV Book 1	EMV Integrated Circuit Card Specification for Payment Systems, Book 1: Application Independent ICC to Terminal Interface Requirement, Version 4.1, 2007 ◆ <i>This part specifies the mechanical and electrical properties of the smart cards and terminals, including definitions of the activation and deactivation sequences, data transmission at the electrical level, the ATR, and the associated parameters. In addition, it specifies the T = 0 and T = 1 transmission protocols, the APDU structure, logical channels, and several fundamental card commands and application selection mechanisms.</i>
EMV Book 2	EMV Integrated Circuit Card Specification for Payment Systems, Book 2: Security and Key Management, Version 4.1, 2007 ◆ <i>This part describes static and dynamic data authentication, PIN encryption, and secure messaging. It also contains general conditions for managing the public keys of a payment system and the requirements for terminal security, including associated key management.</i>
EMV Book 3	EMV Integrated Circuit Card Specification for Payment Systems, Book 3: Application Specification, Version 4.1, 2007 ◆ <i>This part of the EMV specification defines a number of commands needed for smart cards and smart card applications for debit and credit cards, and it specifies the transaction processes. The appendix includes descriptions of all of the data objects and their coding, specifications for TLV coding of data, and general approaches to integrating EMV smart cards into payment systems.</i>
EMV Book 4	EMV Integrated Circuit Card Specification for Payment Systems, Book 4: Cardholder, Attendant and Acquirer Interface Requirements, Version 4.1, 2007 ◆ <i>Book 4 lists the mandatory and optional requirements for terminals that support EMV-compliant smart cards. This includes conceivable configurations, functional and security requirements for terminals, possible and permitted user messages, including the character set used, and the interface to the acquirer. This standard also defines the basic features of the architecture of the terminal software and a model of a terminal-resident interpreter for executable program code. The appendix contains a list of data objects relevant to the terminal and recommendations for</i>

---

	<i>the technical design of the terminal, as well as examples of terminals for point-of-sale, cash dispenser, and goods dispenser applications.</i>
EMV CPA	EMV Integrated Circuit Card Specifications for Payment Systems, Common Payment Application Specification, Version 1.0, December 2005
EN 1332	Identification card systems – Man–Machine Interface
- 1: 2007	Part 1: Design principles and symbols for the user interface
- 2: 1998	Part 2: Definition of a Tactile Identifier for ID-1 cards <i>Specifies a perceptible recess in ID-1 cards for detecting the card orientation.</i>
- 3: 1999	Part 3: Key pads
- 4: 2007	Part 4: Coding of user requirements for people with special needs
- 5: 2006	Part 5: Raised tactile symbols for differentiation of application on ID-1 cards
EN 1545	Identification card systems – Surface transport applications
-1: 2005	Part 1: Elementary data types, general code lists and general data elements
-2: 2005	Identification card systems – Surface transport applications – Part 2: Transport and travel payment related data elements and code lists
EN 1546	Identification card systems – Inter-sector electronic purse ◆ <i>The most important worldwide standard for electronic purses, which forms the basis for most purse systems. This family of standards has been kept relatively general and thus includes many options, but it provides a very good and complete description of an electronic purse.</i>
- 1: 1999	Part 1: Definition, concepts and structures <i>Defines the terms used in the entire family of standards and describes the basic concepts and structures of inter-sector electronic purse systems.</i>
- 2: 1999	Part 2: Security architecture <i>Describes the notation used for security mechanisms, the security architecture, and the associated procedures and mechanisms for inter-sector electronic purse systems.</i>
- 3: 1999	Part 3: Data elements and interchanges <i>Describes the data elements, files, commands and return codes used by all components of an inter-sector electronic purse system.</i>
- 4: 1999	Part 4: Data objects <i>Describes the TLV mechanism for reading arbitrary data objects from files and provides a detailed presentation of the components and states of a state machine for an inter-sector electronic purse system. Also includes a list of tags for all data objects used.</i>
ETSI TS 101 086 Version 8.2.0: 2005	Subscriber Identity Module (SIM) conformance test specification <i>Specifies the test environment, test equipment, test hierarchy, and individual test cases for testing SIMs. The tests described address electrical and information technology aspects exclusively. Tests addressing these</i>

---

*aspects are specified in detail, including electrical power, data transmission, file management, commands, and typical processes used in the GSM application. This specification is a very good and extensive example of how GSM tests can be described, constructed and executed.*

ETSI TS 101 181 Version 8.9.0: 2005 Security mechanisms for SIM application toolkit; Stage 2

◆ *Contains specifications for all security mechanisms needed for a link between the background system and the SIM that is secure against eavesdropping and manipulation. It also describes the basic mechanism of remote file management using the SIM.*

ETSI TS 101 476 Version 8.5.0: 2002 Subscriber Identity Module Application Programming Interface (SIM API); SIM API for Java Card; Stage 2

◆ *Specifies a Java Card variant for use as a SIM with the SIM Application Toolkit, based on the Java Card 2.1 specifications. This standard is the key document for using Java Card in GSM systems. The basis for this is provided by GSM 02.19.*

ETSI TS 101 955 Version 8.3.0: 2005 Test specification for SIM API for Java card

*Specifies the test environment, test applications, test procedures, test coverage, and individual test cases for the SIM API for Java Card in accordance with GSM 03.19. All of the described tests address the information technology aspects of Java Card SIMs for GSM. This specification is a very good and extensive example of how Java Card tests can be described, constructed and executed.*

ETSI TS 102 221 Version 7.10.0: 2008 Smart cards; UICC-Terminal interface; Physical and logical characteristics

◆ *Specifies the physical and logical characteristics of a USIM by means of a description of the interface between the USIM and the UMTS mobile telephone. Includes definitions of the ID-1 and plug-in card formats and specifies the general mechanical parameters of the card and the contacts, as well as all general electrical parameters. It also specifies the structure and data content of the ATR and PPS and defines the transmission protocols, file structures, security mechanisms, commands, and return codes. In addition, it lists all files, associated data objects, and command sequences that are independent of any particular telecommunication application. This standard forms the basis for smart card operating systems for the USIM. It is complemented by TS 31.103, which addresses all application-specific components of a USIM.*

ETSI TS 102 222 Version 7.1.0: 2007 Integrated Circuit Cards (ICC); Administrative commands for telecommunications applications (Release 6), 2005

◆ *Specifies the administrative commands for file management and associated security conditions for telecommunication smart cards.*

ETSI TS 102 223 Version 7.10.0: 2008 Smart cards; Card Application Toolkit (CAT)

◆ *Defines and thoroughly describes a generic application toolkit for telecommunication smart cards. The CAT describes an interface between the mobile equipment and the smart card that allows supplementary*

- applications in the smart card to assume partial control of the telephone. This standard defines commands related to controlling the telephone for functions such as display output, keypad polling, and sending text messages. It forms the basis for other standards such as GSM 11.14 and TS 31.111.*
- ETSI TS 102 230 Version 5.7.0: 2007 Smart cards; UICC–Terminal Interface; Physical, Electrical and Logical Test Specification  
*Specifies physical and electrical tests for UICCs and describes basic tests for the communication link to the UICC and tests for the T = 0 and T = 1 transmission protocols.*
- ETSI TS 102 268 Version 6.0.0: 2007 Smart Cards; Test specification for the UICC Application Programming Interface (AP)I for Java Card
- ETSI TS 102 588 Version 7.1.0: 2007 Smart Cards; Application invocation Application Programming Interface (API) by a UICC web server for Java Card platform
- ETSI TS 102 600 Version 7.1.0: 2007 Smart Cards; UICC–Terminal interface; Characteristics of the USB interface
- ETSI TS 102 613 Version 7.1.0: 2008 Smart Cards; UICC– Contactless Front-end (CLF) Interface; Part 1: Physical and data link layer characteristics
- ETSI TS 102 622 Version 7.0.0: 2008 Smart Cards; UICC– Contactless Front-end (CLF) Interface; Host Controller Interface (HCI)
- FIPS 46-3: 1999 Data Encryption Standard (DES)  
*Describes the DES and triple-DES algorithms.*
- FIPS 74: 1981 Guidelines for Implementing and Using the NBS Encryption Standard
- FIPS 81: 1980 DES Modes of Operation
- FIPS 140-2: 2001 Security Requirements for Cryptographic Modules  
◆ *A basic standard used worldwide to define the security requirements for security modules, which can also include smart cards. It defines four security levels for security modules and provides detailed descriptions of seven security-related requirement areas. The contents of this standard are very practically oriented and also address the details of the technical implementation, such as criteria for the quality of random number generators.*
- FIPS 180-3: 2007 Secure Hash Standard (SHA)  
◆ *Describes the SHA-1 hash function.*
- FIPS 186-2: 2000 Digital Signature Standard (DSS)  
*Describes the DSS algorithm.*
- FIPS 197: 2001 Advanced Encryption Standard (AES)  
◆ *Describes the AES algorithm.*
- Global Platform Card Specification 2.2: 2006  
◆ *The most important specification with regard to managing applications in multiapplication smart cards. This very comprehensive specification contains a detailed presentation of the software and security architectures of multiapplication smart cards and an extensive description of the*

*commands needed for this purpose. The appendix includes the specification of an API for application management with Java Card, which has become the de facto standard for this type of smart card.*

ICAO 9303-1 Vol. 1: 2006 ICAO 9303 Part 1 – Machine Readable Passport – Volume 1: Passports with Machine Readable Data Stored in Optical Character Recognition Format

ICAO 9303-1 Vol. 2: 2006 ICAO 9303 Part 1 – Machine Readable Passport – Volume 2: Specifications for Electronically Enabled Passports with Biometric Identification Capabilities

ICAO 9303-2: 2005 ICAO 9303 Part 2 – Machine Readable Visas

ICAO 9303-3: 2002 ICAO 9303 Part 3 – Size 1 and Size 2 Machine Readable Official Travel Documents

IEEE 1363: 2000 Standard Specifications for Public-Key Cryptography  
◆ *A very extensive and comprehensive standard that addresses almost all aspects of asymmetric cryptographic algorithms, including generating keys, using digital signatures, key exchange, and encryption.*

ISO 639 Codes for the representation of names of languages

- 1: 2002 Part 1 Alpha-2 code
- 2: 1998 Part 2 Alpha-3 code
- 3: 2007 Part 3 Alpha-3 code for comprehensive coverage of languages
- 4: DIS 2006 Part 4 Implementation guidelines and general principles for language coding
- 5: FDIS 2008 Part 5 Alpha-3 code for language families and groups
- 6: 2007 Part 6 Alpha-4 representation for comprehensive coverage of language variants

ISO 3166 Codes for the representation of names of countries and their subdivisions

- 1: 2006 Part 1 Country codes
- 2: 2007 Part 2 Country subdivision code
- 3: 1999 Part 3 Code for formerly used names of countries

ISO 4217 DIS: 2007 Codes for the representation of currencies and funds

ISO 4909: 2006 Bank cards – Magnetic stripe data contents for track 3

ISO 7810: 2003 Identification cards – Physical characteristics  
*Describes the most important physical properties of cards without chips and defines the ID-1, ID-2, and ID-3 card formats.*

ISO 7811 Identification cards – Recording technique  
*This family of standards is an important reference for the mechanical aspects of cards. It specifies the mechanical implementation of the essential card components.*

- 1: 2002 Part 1: Embossing  
*An exact definition of the ten numeric characters and the basic method used to emboss cards.*
- 2: 2001 Part 2: Magnetic stripe – low coercivity

---

	<i>Defines the size and location of the magnetic stripe on the card. Also specifies the physical properties of the magnetic material and the coding of the characters on the magnetic stripe.</i>
- 3: 2003	Part 3: Location of embossed characters on ID-1 cards <i>Defines the possible locations for embossing on ID-1 cards.</i>
- 4: 1996	Part 4: Location of read-only magnetic tracks – Tracks 1 and 2 <i>Defines the locations of the read-only tracks (tracks 1 and 2) on an ID-1 card.</i>
- 5: 1996	Part 5: Location of read-write magnetic track – Track 3 <i>Defines the location of the read/write track (track 3) on an ID-1 card.</i>
- 6 FIDS: 2008	Part 6: Magnetic stripe – High coercivity
- 7 WD: 2001	Part 7: Magnetic stripe – High coercivity, high density
- 8: 2008	Part 8: Magnetic stripe – Coercivity of 51.7 kA/m (650 Oe)
- 9 FDIS: 2008	Part 9: Tactile identifier mark
ISO 7812	Identification cards – Identification of issuers
- 1: 2006	Part 1: Numbering system <i>Specifies a numbering scheme for issuers of ID cards.</i>
- 2: 2007	Part 2: Application and registration procedures <i>Defines the registration authority and a form for registering applications. Also contains an algorithm for generating a Luhn checksum (modulo-10 checksum).</i>
ISO 8583	Financial transaction card originated messages – Interchange message specifications <i>A standard for data transfer between a terminal and its host system. In Germany, communication between debit card terminals and the background system is based on this standard.</i>
- 1: 2003	Part 1: Messages, data elements and code values
- 2: 1998	Part 2: Application and registration procedures for Institution Identification Codes (IIC)
- 3: 2003	Part 3: Maintenance procedures for messages, data elements and code values
ISO 9564	Banking – Personal Identification Number management and security
- 1: 2002	Part 1: PIN protection principles and techniques <i>Fundamental aspects of PIN selection, PIN management and PIN protection for general banking applications. The appendices define general requirements for PIN entry devices, among other things, as well as recommendations for the layout of the corresponding keypads. They also include advice regarding erasing sensitive data on various media, such as magnetic tape, paper, and semiconductor memories.</i>
- 2: 2005	Part 2: Approved algorithm(s) for PIN encipherment <i>A very short standard that defines DES as an algorithm for PIN encryption.</i>

- 3: 2003	Part 3: PIN protection requirements for offline PIN handling in ATM and POS systems
ISO 9992	Financial Transaction Cards – Messages between the Integrated Circuit Card and the Card Accepting Device
- 1: 1990	Part 1: Concepts and structures
- 2: 1998	Part 2: Functions, messages (commands and responses), data elements and structures <i>Defines commands, procedures, and data elements for smart cards used in financial transaction systems. Contains the definitions of the tags used in financial transaction systems and many cross-references to other standards in the ISO/IEC 7816 family.</i>
- 4 DIS: 1993	Part 4: Common data for interchange
- 5 CD: 1991	Part 5: Organization of data elements
ISO 11568	Banking – Key management
- 1: 2005	Part 1: Introduction to Key Management
- 2: 2005	Part 2: Key management techniques for symmetric ciphers
- 3: 1994	Part 3: Key life cycle for symmetric ciphers
- 4: 2007	Part 4: Key management techniques for public key cryptosystems
- 5: 1998	Part 5: Key life for public key cryptosystems
- 6: 1998	Part 6: Key management schemes
ISO 15782	Banking – Certificate management for financial services
- 1: 2003	Part 1: Public Key Certificates
- 2: 2001	Part 2: Certificate extensions
ISO/IEC 646: 1991	Information technology – ISO 7-bit coded character set for information interchange
ISO/IEC 7501	Identification cards – Machine readable travel documents
- 1 DIS: 2007	Part 1: Machine readable passport
- 2: 1997	Part 2: Machine readable visa
- 3: 2005	Part 3: Machine readable official travel documents
ISO/IEC 7813: 2006	Identification cards – Financial transaction cards
ISO/IEC 7816	Identification cards – Integrated circuit(s) cards with contacts ◆ <i>The most important family of ISO standards for microcontroller cards. The first three parts primarily focus on the card and the chip hardware. The remaining parts specify all mechanisms and properties of applications and operating systems for smart cards, as well as the associated information technology aspects.</i>
- 1: 1998	Part 1: Physical characteristics <i>Defines the physical characteristics of a card with a contact chip, as well as the tests to be used with such a card.</i>
- 2: 2007	Part 2: Dimensions and location of the contacts <i>Defines the sizes and locations of the contacts of a smart card, as well as</i>

- the possible arrangements of the chip, magnetic stripe and embossing. Also describes the method to be used to measure the locations of the contacts on the smart card.*
- 3: 2006      Part 3: Electrical interface and transmission protocols  
◆ *The most important ISO standard for the general electrical parameters of a microcontroller smart card. It specifies all basic electrical properties, such as the supply voltage, stopping the clock, and the reset characteristics (cold and warm reset). It also defines the parameters, structure, and possible sequences of the ATR and PPS. A large part of this standard deals with basic aspects of data transmission at the physical level (such as the divisor) and the definition of the two transmission protocols ( $T = 0$  and  $T = 1$ ), and it includes extensive examples of communication processes.*
- 4: 2005      Part 4: Organization, security and commands for interchange  
◆ *The most important application-level ISO standard for smart cards. It defines the file organization, file structures, security architecture, TPDU, APDUs, secure messaging, return codes, and logical channels. The majority of this standard is taken up by an extensive description of smart card commands. Fundamental smart card mechanisms for general industrial applications are also described here.*
- 5: 2004      Part 5: Registration of application identifiers  
*Defines the numbering scheme for uniquely identifying national and international applications in smart cards. Also defines the exact data structure of the AID and describes the process for registering applications.*
- 6: 2004      Part 6: Inter-industry data elements for interchange  
*Defines the data objects (DOs) and associated TLV tags for general industrial applications, and describes the associated TLV structures and procedures for reading data objects from smart cards.*
- 7: 1999      Part 7: Commands for Structured Card Query Language (SCQL)  
*Defines supplementary smart card commands as an extension to ISO/IEC 7816-4. Defines the basic principles of a database system based on SQL, and specifies the commands for the associated SCQL accesses to smart cards.*
- 8: 2004      Part 8: Commands for security operations  
◆ *This part of the family of standards is fully dedicated to functions and commands related to security. As an extension to ISO/IEC 7816-4, it defines additional mechanisms for secure messaging, as well as numerous commands for cryptographic functions such as digital signatures, hash computation, MAC computation, and the encryption and decryption of data.*
- 9: 2004      Part 9: Commands for card management  
◆ *This standard is divided into three sections. The first section describes the initial portion of the life cycle of smart card applications at the file level in terms of states. The second section describes access control*

---

	<i>objects (ACOs) that can be used to govern file accesses. The extensive third section defines search commands for file contents and administrative commands for creating and deleting files, which are necessary for managing applications.</i>
- 10: 1999	Part 10: Electronic signals and answer to reset for synchronous cards <i>For memory cards, this is the counterpart to Part 3 of this family of standards. It specifies the essential electrical characteristics of memory cards and defines the parameters and structure of the ATR and possible ATR processes for synchronous cards.</i>
- 11: 2004	Part 11: Personal verification through biometric methods <i>Defines commands for biometric user identification and the associated data objects. In addition, the appendix illustrates the basic features of methods for recording biometric data in the card (enrollment) and describes a scenario for verifying this biometric data.</i>
- 12: 2005	Part 12: USB electrical interface and operating procedures
- 13: 2007	Part 13: Commands for application management in a multiapplication environment
- 15: 2004	Part 15: Cryptographic information application ◆ <i>This part of the standard, which is based on the PKCS #15 standard, defines all data elements necessary for interoperable smart cards for digital signature applications. It includes descriptions of all data objects, directories and files needed for signature cards, as well as ASN.1 descriptions of all of the certificates, keys, and other administrative data stored in the files.</i>
ISO/IEC 8824	Information technology – Abstract Syntax Notation One (ASN.1) <i>Defines the basic ASN.1 coding rules.</i>
ISO/IEC 8825	Information technology – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1) <i>Defines the ASN.1 data description language.</i>
ISO/IEC 8859	Information technology – 8-bit single-byte coded graphic character sets
ISO/IEC 9075	Information technology – Database languages – SQL <i>Defines the Structured Query Language (SQL) database query language, which is a superset of the smart card database query language SCQL.</i>
ISO/IEC 9979: 1999	Information technology – Security techniques – Procedures for the registration of cryptographic algorithms
ISO/IEC 9796	Information technology – Security techniques – Digital signature scheme giving message recovery <i>Defines methods for generating and verifying digital signatures with message recovery. The appendix contains several numerical examples of key generation, signature generation, and signature verification.</i>
- 1: 1999	Part 1: Mechanisms using redundancy
- 2: 2002	Part 2: Integer factorization based mechanisms
- 3: 2006	Part 3: Discrete logarithm based mechanisms

ISO/IEC 9797	Information technology – Security techniques – Message Authentication Codes (MACs)
- 1 CD: 2007	Part 1: Mechanisms using a block cipher
- 2: 2002	Part 2: Mechanisms using a dedicated hash function
ISO/IEC 9798	Information technology – Security techniques – Entity authentication ◆ <i>This family of standards contains detailed descriptions of various cryptographic methods for authenticating one, two or three parties. It is the most important reference on the subject of authentication.</i>
- 1: 1997	Part 1: General <i>Defines the terms and notation used in the other parts of this family of standards.</i>
- 2: 2004	Part 2: Mechanisms using symmetric encipherment algorithms <i>Specifies authentication methods based on symmetric cryptographic algorithms.</i>
- 3: 1998	Part 3: Mechanisms using digital signature techniques <i>Specifies authentication methods based on asymmetric cryptographic algorithms.</i>
- 4: 1999	Part 4: Mechanisms using a cryptographic check function <i>Specifies authentication methods based on cryptographic check functions.</i>
- 5: 2004	Part 5: Mechanisms using zero knowledge techniques <i>Specifies authentication methods based on zero-knowledge techniques.</i>
ISO/IEC 10116: 2006	Information technology – Security techniques – Modes of operation for an n-bit block cipher algorithm <i>Describes the four standard operating modes (ECB, CBC, CFB, OFD) of a block-oriented encryption algorithm. An appendix contains detailed comments on the use of each of the four modes, while another appendix contains corresponding numerical examples.</i>
ISO/IEC 10118	Information technology – Security techniques – Hash functions <i>General principles of hash functions, as well as associated padding methods.</i>
- 1: 2000	Part 1: General
- 2: 2000	Part 2: Hash functions using an n-bit block cipher algorithm <i>Defines hash functions based on a block encryption algorithm and describes algorithms with single-length and double-length keys. The appendix contains a numerical example for each type of key, based on the DES algorithm.</i>
- 3: 2004	Part 3: Dedicated hash functions
- 4: 1998	Part 4: Hash functions using modular arithmetic
ISO/IEC 10373	Identification cards – Test methods ◆ <i>Fundamental standard for card testing. Contains precise descriptions of test methods for card bodies and card bodies with implanted chips. The individual tests are described in detail, with many explanatory drawings.</i>

---

- 1: 2006	Part 1: General characteristics tests
- 2: 2006	Part 2: Cards with magnetic stripes
- 3: 2001	Part 3: Integrated circuit(s) cards with contacts and related interface devices <i>Specifies the test environment, test methods and test procedures for electrical tests of contact smart cards. Also specifies detailed procedures for checking contact locations, electrical power; ATR and PPS data transfer, and data transmission protocols.</i>
- 5: 2006	Part 5: Optical memory cards
- 6: 2001	Part 6: Proximity cards
- 7 FDIS: 2008	Part 7: Vicinity cards
ISO/IEC 10536	Identification cards – Contactless integrated circuit(s) cards – Close-coupled cards <i>This standard describes contactless smart cards used in applications that restrict them to direct contact with the terminal.</i>
- 1: 2000	Part 1: Physical characteristics <i>Defines the physical characteristics of contactless smart cards and associated test methods.</i>
- 2: 1995	Part 2: Dimension and location of coupling areas <i>Specifies the dimensions and locations of the coupling areas of contactless cards and their use with card terminals having a card slot or surface interface.</i>
- 3: 1996	Part 3: Electronic signals and reset procedures <i>Defines the electrical signals of the inductive and capacitive components used to couple the smart card to the terminal.</i>
ISO/IEC 10646	Information technology – Universal Multiple-Octet Coded Character Set (UCS)
ISO/IEC 11693: 2005	Identification cards – Optical memory cards – General characteristics
ISO/IEC 11694	Identification cards – Optical memory cards and devices – Linear recording method
- 1: 2005	Part 1: Physical characteristics
- 2: 2005	Part 2: Dimensions and location of the accessible optical area
- 3: 2001	Part 3: Optical properties and characteristics
- 4: 2001	Part 4: Logical data structures
- 5: 2006	Part 5: Data format for information interchange for applications using ISO/IEC 11694-4, Annex B
- 6: 2006	Part 6: Use of biometrics on an optical memory card
ISO/IEC 11770	Information technology – Security techniques – Key management
- 1: 1996	Part 1: Framework
- 2 FDIS: 2008	Part 2: Mechanisms using symmetric techniques
- 3: 1999	Part 3: Mechanisms using asymmetric techniques
- 4: 2006	Part 4: Mechanisms based on weak secrets

---

ISO/IEC 12207 FDIS: 2007	Information technology – Software life cycle processes
ISO 13491	Banking – Secure cryptographic devices
- 1: 2007	Part 1: Concepts, requirements and evaluation methods
- 2: 2005	Part 2: Security compliance checklists for devices used in magnetic stripe card systems
ISO/IEC 13888	Information technology – Security techniques – Non-repudiation
- 1: 2004	Part 1: General
- 2: 1998	Part 2: Mechanisms using symmetric techniques
- 3: 1997	Part 3: Mechanisms using asymmetric techniques
ISO/IEC 14443	Identification cards – Contactless integrated circuit(s) cards – Proximity cards  ◆ <i>This standard describes contactless smart cards that can be used at a distance of up to several tens of centimeters from a terminal.</i>
- 1: FDIS 2008	Part 1: Physical characteristics
- 2: 2001	Part 2: Radio frequency power and signal interface
- 3: 2001	Part 3: Initialization and anticollision
- 4: 2001	Part 4: Transmission protocol
ISO/IEC 14888	Information technology – Security techniques – Digital signature with appendix  <i>This standard specifies basic mechanisms and methods for digital signatures with appendix.</i>
- 1: FDIS 2007	Part 1: General
- 2: FDIS 2007	Part 2: Integer factorization based mechanisms
- 3: 2006	Part 3: Discrete logarithm based mechanisms
ISO/IEC 15292: 2001	Information technology – Security techniques – Protection Profile registration procedures
ISO/IEC 15408	Information technology – Security techniques – Evaluation criteria for IT security
- 1: 2007	Part 1: Introduction and general model
- 2: 2007	Part 2: Security functional requirements
- 3: 2007	Part 3: Security assurance requirements
ISO/IEC 15457	Identification cards – Thin flexible cards
-1: 2008	Part 1: Physical characteristics
-2: 2007	Part 2: Magnetic recording techniques
-3: 2008	Part 3: Test methods
ISO/IEC 15693	Identification cards – Contactless integrated circuit(s) cards – Vicinity cards  <i>This standard describes contactless smart cards that can be used at a distance of up to one meter from a terminal.</i>
- 1: 2000	Part 1: Physical characteristics

---

- 2: 2006	Part 2: Air interface and initialization
- 3: 2001	Part 3: Anticollision and transmission protocol
ISO/IEC 15946	Information technology – Security techniques – Cryptographic techniques based on elliptic curves
- 1 FDIS: 2007	Part 1: General
- 2 FDIS: 2001	Part 2: Digital signatures
- 3: 2002	Part 3: Key establishment
- 4 CD: 2000	Part 4: Digital signatures giving message recovery
ISO/IEC 18013	Personal Identification – ISO Compliant Driving Licence
- 1 WD: 2005	Part 1: Physical characteristics and data set
- 2 FDIS: 2007	Part 2: Machine-readable technologies
ISO/IEC 18045: 2005	Information technology – Security techniques – Methodology for IT security evaluation
ISO/IEC 18092: 2004	Information technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1)
ISO/IEC 19794	Information technology – Biometric data interchange formats
ISO/IEC 24727	Identification cards – Integrated circuit card programming interfaces
ITU X.509	ITU X.509: 2005 Information Technology – Open Systems Interconnection – The Directory: Authentication Framework ◆ <i>Specifies the structure and coding of certificates. It forms the most commonly used worldwide basis for certificate structures.</i>
Java Card 2.2.2: 2006	◆ <i>This industrial standard forms the basis for Java Card. It is generated by the Java Card Forum and published by the Sun Corporation. The three standards in this family are mutually complementary and address various aspects of Java Card implementation.</i>
JCAPI 06	Sun Microsystems: Java Card Platform Specification, Version 2.2.2 – Application Programming Interface, Santa Clara 2006 ◆ <i>Specifies the complete interface (API) available to an applet in a Java Card environment. It essentially consists of a comprehensive list of all classes and interfaces of the Java Card API.</i>
JCBio 02	Biometric Application Programming Interface (API) for Java Card, Version 1.1, NIST 2002
JCRES 06	Sun Microsystems: Java Card Platform Specification, Version 2.2.2 – Runtime Environment Specification, Santa Clara 2006 ◆ <i>Specifies the Java Card runtime environment, which essentially consists of the Java virtual machine and the Java Card API. It addresses the following topics in detail: the lifetime of the virtual machine, applet lifetimes, applet selection, transient objects, object sharing, transactions, atomic operations, and applet installation.</i>
JCVMS 06	Sun Microsystems: Java Card Platform Specification, Version 2.2.2 – Virtual Machine Specification, Santa Clara 2006

♦ *Specifies the Java Card virtual machine, including its detailed architecture, its instruction set, and the format of CAP files.*

MKT Specification, Version 1.0: 1999 Multifunctional Card Terminals

*The MKT specification published by Teletrust Deutschland is the quasi-standard in Germany for connecting terminals to PCs.*

Part 1	MKT Basiskonzept
Part 2	CT-ICC-Interface – MKT-Schnittstelle für kontaktorientierte Chipkarten mit synchroner und asynchroner Übertragung
Part 3	CT-API 1.1 – Anwendungsunabhängiges Card Terminal Application Programming Interface
Part 4	CT-BCS – Anwendungsunabhängiges Card Terminal Basic Command Set
Part 5	Chipkarten mit synchroner Übertragung – ATR und Datenbereiche
Part 6	Chipkarten mit synchroner Übertragung – Übertragungsprotokolle
Part 7	Chipkarten mit synchroner Übertragung – Anwendung von Interindustry Commands

OMA Wireless Application Protocol Identity Module Specification, Version 260: July 2001

*Specifies the physical and electrical properties of a WIM, which is the digital signature application for telecommunications smart cards. It lists all mechanisms, commands, data objects, and files needed for a WIM application.*

OMA Smartcard-Web-Server Version 1.0: 2007

PC/SC V 2.01.4: July 2007 Interoperability Specification for ICCs and Personal Computer Systems

♦ *This extensive, detailed specification forms the basis for linking smart cards and terminals to the resource management functions of 16-bit and 32-bit Microsoft operating systems.*

- 1	Part 1: Introduction and Architecture Overview
- 2	Part 2: Interface Requirements for Compatible IC Cards and Readers
- 3	Part 3: Requirements for PC-Connected Interface Devices
- 4	Part 4: IFD Design Considerations and Reference Design Information
- 5	Part 5: ICC Resource Manager Definition
- 6	Part 6: ICC Service Provider Interface Definition
- 7	Part 7: Application Domain and Developer Design Considerations
- 8	Part 8: Recommendations for ICC Security and Privacy Devices

PKCS

*The Public Key Cryptography Standards (PKCS) are industry standards published by RSA Inc. that focus on the use of asymmetric cryptographic algorithms.*

PKCS #1 V 2.1: 2001 RSA Cryptography Standard

*This PKCS standard describes mechanisms for encryption and decryption using the RSA algorithm.*

PKCS #3 V 1.4: 1993 Diffie–Hellman Key-Agreement Standard

*This PKCS standard describes a procedure for key exchange between two parties using the Diffie–Hellman method.*

PKCS #5 V 2.1: 2006 Password-Based Cryptography Standard

*This PKCS standard contains recommendations for the implementation of encryption, key derivation, and MAC generation based on keys generated from passwords.*

PKCS #11 V 2.2: 2004 Cryptographic Token Interface Standard

◆ *The de facto international standard for an API for invoking cryptographic functions. This API is called ‘Cryptoki’ (cryptographic token interface) and includes functions such as RC2, RC4, RC5, MD5, SHA-1, DES, triple-DES, IDEA, RSA, DSA, MAC computation, and key generation for a wide variety of cryptographic algorithms.*

PKCS #13 V 1.0: 1998 Elliptic Curve Cryptography Standard

PKCS #15 PKCS #15 V 1.1: 2000 Cryptographic Token Information Format Standard

◆ *The de facto worldwide standard for the data objects needed for an interoperable smart card for digital signatures. It includes descriptions of all directories and files needed for a signature card and ASN.1 descriptions of all certificates, keys, and other administrative data stored in the files.*

RFC 768: 1980 User Datagram Protocol

RFC 791: 1981 Internet Protocol

RFC 793: 1981 Transmission Control Protocol

RFC 1321: 1992 The MD5 Message-Digest Algorithm

RFC 1750: 1994 Randomness Recommendations for Security

*Describes the operating principles of various types of random number generators, and based on these principles, recommends methods for designing high-quality pseudorandom number generators for PCs.*

RFC 1945: 1996 Hypertext Transfer Protocol – HTTP/1.0

RFC 2104: 1997 HMAC: Keyed-Hashing for Message Authentication

RFC 2246: 1999 The TLS Protocol

RFC 2289: 1998 A One-Time Password System

RFC 2460: 1998 Internet Protocol, Version 6 (IPv6)

RFC 2616: 1999 Hypertext Transfer Protocol – HTTP/1.1

RFC 2617: 1999 HTTP Authentication: Basic and Digest Access Authentication

RFC 2818: 2000 HTTP Over TLS

RFC 3161: 2001 Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)

RFC 3748: 2004 Extensible Authentication Protocol (EAP)

SP 800-57 Recommendation for Key Management

- 1: 2007 Part 1: General

- 2 Part 2: Best Practices for Key Management Organization

---

TR-3110: 2007	BSI Technical Guideline TR-03110; Advanced Security Mechanism for Machine Readable Travel Documents – Extended Access Control (EAC), Version 1.1
TS 15480	Identification card systems – European Citizen Card
Part 1: 2007	Physical, electrical and transport protocol characteristics
Part 2: 2007	Logical data structures and card services
Part 3: 2007	Interoperability using an application interface
Part 4: 2007	Card issuance, operation and use
Universal Serial Bus Revision 2.0 Specification: 2000	<p>◆ <i>This very comprehensive specification forms the basis for the USB interface.</i></p>

## 25.5 WEB ADDRESSES

The list of World Wide Web addresses in Table 25.3 is a cross-section of the various companies and institutions that are active in the field of smart cards. Of course, it is not intended to be complete, but it can form a useful starting point for further research. A current version of this list in HTML format is available on the website of Wolfgang Rankl [[www.wrankl.de](http://www.wrankl.de)].

Generally speaking, a company's website includes the company's postal address and the telephone number of a contact person, so this information is not included in the following list. If you have a specific need for particular information, we generally advise you to use appropriate search terms (keywords) and a suitable search engine to comb through the World Wide Web.

**Table 25.3** Web addresses of companies and organizations active in the smart card domain

---

Reference	Company or organization	URL
[3GPP]	3GPP	<a href="http://www.3gpp.org">www.3gpp.org</a>
[3GPP2]	3GPP2	<a href="http://www.3gpp2.org">www.3gpp2.org</a>
[AC]	Austria Card	<a href="http://www.austriacard.at">www.austriacard.at</a>
[AFNOR]	Association française de Normalisation	<a href="http://www.afnor.org">www.afnor.org</a>
[AmEx]	American Express	<a href="http://www.americanexpress.com">www.americanexpress.com</a>
[ANSI]	American National Standards Institute	<a href="http://www.ansi.org">www.ansi.org</a>
[Atmel]	Atmel	<a href="http://www.atmel.com">www.atmel.com</a>
[BasicCard]	BasicCard	<a href="http://www.basiccard.com">www.basiccard.com</a>
[Bluetooth]	Bluetooth	<a href="http://www.bluetooth.com">www.bluetooth.com</a>
[BMG]	Bundesministerium für Gesundheit	<a href="http://www.bmg.bund.de">www.bmg.bund.de</a>
[BSI]	Bundesamt für Sicherheit in der Informationstechnik	<a href="http://www.bsi.bund.de">www.bsi.bund.de</a>
[Hanser]	Carl Hanser Verlag	<a href="http://www.hanser.de">www.hanser.de</a>
[CC]	Common Criteria	<a href="http://www.commoncriteria.org">www.commoncriteria.org</a>
[CCC]	Chaos Computer Club	<a href="http://www.ccc.de">www.ccc.de</a>
[CDG]	CDMA Development Group (CDG)	<a href="http://www.cdg.org">www.cdg.org</a>
[CEN]	Comité Européen de Normalisation	<a href="http://www.cenorm.be">www.cenorm.be</a>
[CEPS]	Common Electronic Purse Specifications	<a href="http://www.cepsco.com">www.cepsco.com</a>
[DECT]	DECT Forum	<a href="http://www.dect.org">www.dect.org</a>

**Table 25.3** (*Continuation*)

Reference	Company or organization	URL
[Diehard]	Diehard	<a href="http://www.stat.fsu.edu/pub/diehard">www.stat.fsu.edu/pub/diehard</a>
[DIN]	Deutsches Institut für Normung	<a href="http://www.din.de">www.din.de</a>
[DPA]	Deutsches Patentamt	<a href="http://www.deutsches-patentamt.de">www.deutsches-patentamt.de</a>
[Diners]	Diners Club	<a href="http://www.diners-club.com">www.diners-club.com</a>
[Dolomiti]	Dolomiti Superski	<a href="http://www.dolomitisuperski.de">www.dolomitisuperski.de</a>
[Drexler]	Drexler Technology	<a href="http://www.lasercard.com">www.lasercard.com</a>
[ECBS]	European Committee for Banking Standards	<a href="http://www.ecbs.org">www.ecbs.org</a>
[EMV]	EMVCO	<a href="http://www.emvco.com">www.emvco.com</a>
[ETSI]	European Telecommunications Standards Institute	<a href="http://www.etsi.org">www.etsi.org</a>
[Eurosmart]	Eurosmart	<a href="http://www.eurosmart.com">www.eurosmart.com</a>
[EZ-link]	EZ-link Card	<a href="http://www.ezlink.com.sg">www.ezlink.com.sg</a>
[FeliCa]	FeliCa	<a href="http://www.sony.co.jp/Products/felica">www.sony.co.jp/Products/felica</a>
[FINEID]	FINEID	<a href="http://www.fineid.fi">www.fineid.fi</a>
[GD]	Giesecke & Devrient	<a href="http://www.gi-de.com">www.gi-de.com</a>
[GeldKarte]	GeldKarte	<a href="http://www.geldkarte.de">www.geldkarte.de</a>
[Gemalto]	Gemalto	<a href="http://www.gemalto.com">www.gemalto.com</a>
[Gematik]	Gematik	<a href="http://www.gematik.de">www.gematik.de</a>
[GP]	Global Platform	<a href="http://www.globalplatform.org">www.globalplatform.org</a>
[GSM]	GSM Association	<a href="http://www.gsmworld.com">www.gsmworld.com</a>
[IATA]	International Air Transport Association	<a href="http://www.iata.org">www.iata.org</a>
[ICAO]	International Civil Aviation Organization	<a href="http://www.icao.int">www.icao.int</a>
[ICMA]	International Card Manufacturers Association	<a href="http://www.icma.com">www.icma.com</a>
[IEC]	International Electrotechnical Commission	<a href="http://www.iec.ch">www.iec.ch</a>
[IEEE]	Institute of Electrical and Electronics Engineers	<a href="http://www.ieee.org">www.ieee.org</a>
[IMT-2000]	IMT-2000	<a href="http://www.imt-2000-online.com">www.imt-2000-online.com</a>
[Infineon]	Infineon	<a href="http://www.infineon.de">www.infineon.de</a>
[Integri]	Integri	<a href="http://www.integri.be">www.integri.be</a>
[Iridium]	Iridium	<a href="http://www.iridium.com">www.iridium.com</a>
[ISC]	Internet Systems Consortium	<a href="http://www.isc.org">www.isc.org</a>
[ISO]	International Organization for Standardization	<a href="http://www.iso.org">www.iso.org</a>
[ITU]	International Telecommunication Union	<a href="http://www.itu.int">www.itu.int</a>
[JCB]	Japan Credit Bureau	<a href="http://www.jcb-global.com">www.jcb-global.com</a>
[JCF]	Java Card Forum	<a href="http://www.javacardforum.org">www.javacardforum.org</a>
[JTC1]	Joint Technical Committee One	<a href="http://www.jtc1.org">www.jtc1.org</a>
[Maosco]	Maosco Ltd.	<a href="http://www.multos.com">www.multos.com</a>
[MasterCard]	MasterCard International	<a href="http://www.mastercard.com">www.mastercard.com</a>
[Mifare]	Mifare	<a href="http://www.mifare.net">www.mifare.net</a>
[MMCA]	MultiMediaCard Association	<a href="http://www.mmca.org">www.mmca.org</a>
[Mondex]	Mondex International Ltd	<a href="http://www.mondex.com">www.mondex.com</a>
[MUSCLE]	Movement for the Use of Smart Cards in a Linux Environment	<a href="http://www.linuxnet.com">www.linuxnet.com</a>
[NCSC]	National Computer Security Center	<a href="http://www.ncsa.uiuc.edu">www.ncsa.uiuc.edu</a>

**Table 25.3** (*Continuation*)

Reference	Company or organization	URL
[NFC Forum]	NFC Forum	<a href="http://www.nfc-forum.org">www.nfc-forum.org</a>
[NIST]	National Institute of Standards and Technology	<a href="http://www.nist.gov">www.nist.gov</a>
[NSA]	National Security Agency	<a href="http://www.nsa.gov">www.nsa.gov</a>
[NXP]	NXP	<a href="http://www.nxp.com">www.nxp.com</a>
[Oberthur]	Oberthur Smart Cards	<a href="http://www.oberthur.com">www.oberthur.com</a>
[OCF]	OCF	<a href="http://www.opencard.com">www.opencard.com</a>
[Octopus]	Octopus Card	<a href="http://www.octopuscards.com">www.octopuscards.com</a>
[OMA]	OMA	<a href="http://www.openmobilealliance.org">www.openmobilealliance.org</a>
[OMG]	OMG	<a href="http://www.omg.com">www.omg.com</a>
[OTP]	Open Trading Protocol	<a href="http://www.otp.org">www.otp.org</a>
[Oyster]	Oyster Card	<a href="http://www.oystercard.com">www.oystercard.com</a>
[PC/SC]	PC/SC Working Group	<a href="http://www.pcscworkgroup.com">www.pcscworkgroup.com</a>
[Proton]	Proton	<a href="http://www.protonworld.com">www.protonworld.com</a>
[Radicchio]	Radicchio	<a href="http://www.radicchio.org">www.radicchio.org</a>
[RFC]	RFC Server	<a href="http://www.rfc.net">www.rfc.net</a>
[RFID]	RFID Handbook	<a href="http://www.rfid.com">www.rfid.com</a>
[RSA]	RSA Inc.	<a href="http://www.rsa.com">www.rsa.com</a>
[Sagem Orga]	Sagem Orga	<a href="http://www.sagem-orga.com">www.sagem-orga.com</a>
[SEIS]	Secured Electronic Information in Society	<a href="http://www.seis.se">www.seis.se</a>
[Semper]	Semper	<a href="http://www.semper.org">www.semper.org</a>
[SEPT]	SEPT	<a href="http://www.sept.fr">www.sept.fr</a>
[SET]	Secure Electronic Transaction LLC	<a href="http://www.setco.org">www.setco.org</a>
[SIM Alliance]	SIM Alliance	<a href="http://www.simalliance.org">www.simalliance.org</a>
[SIT]	Fraunhofer Institute for Secure Information Technology (SIT)	<a href="http://www.sit.fraunhofer.de">www.sit.fraunhofer.de</a>
[Skiamade]	Ski amadé	<a href="http://www.skiamade.com">www.skiamade.com</a>
[Skidata]	Skidata	<a href="http://www.skidata.com">www.skidata.com</a>
[SmartTrust]	SmartTrust	<a href="http://www.smarttrust.com">www.smarttrust.com</a>
[Smartrust]	Smartrust	<a href="http://www.smarttrust.com">www.smarttrust.com</a>
[STM]	ST Microelectronics	<a href="http://www.st.com">www.st.com</a>
[Suica]	Super Urban Intelligent Card	<a href="http://www.jreast.co.jp/suica">www.jreast.co.jp/suica</a>
[Telesec]	Telesec	<a href="http://www.t-systems-telesec.com">www.t-systems-telesec.com</a>
[TETRA]	TETRA	<a href="http://www.tetramou.com">www.tetramou.com</a>
[TIA]	Telecommunications Industry Association	<a href="http://www.tiaonline.org">www.tiaonline.org</a>
[Top500]	Top 500 Supercomputer	<a href="http://www.top500.org">www.top500.org</a>
[TTT]	Teletrust	<a href="http://www.teletrust.de">www.teletrust.de</a>
[UMTS Forum]	UMTS Forum	<a href="http://www.umts-forum.org">www.umts-forum.org</a>
[Unicode]	Unicode Consortium	<a href="http://www.unicode.org">www.unicode.org</a>
[USB]	USB	<a href="http://www.usb.org">www.usb.org</a>
[Visa]	Visa International	<a href="http://www.visa.com">www.visa.com</a> <a href="http://www.visa.de">www.visa.de</a>
[WikiDE]	Wikipedia (German)	<a href="http://de.wikipedia.org">de.wikipedia.org</a>
[WikiEN]	Wikipedia (English)	<a href="http://en.wikipedia.org">en.wikipedia.org</a>
[Rankl]	Wolfgang Rankl	<a href="http://www.wrankl.de">www.wrankl.de</a>
[Wiley]	John Wiley & Sons, Ltd	<a href="http://www.wiley.co.uk">www.wiley.co.uk</a>
[Zeitcontrol]	Zeitcontrol Cardsystems GmbH	<a href="http://www.zeitcontrol.de">www.zeitcontrol.de</a>

# Index

- $\chi^2$  test, 165
- 16-bit code, 116
- 1G, 793
- 2-DES, 145
- 2.5G, 794
- 22C3, 732
- 2G, 794
- 2TDES, 145
- 3-DES, 145
- 32-bit code, 116
- 3FF, 31
- 3G, 794
- 3G system, 848
- 3GPP, 848
- 3GPP2, 813
- 3TDES, 145
- 4G, 795
- 7-bit code, 115
- 8-bit code, 115
  
- A-Netz, 793
- A-PET, 41
- ABS, 40
- Abstract Syntax Notation 1, 109
- Access conditions, 437, 474, 931
  - object oriented, 722
- Access control, 877, 931
- Access control descriptor, 121
  
- ACD, 121
- ACOS, 442
- Acquirer, 762, 931
- Acrylonitrile butadiene styrene, 40
- ACTIVATE FILE, 387, 423
- ACTIVATE RECORD, 386
- Activation sequence, 70, 485, 931
- Active mode, 964
- Active shield, 688
- ADF, 425, 851, 931
- Adhesion, 635
- Administrative access, 884
- Administrative commands, 355
- ADN, 811
- Advanced Encryption Standard, 141
- AES, 140, 851, 931
  - computation time, 141
- AFL, 405
- AFNOR, 931
- AFNOR location, 739
- AID, 429, 931
- AIP, 405
- Algorithm execution time, 920
- Amplitude shift keying, 286
- AMPS, 794, 931
- Analysis, 648, 932
- Analyzing a smart card, 926
- Anonymization, 932

- ANSI, 932  
ANSI code, 116  
ANSI T1, 848  
Answer to reset, 203  
Answer to select, 334  
Antenna array, 798  
Anticollision method, 932  
APDU, 221, 932  
API, 932  
APPEND RECORD, 364  
Apple Desktop Bus, 272  
Applet, 932  
Applet developer, 932  
Application, 932  
  files, 425  
  installation, 386  
  operator, 932  
Application areas for smart cards, 7  
APPLICATION BLOCK, 387, 781  
Application dedicated file, 425, 851  
Application generator, 563, 925  
Application identifier, 429  
Application management commands, 389  
APPLICATION MANAGEMENT  
  REQUEST, 390  
Application protocol data unit, 221  
APPLICATION UNBLOCK, 387  
Application-specific commands, 406, 495  
ARIB, 848  
Arimura, Kunitaka, 3  
ARM 7, 82  
ARM processor, 82  
ASC, 495  
ASCII, 115, 932  
ASK, 932  
ASN.1, 109, 856, 912, 933  
Assembler, 933  
Asymmetric cryptographic algorithm, 137,  
  145, 933  
Asynchronous data transmission, 933  
Asynchronous transmission protocol, 270  
Atomic operation, 480, 933  
Atomic write access, 439  
ATR, 203, 933  
  file, 212  
  practical examples, 214  
  test, 641  
  use for EEPROM write operations, 204  
ATS, 334  
Attack  
  brute-force, 137  
  chosen-ciphertext, 137  
  chosen-plaintext, 137  
  ciphertext-only, 137  
  differential fault analysis, 707  
  differential power analysis, 698  
  fault analysis, 707  
  invasive, 671  
  known-plaintext, 137  
  meet-in-the-middle, 144  
  noninvasive, 671  
  power analysis, 698  
  simple power analysis, 698  
  timing analysis, 718  
  trial and error, 557  
Attack tree analysis, 669  
Attackers, 668  
  classification of, 672  
Attacks, 668, 675, 682  
  attractiveness of, 674  
  avalanche effect, 668  
  classification of, 669  
  consequences of, 672  
  during production, 682  
  history of, 675  
ATTRIB, 325  
Attribute, 933  
Attributes of files, 438  
AuC, 808  
Authentic mode, 228  
Authentication, 166, 933  
  commands, 370, 374  
  dynamic, 167  
  mutual, 169  
  static, 167  
  unilateral, 168  
Authentication center, 808  
Authentication triple, 828  
Authenticity, 933  
Authorization, 933  
Auto top-up, 873  
Auto-eject reader, 934  
Avalanche criterion, 161  
Börsenevidenzzentrale, 785  
BAC, 895  
Background system, 755, 879, 934  
Bad case, 934  
Bad-day scenario, 934  
Barcode, 44  
Base station, 806  
Base station controller, 806

- Base transceiver station, 806  
Base wafer, 934  
Basic Access Control, 895  
Basic Encoding Rules, 110  
Basic interpreter, 493  
Basic multilingual plane, 116  
Basic test, 654  
BasicCard, 442, 520  
Baud, 934  
Bearer, 934  
Bearer services, 802  
Behavior test, 654  
Bellcore attack, 707, 934  
Bending stiffness, 635  
BER, 110, 934  
Bernoulli's theorem, 163  
Best-fit algorithm, 469  
BEZ, 785  
BGT, 267  
BIBO, 874  
Big endian, 934  
Bimetallic effect, 586  
Binding force, 134  
Biometric methods, 194  
Biometrics, 934  
BIP protocol, 235, 236  
Birthday paradox, 158  
Bit stuffer, 274  
Blackbox test, 934  
Blacklist, 732, 934  
Blank, 627  
Blank card, 628  
Block chaining, 268  
Block guard time, 267  
Block waiting time, 266  
Block-oriented algorithm, 134  
Bluetooth, 935  
BMP, 116  
Body, 422  
Bond-out chip, 574, 935  
Bonding, 582  
Boot loader, 92, 392, 455, 568, 574, 610,  
    935  
BPSK, 935  
Bring principle, 935  
Browser, 857, 935  
Brute-force attack, 137, 935  
BSC, 806  
BSI, 935  
BTS, 806  
Buffering, 8, 935  
Bug fix, 492, 936  
Burst, 936  
Bus scrambling, 694  
buses, 687  
BWT, 266  
Byte repetition, 256  
Bytecode, 501, 936  
C interpreter, 493  
C-450, 794  
C-Netz, 793  
CA, 178  
Caching, 485  
CAD, 735, 936  
CALCULATE EDC, 396  
CAMEL, 936  
CAP file, 515, 936  
Capability test, 654  
Capacitive coupling, 283, 287  
Card, 936  
    materials, 39  
    thickness, 636  
    types, 15  
    warpage, 636  
Card acceptor, 936  
Card Application Toolkit, 838  
CARD BLOCK, 389, 781  
Card body, 38, 936  
    injection molded, 586  
    monolayer, 585  
    multilayer, 586  
    symmetrical, 598  
    with antenna, 589  
Card component, 42, 936  
    layout, 38  
Card ejection, 741  
Card format, 29, 636  
    definition, 31  
    ID-00, 33  
    ID-000, 31  
    ID-1, 31  
    relative sizes, 36  
Card issuer, 936  
Card manager, 486  
Card modeling language, 936  
Card operating system, 442  
Card owner, 937  
Card possessor, 937  
Card producer, 937  
Card reader, 937  
Card registry, 487

- Card user, 937  
 Card withdrawal, 742  
 Cardholder, 937
  - verification, 188, 814
  - verification management, 486
  - verification method, 937
 Cardlet, 937  
 CardOS, 442  
 Cards
  - close-coupling, 291
  - embossed, 15
  - magnetic-stripe, 16
  - memory, 20
 CASCADE project, 82  
 Case, 223  
 CAST, 663  
 CAT, 838, 937  
 CAT\_TP, 237  
 CAVE, 814, 937  
 Cavity, 600, 937  
 CBC mode, 143
  - inner, 144
  - outer, 144
 CC, 660, 661, 811  
 CCID, 276  
 CCITT, 937  
 CCR, 735  
 CCS, 123, 155, 937  
 CDC, 277  
 CDMA, 798, 848, 938  
 CDMA 2000, 795, 848  
 Cell, 938
  - types, 800
 Cellular technology, 799  
 CEN, 12, 938  
 CEPS, 774, 938  
 CEPT, 802, 938  
 Certificate, 178, 938
  - revocation list, 938
  - structure, 178
 Certification, 938  
 Certification authority, 178, 900, 938  
 Chaining, 268  
 Challenge-response method, 167, 939  
 CHANGE CHV, 371  
 CHANGE REFERENCE DATA, 371  
 Character waiting time, 265  
 Check character, 214  
 CHECK COMPLETION, 393  
 Checkerboard pattern, 606  
 Chinese remainder theorem, 147, 939  
 Chip accepting device, 735  
 Chip card, 939
  - reader, 735
 Chip design, 572, 686  
 Chip hardware extension, 105, 106  
 Chip module, 50, 939  
 Chip on tape, 939  
 Chip size, 939  
 Chip-on-flex module, 54  
 Chosen-ciphertext attack, 137  
 Chosen-plaintext attack, 137  
 CHV, 188, 814, 939  
 CICC, 939  
 CICO, 874  
 Cipher block chaining, 143  
 Ciphertext, 134  
 Ciphertext-only attack, 137  
 Circuit-switched, 939  
 CISC, 79  
 Class, 939  
 Class byte codes, 222  
 Class file, 503, 940  
 Cleanroom VM, 940  
 Clear on deselect, 511, 940  
 Clear on reset, 510, 942  
 Clearing, 755, 940  
 Client, 940  
 Clock stop mode, 68  
 Clock supply, 69  
 Clone, 940  
 Cloning, 940  
 CLOSE APPLICATION, 358  
 Close-coupling card, 291  
 Closed application, 940  
 Closed purse, 940  
 Closed system architecture, 750  
 CMM, 940  
 CoD, 511  
 Code density, 940  
 Code division multiple access, 798  
 Code injection, 720  
 Code inspection, 649  
 Code walkthrough, 649  
 CODEC, 940  
 Coercivity, 940  
 Coextruded foil, 586  
 Coil, 589
  - on chip, 940
 Cold reset, 71, 221, 941  
 Collision, 941  
 CombiCard, 941

- Combined mode, 228, 230  
ComboOS, 442  
Command, 941  
    case, 223  
    chaining, 214, 222  
    interpreter, 539  
    processing, 447  
    processing time, 408, 417  
Command APDU, 221, 941  
Command set, 353  
    determining, 712  
Commands, 353  
    application management, 389  
    application-specific, 406  
    authentication of devices, 374  
    authentication of persons, 370  
    classification, 355  
    completion, 391  
    credit and debit card, 405  
    cryptographic algorithm, 378  
    data transmission, 398  
    database, 399  
    electronic purse, 402  
    file management, 384  
    file operation, 368  
    file selection, 356  
    hardware test, 395  
    private-use, 221  
    read, 358  
    search, 366  
    undocumented, 712  
    write, 358  
Commit buffer, 941  
Common Criteria, 660, 661, 941  
Common Electronic Purse Specifications, 774  
Common Stored Value Ticket, 869  
Communication  
    using I<sup>2</sup>C, 96  
    using MMC, 95  
    using SWP, 95  
    using T = 0 or T = 1, 93  
    using USB, 94  
COMP128, 142  
COMP128 algorithms, 142  
Company address, 1018  
COMPARE KEY, 392  
COMPARE NVM, 396  
Compatibility, 941  
Complementary logic, 702  
Completion, 391, 452, 609, 941  
    commands, 391  
    process, 394  
COMPLETION END, 393  
Compliance with standards, 923  
Composite evaluation, 663  
COMPUTE CRYPTOGRAPHIC CHECKSUM, 379  
COMPUTE DIGITAL SIGNATURE, 382  
Concurrent access, 440  
Confusion, 138  
Conservative access privileges, 727  
Contact designations, 65  
Contact field, 941  
Contact functions, 65  
Contact unit, 740  
Contactless cards, 9, 941  
    answer to reset, 296  
    capacitive coupling, 287  
    capacitive data transmission, 295  
    collision avoidance, 289  
    coupling components, 292  
    data transmission, 286  
    inductive coupling, 284  
    inductive data transmission, 293  
    power transmission, 285, 292  
    state of standardization, 290  
    structure, 285  
    test methods, 642  
Contactless memory card, 20  
Contactless processor card, 23  
Contactless smart card, 23  
Contacts, 36  
    dimensions, 38  
    location, 37, 639  
    resistance, 637  
Convention, 942  
    data transmission, 244  
Coprocessor, 103  
    for cryptographic algorithms, 102, 103  
CoR, 510  
Core foil, 586, 942  
Core voltage, 78, 942  
Cortex M3, 82  
COS, 442  
Coupon collector test, 165  
CP8, 942  
CPA, 782  
CRC, 125  
    calculation unit, 97  
    checksum, 125  
    generator polynomial, 126

- CREATE FILE, 385  
 Credential, 942  
 Credit card, 748, 942  
 Cross-application access, 472  
 Cryptanalysis, 133  
 Crypto 1, 679  
 Cryptocard, 942  
 Cryptoflex, 442  
 Cryptographic algorithm, 942
  - commands, 378
  - noise-free, 717
  - summary table, 145, 153
 Cryptographic checksum, 123, 155  
 Cryptographic library, 448  
 Cryptographic token, 909  
 Cryptography, 133  
 Cryptology, 133  
 Cryptoprocessor, 943  
 CT-API, 242, 943  
 CT-ICC, 241  
 Current consumption, 698  
 Customer card, 943  
 CVM, 486  
 CWT, 265  
 Cyberflex, 442  
 Cyclic redundancy check, 125  
 Cyclic structure, 435  
  
 DAM, 813  
 Danmønt, 6  
 DAP, 489  
 Data
  - compression, 129
  - generation, 611
  - integrity, 470, 471
  - structure, 109
 Data dictionary, 764  
 Data Encryption Algorithm, 138  
 Data Encryption Standard, 138  
 Data exchange, 921  
 Data group, 895  
 Data protection, 943  
 Data transmission, 69
  - commands, 398
  - contact cards, 243
  - contactless, 283
  - interception, 713
  - secure, 225
 Database commands, 399  
 Database file, 400  
 DBF, 400  
  
 DC/SC, 910  
 DCS, 804  
 DEA, 138, 943  
 DEACTIVATE FILE, 386, 423  
 DEACTIVATE RECORD, 386  
 Deactivation, 722
  - sequence, 70, 943
 Dead on arrival, 625  
 Deadlock, 118, 260  
 Deanonymization, 943  
 Debit card, 749, 943  
 DEBIT IEP, 404  
 Debiting cycle, 758  
 Debugging, 646, 943  
 DECIPHER, 380  
 Decision coverage, 651  
 DECREASE, 368  
 DECRYPT, 378  
 DECT, 813, 943
  - authentication module, 813
 Dedicated file, 424  
 Defect correction, 452  
 Defects – cost of searching for, 654  
 Defragmentation, 469, 943  
 Delamination, 635, 944  
 Delamination resistance, 636  
 DELETE, 236  
 DELETE FILE, 388  
 DELETE NVM, 394, 397  
 Delta evaluation, 663  
 DEMA, 705, 944  
 Denial of service, 678, 733  
 Depersonalization, 624, 944  
 DER, 110, 944  
 Derived key, 181  
 DES, 138, 944
  - algorithm, 138
  - computation time, 140
 DES-3, 145  
 DESFire, 944  
 Design, 648
  - criteria, 679
 Design model, 917  
 Destination management, 886  
 Deterministic, 944  
 Dettloff, Jürgen, 3  
 Development principles, 680  
 DF, 424, 944  
 DF name, 429, 944  
 DF separation, 467  
 DFA, 707

- DG, 895  
Dice, 579, 944  
Die, 579, 944  
Die bonding, 52, 594  
Differential crypto analysis, 944  
Differential fault analysis, 707, 945  
Differential power analysis, 698  
Diffusion, 138  
Digital Cellular System, 804  
Digital fingerprint, 945  
Digital printing, 597  
Digital signature, 155, 174, 899, 945  
    system architecture, 903  
    with appendix, 175  
    with message recovery, 175  
Digital Signature Algorithm, 151  
Digital Signature Standard, 151  
Digital watermark, 945  
Diners Club, 2  
Direct convention, 206, 243  
Direct issuing, 628  
Direct memory access, 99  
Directory file, 424  
Directory service, 945  
DISABLE CHV, 373  
DISABLE VERIFICATION  
    REQUIREMENT, 373  
Discoloration, 635  
Discrete logarithm problem, 151  
DISPATCHING, 624  
Distinguished Encoding Rules, 110  
Disturbance test, 650, 945  
Diversity, 228  
Divisor, 945  
DMA, 99  
DoA, 625  
Dolomiti Superski, 878  
DoS, 733  
Double access security, 728  
DOV-Modem, 790  
Downlink, 945  
Download, 945  
Downloadable program code, 491  
DPA, 698, 946  
DRAM, 92, 946  
DSA, 151, 946  
DSS algorithm, 151  
Dual IMSI, 842  
Dual-interface card, 24, 946  
Dual-rail bus, 702, 946  
Dual-slot mobile telephone, 946  
Dual-slot solution, 946  
Dummy cash dispenser, 192  
Dummy smart card, 731  
Dummy structures, 686  
Dump command, 681  
Duplication, 946  
Dynamic asymmetric authentication, 172  
Dynamic bending stress test, 636  
Dynamic key, 182  
Dynamic STK, 946  
Dynamic test, 653  
    techniques, 649  
Dynamic torsional stress, 637  
e-card, 861  
EAC, 895  
EAL, 949  
EAP, 947  
EBCDIC, 116  
EC, 152  
ECB mode, 143  
ECBS, 947  
ECC, 123, 127, 152, 893, 947  
    key size, 152  
EDC, 123  
EDGE, 794  
EEPROM, 85, 947  
    cell charging, 88  
    cell discharging, 89  
    dynamic programming, 606  
    error correction, 103  
    error detection, 103  
    reduced write time, 606  
    secure state, 89  
EF, 425, 947  
EF<sub>ATR</sub>, 427  
EF<sub>DIR</sub>, 427, 428  
eGK, 864  
EHIC, 865  
Eight-bit code, 115  
EIR, 808  
Electrical connections, 62  
Electrical properties, 61  
Electromagnetic fields, 638  
Electromagnetic radiation, 705  
Electron-beam writer, 576  
Electronic check, 947  
Electronic code book, 143  
Electronic health care card, 865  
Electronic health care professional card,  
    865

- Electronic money, 753  
   properties, 753
- Electronic payment transactions, 748
- Electronic prescription, 868
- Electronic purse, 749, 759, 947  
   commands, 402
- Electronic signature, 174
- Electronic ticket, 869
- Electronic toll system, 887  
   in-vehicle unit, 888  
   multi-lane, 888  
   onboard unit, 888  
   single-lane, 888
- Elementary file, 425
- Elementary time unit, 245
- Elliptic curve, 152  
   cryptosystems, 152
- Embedded coil, 591
- Embossed card, 15
- Embossed hologram, 45
- Embossing, 46, 947
- EMP, 733
- Emulator, 948
- EMV, 948  
   application, 776  
   specification, 776, 948
- EN 1546, 402  
   data elements, 763  
   files, 765
- ENABLE CHV, 373
- ENABLE VERIFICATION  
   REQUIREMENT, 373
- Encapsulation of application, 721
- ENCIPHER, 380
- Encoding alphanumeric data, 115
- ENCRYPT, 378
- Encryption, 133  
   algorithm operating modes, 142
- End-to-end communication, 805, 838
- End-to-end connection, 948
- Endianness, 948
- Endless loop, 260
- Endpoint, 235, 274
- Energy recycling, 631
- Engineering sample, 611
- Enrollment, 948
- ENVELOPE, 398, 834
- ENVELOPE STUFFING, 624
- Envelope stuffing, 948  
   machine, 624
- Environmental protection, 632
- EP SCP, 803
- Epilogue field ( $T = 1$ ), 264
- EPROM, 85, 948
- Equipment identity register, 808
- Equivalence classes, 653
- ERASE BINARY, 360
- Error ATR, 720
- Error correction code, 122, 123, 127, 128, 948
- Error counter, 948
- Error detection code, 122, 123, 949
- Error handling ( $T = 1$ ), 269
- Error recovery, 480, 729
- Error recovery function, 729
- ESD damage, 292
- ESD resistance, 639
- ETS, 949
- ETSI, 803, 848, 949
- ETSI Project Smart Card Platform, 803
- etu, 245, 949
- Eurocheque system in Germany, 783
- European Citizen Card, 893
- European Health Insurance Card, 865
- Eurosmart, 949
- Evaluation, 659, 949  
   ZKA criteria, 663
- Evaluation Assurance Level, 949
- Evaluation authority, 665
- Event independence, 163
- Exception, 501
- Executable native code, 493
- EXECUTE, 370, 494
- Execute structure, 436
- Execution paths, 651
- Execution speed, 485
- Exhaustive key search, 724
- Explicit EF selection, 431
- Extended Access Control, 895
- Extended length, 787, 866
- EXTERNAL AUTHENTICATE, 376
- Extra guard time, 209
- EZ-link card, 869
- f1, f2, f3, f4, f5, 851, 949
- F2F, 950
- F4, 147
- Fab, 950
- Face, 950
- Face to face, 105, 107
- False acceptance rate, 199
- False rejection rate, 199

- FAR, 199  
Faraday cage, 706  
FAT, 950  
FAT 16, 915  
FAT 32, 915  
Fault induction attack, 706  
Fault tree analysis, 669, 950  
FCOS, 52  
FD/CDMA, 798  
FDMA, 796  
FDN, 810  
FeliCa, 352, 950  
FETCH, 834  
FIB, 950  
FID, 426, 950  
    unique, 428  
File, 421  
    access conditions, 436  
    application, 425  
    attributes, 438  
    body, 422, 950  
    encryption, 440  
    header, 421, 950  
    layout in memory, 726  
    life cycle, 422  
    management, 421  
    names, 426  
    selection, 430  
    structure, 432, 950  
    type, 423, 951  
File access checking, 476  
File identifier, 426  
File management  
    commands, 384  
File manager, 536  
File operation  
    commands, 368  
File selection  
    commands, 356  
File structure, 950  
    classification, 432  
    cyclic, 435  
    data objects, 435  
    database, 436  
    execute, 436  
    linear fixed, 433  
    linear variable, 434  
    sequence control, 436  
    TLV, 435  
    transparent, 432  
FINEID, 893, 910  
FIPS, 951  
Firewall, 951  
Fixed terminal, 735  
Flammability, 638  
Flash, 951  
Flash EEPROM, 951  
Flash memory, 90  
    error correction, 103  
    error detection, 103  
Flash translation layer, 461  
Flash-light attack, 951  
Flip-chip, 52, 594  
Float, 749  
Floating gate, 87  
Floor limit, 951  
Focused ion beam, 697, 706  
Foil stamping, 599  
Footprint, 951  
Forcing, 687  
Forensics, 951  
Form factor, 951  
Format character, 207  
Foundry, 951  
Four-eyes principle, 681  
Fowler-Nordheim effect, 87  
FPLMTS, 794  
FRAM, 92, 951  
Frame, 952  
Free-running oscillator, 99  
Frequency division multiple access, 796  
Frequency hopping, 797  
Frequency monitoring, 693  
Frequency reuse, 799  
Frequency shift keying, 286  
Frequency spectrum, 795, 848  
FRR, 199  
FTL, 461, 952  
Full duplex, 952  
Functional extension, 452  
Fuse, 695  
GalaxSIM, 442  
Galois field, 127  
Gap test, 165  
Garbage collection, 470, 952  
Gate, 280, 880  
Gateway server, 859  
GeGKOS, 442  
GeldKarte, 952  
GENERAL AUTHENTICATE, 377  
Generating secret data, 618

- Generator polynomial, 126  
GET, 236  
GET CHALLENGE, 374  
GET DATA, 366  
GET PROCESSING OPTIONS, 405  
GET RESPONSE, 398  
GET STATUS, 391  
Ghost card, 733  
GIGAntIC, 442  
Glitch, 707, 710, 952  
Glob top, 583, 952  
Global Platform, 485, 952
  - API, 489
  - commands, 489
  - environment, 486
  - issuer security domain, 488
  - security domain, 487Global Positioning System, 890  
Glossary, 929  
Glue tape, 601  
Good case, 952  
GP, 485  
GPRS, 794  
GPS, 890  
Grötrupp, Helmut, 3  
Granularity, 952  
Graph theory, 119  
Greybox test, 653, 953  
Greylist, 953  
Groupe Spécial Mobile, 802  
GSM 1800, 804  
GSM 1900, 804  
GSM 900, 804  
GSM Association, 803  
GSM components, 806  
GSM Permanent Nucleus, 802  
GSM system, 802
  - architecture, 806Guilloche pattern, 42, 953  
  
HAL, 953  
Half duplex, 953  
Handle, 953  
Handover, 808, 953  
Happy-day scenario, 953  
Hard mask, 456, 953  
Hardware initialization, 534  
Hardware test, 641, 719
  - commands, 395HASH, 381  
Hash function, 156, 954  
HBCI, 954  
HCA, 866  
HCI, 280  
HD65901, 76  
Header, 421  
Health care card, 861
  - in Germany, 864Health care system, 861  
Health insurance card
  - in Germany, 861High update activity, 439, 833  
High-level design, 923  
High-order differential power analysis, 702  
Historical characters, 211  
HLR, 808  
HMAC, 159  
HMAC-MD5, 159  
HMAC-SHA-1, 159  
HO-DPA, 702  
Hologram, 45, 599, 954  
Home location register, 808  
Home net, 954  
Home zone, 844, 954  
Host, 280  
Hot stamp, 596, 599
  - method, 45Hot-electron injection, 91  
Hot-spot analysis, 657  
Hotlist, 954  
HPC, 865  
HSM, 954  
HTML, 954  
HTTP, 954
  - server, 912Huffman algorithm, 130  
Hybrid card, 955  
Hypertext, 955  
  
I<sup>2</sup>C interface, 955  
I<sup>2</sup>C bus, 96, 251  
I/O manager, 536  
ICAO, 955  
ICC, 955  
ICCD, 276  
ICCID, 630, 810  
ICCSN, 866  
ID, 955  
ID card, 955  
ID-00 format, 33  
ID-000 format, 31, 955  
ID-1 format, 15, 31, 955

- ID-2 format, 15  
ID-3 format, 15, 894  
IDEA, 141  
  computation time, 142  
Identification, 187, 893, 955  
Idle state, 68  
IEC, 955  
IEF, 425  
IFD, 735, 955  
IFSC, 264  
IFSD, 264  
IMEI, 810  
Implanter, 602, 956  
Implanting, 601  
Implementation, 956  
Implementation and test, 648  
Implicit EF selection, 431  
IMSI, 810, 827  
IMSI catcher, 826, 956  
IMT-2000, 794, 848  
In-mold labeling, 587  
In-vehicle unit, 888  
INCREASE, 368  
Indexed TAN, 956  
Individualization, 956  
Induction telegraphy, 792  
Inductive coupling, 283, 284  
Industry standard, 956  
Information field ( $T = 1$ ), 264  
Information field size for interface device,  
  264  
Information field size for the card, 264  
Initial booking terminal, 879  
Initial character, 206  
Initial etu, 208  
Initial waiting time, 204  
Initialization, 612, 956  
  cryptographic protection, 622  
Initialization vector, 144  
INITIALIZE IEP, 403  
Initializer, 956  
Injection molding, 586  
Inkjet printing, 599  
Inmarsat, 814  
Input commands, 840  
Insertion cycles, 640  
INSTALL, 390  
Instant issuing, 628  
Instrumentation, 651, 956  
Intel 4004, 79  
Intelligent memory card, 956  
Interface characters, 207  
Interface device, 735  
Interleaving mode, 616  
INTERNAL AUTHENTICATE, 374  
Internal EF, 423  
Internal elementary file, 425  
Internal foil, 956  
International mobile subscriber identity,  
  827  
International Telecommunication Union,  
  794  
Internet smart card, 956  
Interoperability, 957  
Interpreter, 493, 957  
Interpreter-based operating system, 957  
INVALIDATE, 386  
Inverse convention, 206, 243  
IP address, 235  
IPES, 142  
Iridium, 814  
ISDN, 755, 957  
ISO, 11, 957  
  file system, 957  
  protocol, 254, 957  
ISO location, 739  
Issuer security domain, 486  
IT security, 899  
ITSEC, 957  
ITU, 794  
IuKDG, 900  
IV, 144  
IVU, 888  
Jative card, 957  
Java, 499, 958  
  accelerator, 101  
  API, 507  
  class file, 503  
  future prospects, 519  
  programming language, 500  
  properties, 501  
  software development, 514  
Java Card, 499  
  API, 958  
  application selection, 509  
  atomic operations, 510  
  classic, 518  
  clear on deselect RAM, 510  
  clear on reset RAM, 510  
  command dispatching, 509  
  connected, 518

- Java Card (*Continued*)  
     cryptography , 518  
     deleting objects, 510  
     dispatcher, 509  
     execution speed, 517  
     export restrictions, 518  
     file system, 517  
     firewall, 508  
     persistent objects, 510  
     runtime environment, 958  
     transaction integrity, 510  
     transient objects, 510  
     virtual machine, 958
- Java card, 958
- Java Card 3.0, 518
- Java Card Forum, 500, 958
- Java Card virtual machine, 504
- Java Development Kit, 958
- Java virtual machine, 502  
     evaluation, 506  
     stack, 506
- Java VM, 502  
     heap, 506
- JCF, 500
- JCOP, 442, 958
- JCVM, 504
- JFFS/JFFS2, 461
- JIT compiler, 502
- Just-in-time compiler, 502
- Just-in-time production, 627
- JVM, 502
- Kc, 810
- KCV technique, 619
- Kerckhoff's principle, 134, 674, 958
- Kerckhoff, Auguste, 134
- Kernel, 958
- Key, 959  
     data, 183  
     diversification, 182  
     management, 180, 920, 959  
     versions, 182
- Key space size, 137
- Keyed hash algorithm, 159
- Ki, 810
- Kinegram, 45, 959
- Known-plaintext attack, 137
- KVK, 861
- L<sub>c</sub> field, 222
- L<sub>e</sub> field, 222
- LAC, 811
- LAI, 811
- Laminate, 959
- Lamination, 586
- Landing zone, 782
- Large cell, 800
- Large files, 366, 436
- Laser cutter, 959
- Laser engraving, 47, 959
- Lasing, 47
- Latent defect, 654
- Layer-7 chaining, 222, 381, 390
- Layered operating system, 441
- LDS, 895
- Lead time, 959
- Lead-frame module, 57, 959
- LEN field, 262
- Lettershop, 624
- LFSR, 160
- License fees, 848
- Life cycle, 569, 959
- Lifekey, 653
- LIFO memory management, 468
- Light attack, 708, 959
- Light transmittance, 638
- Linear feedback shift register, 160
- Linear fixed structure, 433
- Linear variable structure, 434
- Linker, 960
- Linux, 242, 499, 521
- Little endian, 960
- LLC, 280
- LOAD, 390
- Load agent, 762, 960
- LOAD APPLICATION, 390
- Load modulation, 287
- Loader, 960
- Loading center, 785
- Lobby machine, 784
- Local public transport, 869
- Local public transport company, 869
- Logical channel, 233, 960
- Logical data structure, 895
- Longitudinal redundancy check, 124
- LRC, 124
- m-Commerce, 961
- M/Chip, 960
- M2M, 960
- M2M application, 107
- MAC, 123, 155, 280, 960

- Machine readable travel document, 894  
Machine readable zone, 894  
Machine to machine, 107  
Macrocell, 800  
Magnetic card, 960  
Magnetic stripe, 960  
    signal amplitude, 635  
    storage capacity, 17  
Magnetic-stripe card, 16, 960  
MANAGE CHANNEL, 399  
MANAGE SECURITY ENVIRONMENT, 378  
Management data, 961  
Manipulation of EF<sub>ARR</sub>, 730  
Manufacturing, 567  
Maosco, 961  
Mask, 575, 576, 961  
Mass memory interface, 104  
Master file, 424  
Master key, 181  
Master-slave operation, 275  
Master-slave relationship, 201, 834  
MasterCard, 31, 663, 782  
Matching on chip, 199  
mc2, 31  
MChip, 782  
MD5, 159  
Mechanism strength, 662  
Meet-in-the-middle attack, 144  
MEL, 493  
Memorandum of Understanding, 803  
Memory  
    checksum, 721  
    design, 687  
    encryption, 691  
    expansion, 104  
    footprint, 961  
    organization, 457  
    paged, 467  
    reuse, 725  
    scrambling, 690  
    types, 82  
Memory card, 8, 20, 961  
    protocols, 248  
Memory management, 100  
Memory management unit, 100, 494, 721  
Memory protection, 484, 961  
Memory protection unit, 963  
Merchant card, 961  
Message authentication code, 123, 155  
Message structure, 221  
Metalization layer, 688, 689  
MF, 424, 961  
MFC, 783  
Micardo, 442  
Microbrowser, 857, 961  
Microcell, 800  
Microcontroller, 73, 961  
Microcontrollers  
    availability, 75  
    chip area, 75  
    functionality, 74  
    manufacturing cost, 73  
    security, 75  
    self-destructive, 682  
Micromodule, 51  
Microprobe needle, 696  
Microprocessor, 961  
Microprocessor card, 21  
Microtext, 44  
Midlet, 962  
Mifare, 352, 962  
Mifare Classic, 679  
Milenage algorithm, 142, 851, 962  
Miller–Rabin test, 150  
Mini card, 31  
mini-UICC, 31, 962  
MKT, 241, 962  
    specification, 237, 241  
MLI, 46  
MM method, 48  
MMC, 95, 277  
    transmission protocol, 277  
MMU, 100, 494, 721, 962  
MNC, 811  
Mobile equipment, 806  
Mobile station, 806  
Mobile switching center, 801, 808  
Mobile telecommunication system, 795  
MOC, 199  
Modular exponentiation, 147  
Module, 581, 962  
    floating implantation, 601  
    multichip, 105  
    types, 50  
    with integrated coil, 589  
Module producer, 962  
Module/interface concept, 449  
Mold body, 57  
Mondex, 962  
Mono-application smart card, 962  
Monofoil, 585

- Monofunctional smart card, 962  
Monolayer card, 962  
Moore's Law, 76  
More time, 258  
Moreno, Roland, 3  
Morse code, 792  
MOSFET, 87  
MoU, 803  
MPCOS, 442  
MPU, 963  
MRTD, 894  
MRZ, 894  
MSC, 277, 808  
MSE, 378  
MSISDN, 811  
Multi-megabyte card, 24, 854  
Multiapplication smart card, 963  
Multicopy sheet, 595, 963  
Multifunctional card, 963  
Multifunctional chipcard, 783  
Multilayer card, 963  
Multilevel cell, 104  
Multiple encryption, 144  
Multiple laser image, 46  
Multiple-access methods, 795  
Multiproject wafer, 578, 963  
Multitasking, 483, 963  
Multithreading, 963  
Multiuser system, 399  
Multos, 442, 519, 963  
MUSCLE, 242  
MUTUAL AUTHENTICATE, 377  
Mutual authentication, 169
- NAD, 262  
Name space, 963  
NAND flash, 963  
Native card, 964  
Native code, 964  
Native program code, 491  
Native smart card operating system, 964  
NBS, 964  
NCSC, 964  
Near field communication, 348  
Needle probe, 641  
Negative file, 964  
Negotiable mode, 217  
Network card, 964  
NFC, 348, 964  
NFC Forum, 964  
Nibble, 964  
NIST, 964  
Node address, 262  
Noise-free algorithm, 137  
Noiseless, 964  
Nonrepudiation, 134, 964  
Nonvolatile memory, 965  
NOR flash, 91, 965  
NPU, 965  
NRZ coding, 293  
NRZI, 274  
NSA, 965  
Null PIN, 371, 965  
    method, 189  
Null-PIN method, 371  
Numbering, 965  
Numbering scheme  
    classificatory, 921  
    identificatory, 921  
    mixed, 921  
Object, 965  
OBU, 888  
OCF, 241, 965  
OCR-B, 894  
Octopus card, 869  
Offcard application, 965  
Offcard VM, 515  
Offline content, 860  
Offset printing, 596  
OMC, 808  
Onboard unit, 888  
Oncard application, 965  
Oncard matching, 199, 965  
Oncard VM, 515  
One-time password, 966  
One-to-one, 966  
One-way function, 966  
Online behavior, 729  
Online content, 860  
Online system, 859  
OP, 485, 966  
OPEN, 486  
Open application, 966  
Open Card Consortium, 241  
Open Card Framework, 241  
Open Gate, 886  
Open Platform, 485, 966  
Open platform, 499, 966  
Open purse, 966  
Open smart card operating system, 966  
Open source software, 710

- Open system architecture, 750  
Open terminal architecture, 499  
Operating system, 441, 966  
  initialization, 534  
  kernel, 538  
  layer separation, 720  
  main loop, 534  
  undocumented commands, 712  
Operating system activities  
  concealment, 721  
Operating system producer, 966  
Operation and maintenance center, 808  
Operational commands, 355  
Operational data, 880  
Optical fault induction attack, 708, 959  
Optical memory card, 25, 966  
Orange Book, 660  
Order data, 609  
OSI communication model, 203  
  layers 1, 2, & 7, 203  
OTA, 499, 805, 838, 966  
Output commands, 840  
Overlay, 967  
Overlay foil, 586, 598  
Oyster card, 869
- PA, 698  
PACE, 377  
Package, 967  
Packet-switched, 967  
Padding, 154, 967  
Pads, 53  
Page-oriented, 967  
Parallel data transmission, 967  
Parity bit, 967  
Passage frequency, 880  
Passivation, 967  
Passivation monitoring, 692  
Passive mode, 964  
Passport, 894, 967  
Patch, 452, 967  
Patent, 968  
Pay before, 748, 968  
Pay later, 748, 968  
Pay now, 748, 968  
Payment cards, 782  
Payment flows, 761  
Payment systems, 747  
  system architecture, 755  
PayPass, 782  
payWave, 782
- PC, 40  
PC ASCII, 115  
PC Card, 738  
  terminal, 738  
PC/SC, 237, 968  
  crypto service provider, 239  
ICC, 241  
ICC resource manager, 240  
ICC service provider, 239  
ICC-aware application, 239  
IFD, 240  
IFD handler, 240  
  service provider, 239  
  specification, 237, 238  
PCB field, 262  
PCD, 968  
PCK, 219  
PCMCIA, 738  
  terminal, 738  
PCS, 804  
PDF 417, 44  
PEM, 178  
Pentium floating-point error, 491  
PERFORM SCQL OPERATION, 401  
PERFORM SECURITY OPERATION,  
  379  
PERFORM TRANSACTION  
  OPERATION, 401, 402  
PERFORM USER OPERATION, 401, 402  
Performance, 484  
  optimization, 485  
Persistence, 968  
Personal Communication System, 804  
Personal identification number, 188  
Personal unblocking key, 189  
Personalization, 619, 968  
  cryptographic protection, 622  
  tests, 623  
  throughput, 621  
Personalizer, 968  
Perspiration resistance, 640  
PES, 141  
PET, 41  
PETP, 41  
Phase 1, 803  
Phase 1, Phase 2, Phase 2+, 968  
Phase shift keying, 286  
Phising, 969  
Phone card contact assignments, 791  
Photoresist, 576  
Physical properties, 29

- Physical transmission layer, 243  
 PICC, 969  
 Pick-and-place robot, 57  
 Picocell, 800  
 Picture card, 626  
 PIN, 188, 370, 969  
     modifiable, 189  
     static, 189  
 PIN comparison  
     power analysis, 715  
     timing analysis, 716  
 PIN letter, 624  
 PIN pad, 189, 969  
 Pipe, 274, 280  
 Pipelining, 616, 969  
 PIX, 430  
 PKCS #1 . . . PKCS #15, 969  
 PKCS #15, 909  
 PKI, 969  
 Plaintext, 134  
 Plastic card, 969  
 Plasticizer stability, 640  
 Platform independence, 508  
 Platform-independent implementation, 449  
 PLL circuit, 98  
 PLMN, 793, 969  
 Plug-in, 857, 969  
     card, 31  
     SIM, 812  
     stamping, 604  
 PoD, 625  
 Poker test, 165  
 Polling, 834, 969  
 Polycarbonate, 40  
 Polyethylene terephthalate, 41  
 Polyvinyl chloride, 39  
 Port, 235  
 Portable terminal, 735  
 POS, 970  
 POST, 236  
 Postal optimization, 624  
 Postpaid, 889, 970  
 Power analysis, 698  
 Power interruption, 714  
 Power management, 98  
 Power-on reset, 970  
 POZ, 783  
 PP, 661  
 PPC system, 609  
 PPS, 217, 812  
     request, 217  
     sequence, 220  
 PPS<sub>1</sub>, 219  
 PPS0, 218, 219  
 Prepaid, 786, 889, 970  
     memory card, 757  
     SIM, 845, 970  
 Prepersonalization, 970  
 Primary flat, 575  
 Prime number test (probabilistic), 150  
 Primitive, 855  
 Printed coil, 593  
 Printing methods, 596  
 Private-use command, 221  
 PRNG, 160  
 Proactivity, 970  
 Probabilistic, 970  
 Probing, 687  
 Procedural programming, 970  
 Processing data, 609  
 Processing time  
     data transfer, 410  
     NVM operations, 409  
 Processing time estimation, 407  
 Processor, 970  
 Processor card, 8, 21, 970  
 Processor types, 79  
 Production, 567  
     on demand, 625  
 Profile, 611, 971  
 Profile description, 611  
 Program code, 971  
 Prologue field ( $T = 1$ ), 262  
 Proprietary, 971  
 Proprietary application identifier extension, 430  
 Protection profile, 971  
 Protocol control byte, 262  
 Protocol Parameter Selection, 217  
 Protocol Type Selection, 217  
 Proton, 775, 971  
 Proximity card, 297  
 Pseudonymization, 971  
 Pseudorandom number generator  
     initial value, 160, 723  
     seed number, 723  
 PSO, 379  
 PSTN, 793, 971  
 PTS, 217, 971  
 PTSS, 218

- Public card phones in Germany, 789  
Public land mobile network, 793  
Public switched telephone network, 793  
Public-key algorithm, 971  
PUK, 189, 372, 971  
Pull principle, 972  
Pull technology, 972  
Pullup resistor, 69  
Purse holder, 972  
Purse provider, 762, 972  
Purse-to-purse transaction, 754, 972  
Push technology, 972  
PUT, 236  
PUT DATA, 366  
PUT KEY, 391  
PVC, 39, 972
- Quality assurance, 633  
Quick, 760, 972  
Quota, 470, 972
- R-UIM, 813, 848  
RAM, 92, 805, 972, 973  
Random delay, 701  
Random number generator, 97, 160, 723, 972  
Random numbers, 159
  - coupon collector test, 164
  - gap test, 164
  - generation, 160
  - long run test, 164
  - monobit test, 164
  - pattern test, 164
  - poker test, 164
  - quality of, 163
  - runs test, 164
  - serial test, 164
- Random wait states, 701  
Randomizer, 695  
RATS, 332  
RC oscillator, 98  
READ BINARY, 359, 360  
Read commands, 358  
Reader, 973  
Real merchant card, 786  
Real-time clock, 743  
Receipt, 488  
Receive sequence counter
  - T = 1, 264
- Record, 973  
Recovery test, 650
- Recycling, 630  
Redlist, 732, 973  
Reed–Solomon code, 44, 127  
Reed–Solomon error correction, 128  
Reed–Solomon code, 471  
REGISTER, 385  
Registered identifier, 429  
Registration authority, 973  
REHABILITATE, 387  
Reject handling, 588  
Relative humidity, 636  
Relay attack, 733, 973  
Release sample, 612  
Remote applet management, 805, 841, 973  
Remote file management, 805, 840, 973  
Remote-coupling cards, 296  
REMOVE APPLICATION, 390
- Replay, 169  
Replay attack, 679  
Reset, 973  
RESET RETRY COUNTER, 372  
Resistance to chemicals, 639  
RESIZE, 365  
Resources and access rules, 476  
Resources and security attributes, 475  
Response, 973
  - APDU, 221, 224, 973
  - data, 611, 619, 973
- Reticule, 973  
RETRIEVE DATA, 366, 436  
Return code classification, 224  
Return code manager, 448, 536  
Review, 649  
RFID, 283, 973  
RFID zapper, 732  
RFM, 805, 840, 973  
RID, 429
- RISC, 81  
RMI, 973  
RNDIS, 277  
RNG, 97
- Road pricing, 888  
Roaming, 973  
Robustness, 39  
Roll back, 729, 973  
Roll forward, 974  
Roll-on method, 45, 599  
ROM, 84, 974  
ROM mask, 974  
ROMable applet, 516  
ROMed application, 974

- RSA, 146, 974  
computation time, 149  
key generation, 148, 151  
private key, 147  
public key, 147  
Rule-based programming, 975  
RUN GSM ALGORITHM, 406  
Run-length encoding, 130  
  
S<sup>2</sup>C, 278, 975  
Saliva resistance, 640  
Salt, 975  
Salting, 725  
SAM, 762, 975  
Sandbox, 502, 975  
SAT, 833  
SC 100, 82  
SC21, 76  
Scheduler, 484  
SCOPE, 975  
SCQL, 399  
object description table, 401  
privilege description table, 401  
Scrambling, 975  
bus, 694  
memory, 690  
Scrap rate, 975  
Scratch card, 975  
Scratch field, 47  
Screen printing, 596, 598  
Screening, 627  
Script, 975  
SCWS, 912  
SDL notation, 117  
SDMA, 798  
Seal layer, 47  
SEARCH BINARY, 367  
Search commands, 366  
SECCOS, 438, 787, 976  
Secret generation, 618  
Secret number, 188  
Secret-key algorithm, 976  
Sector, 976  
Secure application module, 762  
Secure Channel Protocol, 226  
Secure data transmission, 225  
Secure messaging, 225, 226, 728, 976  
Secure operating system, 446  
Secure signature creation device, 976  
Secure state, 89  
  
Security, 744  
by obscurity, 134  
features, 42, 49  
function, 851  
technology, 133  
Security chips, 50  
Security component, 351  
Security domain, 487  
Security environment, 976  
Security features, 728  
Security module, 745, 976  
Security requirements, 976  
Security target, 976  
Security targets, 661  
Security token, 25  
Seed number, 160, 723, 976  
SEEK, 366  
SEIS specification, 910  
SELECT, 356  
Selection  
  explicit, 431  
  implicit, 431  
  using a path name, 432  
Selective cell, 801  
Selective etching, 688  
Self-destruction, 682  
SEMA, 705, 976  
Semi-formal description, 523  
Semiconductor fabrication  
  elapsed time time, 577  
  masks, 576  
  processing time, 577  
Semiconductor technology, 76, 685  
Send sequence counter, 231  
  (T = 1), 264  
Send sequence number, 264  
Sensing shield, 688  
Sequence control, 472, 977  
Serial data transmission, 977  
Serial test, 165  
Server, 977  
Service provider, 762, 977  
Servlet, 915  
Session, 977  
Session key, 182, 977  
SET, 178  
SET DATA, 366, 436  
SET STATUS, 391  
Seven-bit code, 115  
SHA, 159  
SHA-1, 159, 977

- SHA-2, 159  
Shadow account, 785  
Shared batch, 578  
Shared secrets, 681  
Shield, 688  
Short FID, 428, 977  
Short file identifier, 428  
Short-circuit protection, 743  
Shrink, 978  
Shrink process, 78  
Shutter, 714, 746, 978  
Side channel attack, 978  
Side-channel attack, 702  
Sieve of Eratosthenes, 149  
SigG, 900  
SIGN DATA, 378  
Signal burst, 978  
Signature, 155, 174  
Signature Act, 978  
Signature application, 909  
Signature card, 903, 906, 978  
    commands, 908  
    DIN specification, 903  
    implementation provisions, 902  
    key generation, 906, 907  
    legally compliant, 902  
    legally noncompliant, 902  
    pseudonym, 901  
Signature Ordinance, 901  
Signature panel, 44  
    nonerasable, 44  
Signature strip, 978  
Signaturgesetz, 900  
Signaturverordnung, 901, 978  
SigV, 901  
SIM, 802, 806, 811, 978  
SIM Alliance, 858, 979  
SIM Application Toolkit, 833, 979  
SIM detection, 815  
SIM lock, 845, 979  
SIM service table, 810  
SIM Toolkit, 979  
SIMEG, 803, 979  
SIMphonIC, 442  
Simple power analysis, 698  
Simplex, 979  
SIMply, 442  
SIMtonIC, 442  
Simulator, 979  
Simultaneous engineering, 647  
Single sign-on, 980  
Single Wire Protocol, 95  
Single-byte reception, 258  
Single-card printing, 595  
    machine, 587  
Single-card production, 627  
Single-level cell, 104  
Single-user system, 399  
Single-Wire protocol (SWP), 278  
Sixteen-bit code, 116  
Ski Amadé, 878  
Ski card, 880  
Ski pass, 878  
    system, 878  
Skimming, 980  
SkySIM, 442  
Sleep mode, 68  
Small-OS, 521  
Smart card, 18, 980  
    analyzing an unknown card, 926  
    commands, 353  
    history, 2  
    ISO 10202-1 life cycle, 570, 572  
    life cycle, 569  
    production volume, 4  
    reader, 735  
    security, 667  
    service life, 918  
    simulator, 926  
    tasks and roles, 568  
    terminal, 735  
    typical application areas, 7  
    web server, 912  
    with chip, 776  
Smart card operating system, 441  
    hardware recognition, 456  
Smart label, 980  
Smart object, 981  
Smartcard, 981  
SMG9, 803  
SMIME, 178  
SMS, 811, 981  
SMSC, 859  
Sniffer, 981  
Socket, 235  
Soft mask, 456, 981  
Soft SIM, 981  
Software defects – cost of correction, 646  
Software life cycle, 647  
Software quality metrics, 660  
Software test, 719  
Solovay–Strassen test, 150

- Sonotrode, 593  
 SPA, 698, 981  
 SPA/DPA-resistant, 981  
 Space division multiple access, 798  
 Special modules, 59  
 Specific mode, 217  
 Specification, 982  
 Specifications (annotated directory), 999  
 Spectral distribution, 165  
 Spectral test, 165  
 Spike, 710  
 Spread-spectrum technology, 798  
 SQL, 399  
 SRAM, 92, 982  
 SSCD, 976  
 SSL, 178, 226, 236, 859, 982  
 SST, 810  
 Stack, 982  
 Standard cell, 686  
 Standardization, 10  
 Standards, 982
  - annotated directory, 999
  - compliance, 923
  - generation, 10, 12
 STARCOS, 438, 442, 982  
 STARSIM, 442  
 State machine, 118, 982
  - for completion, 395
  - for life cycle implementation, 611
  - $T = 0$ , 259
  - theory, 118
 Statement coverage, 651  
 Static asymmetric authentication, 170  
 Static electricity, 639  
 Static test techniques, 649  
 Steganography, 650, 982  
 Stepper, 577  
 Stepping, 576  
 STORE DATA, 391  
 Storing secrets, 691  
 Structural formulas of card materials, 40  
 Structured card query language, 399  
 Structured query language, 399  
 Subscriber identity module, 802, 806  
 Suica, 869  
 Sun, 500  
 Super PIN, 189  
 Super smart card, 43, 982  
 Super Urban Intelligent Card, 869  
 Supplementary hardware, 93  
 Supply current, 65
  - Supply shield, 688
  - Supply voltage, 62
  - Surface profile, 639
  - Surface roughness, 640
  - SWP, 95, 278, 982
    - protocol stack, 280
  - Symmetric cryptographic algorithm, 137, 138, 983
  - Symmetrical card structure, 598
  - Synchronous data transmission, 248, 983
  - System architecture, 751
  - System integration, 648
  - System on card, 983- $T = , 254$
- $T = 0, 255$ 
  - transmission protocol, 255
- $T = 1, 260$ 
  - transmission protocol, 260
- $T = 14, 271$ 
  - transmission protocol, 271
- $T = 0, 983$
- $T = 1, 983$
- $T = CL, 983$
- $TA_1, 208$
- $TA_2, 211$
- $TA_i, 209$
- $TA_i, 210$
- TAB, 53
- Tachosmart, 887
- Tag, 983
- Tape out, 983
- Tape test, 983
- Tape-automated bonding, 53
- Target of evaluation, 660, 983
- $TB_i, 210$
- TC, 905
- TC display, 48
- $TC_1, 209$
- $TC_2, 210$
- $TC_i, 211$
- TCOS, 442, 984
- TCP/IP, 755, 984
- TCP/IP protocol, 234
- TCSEC, 660, 984
- TD/CDMA, 798
- TDES, 145
- TDMA, 796
- Tear save test, 650
- Telecommunication, 789
- Telegraph, 792

- Telephone bandwidth, 802  
Temperature monitoring, 694  
Template, 113  
Temporary key, 182  
Temporary mobile subscriber identity, 827  
Terminal, 735, 984  
    connection, 237, 751  
    PC Card, 738  
    verifying as genuine, 192  
TERMINAL RESPONSE, 834  
TERMINATE CARD USAGE, 389  
TERMINATE DF, 388, 423  
TERMINATE EF, 388, 423  
Test, 984  
    Test application, 655  
    Test key, 653  
    Test levels, 728  
    Test methods  
        contactless smart cards, 642  
        proximity cards, 644  
        software, 645  
        vicinity coupling cards, 645  
    Test mode, 578, 584  
    switchover, 695  
TEST NVM, 396  
Test pads, 696  
TEST RAM, 396  
Test ROM, 641  
Test specification, 646  
Test strategy, 649, 654  
Test zone, 247  
Testing, 646  
Testing random numbers, 163  
Testing techniques, 649  
Tests  
    card body, 634  
    microcontroller hardware, 641  
    production, 633  
    qualification, 633  
TETRA, 813, 984  
Text messages, 811  
THC, 861  
Thermal dye sublimation printing, 595, 598  
Thermal transfer printing, 595, 598  
Thermochrome display, 48  
Third Form Factor, 31  
Third Generation Partnership Project, 813,  
    848  
Thirty-two-bit code, 116  
Thread, 984  
Threat tree analysis, 669  
Three-factor-authentication, 166  
Ticket, 869  
Time division multiple access, 796  
Time slicing, 484  
Time stamp, 985  
Timer, 96  
Timing analysis, 718  
Timing attack, 718, 985  
TLS, 226  
TLV, 985  
    length, 110  
    object, 113  
    structure, 110  
    tag, 110  
    value, 110  
TMSI, 811, 827  
TOE, 661  
Token, 25  
Toolchain, 985  
Top-level CA, 985  
Top-up, 873  
Touch-in/touch-out, 874  
TPDU, 221, 254, 985  
Traceability, 570  
Tracer, 985  
Transaction, 985  
Transaction number, 985  
Transfer card, 756, 985  
Transferability, 188  
Transient, 985  
Transition zone, 247  
Transmission error signaling ( $T = 0$ ), 256  
Transmission hologram, 45  
Transmission protocol, 254, 985  
    comparison, 271  
    summary, 255  
    USB, 272  
Transmission protocol data unit, 221, 254  
Transparent structure, 432  
Transport PIN, 189  
Transport protocol, 986  
Transportation company, 874  
Transportation systems, 869  
Trapdoor, 986  
Tri-state, 252  
Trial and error, 557  
Triple DES, 145, 986  
Triple-interface card, 986  
Trivial PIN, 189, 986  
TRNG, 160  
Trojan horse, 681, 723, 986

- Trust center, 178, 900, 903, 905, 986  
Trusted third party, 986  
Trustworthiness level, 662  
TSL, 859, 986  
TTA, 848  
TTC, 848  
TTP, 986  
Tunnel, 986  
Tunnel effect, 87  
Tunnel oxide layer, 87  
Two-factor authentication, 166  
Two-chip solution, 105  
  
UART, 93, 987  
UATK, 814, 838, 987  
UCS, 116, 808, 987  
  transformation format, 116  
UCS-2, 117, 808  
UCS-4, 117  
UICC, 806, 850, 987  
UIM, 987  
UIM Application Toolkit, 838  
Ultraviolet light, 640  
Ultraviolet text, 44  
Umbrella cell, 801  
UML, 987  
UMTS, 848, 987  
  system, 848  
UNBLOCK CHV, 372  
Unicode, 116, 987  
Unilateral authentication, 168  
Unique chip number, 680  
Unique identifying features, 625  
Universal Character Set, 116  
Universal integrated circuit card, 806  
Universal subscriber identity module, 850  
UniverSIM, 442  
UPDATE BINARY, 359  
Uplink, 987  
Upload, 988  
USAT, 838, 852  
USAT Interpreter, 852  
USB, 94, 272, 988  
  bulk transfer, 275  
  composite device, 276  
  control transfer, 275  
  data packet, 275  
  device classes, 276  
  enumeration, 276  
  ICC, 274  
  interrupt transfer, 275  
  isochronous transfer, 275  
  protocol, 94  
  resume, 276  
  suspend, 276  
Use access, 883  
User, 988  
  data, 880, 988  
  interface, 922  
User identification transferability, 195  
User mode, 584  
  switchover, 695  
USIM, 849, 850, 988  
USIM Application Toolkit, 838, 852, 988  
UTF, 116  
UTRAN, 848  
  
Validation, 988  
Value-added service, 833, 988  
Vanity card, 627  
Variable-length encoding, 130  
VAS, 833  
VERIFY, 370  
VERIFY CERTIFICATE, 383  
VERIFY CRYPTOGRAPHIC  
  CHECKSUM, 379  
VERIFY DIGITAL SIGNATURE, 382  
VERIFY SIGNATURE, 378  
Vertical system integration, 106  
Vibration, 640  
Vicinity card, 297, 880  
Vicinity integrated circuit card, 344  
Virgin card, 988  
Virtual machine, 502, 988  
Virtual merchant card, 786, 988  
Virtual smart card, 989  
Virus, 723  
Visa, 31, 782, 989  
Visa Cash, 760, 989  
Visa Easy Entry, 989  
Visa Mini, 31, 989  
Visa Open Platform, 485  
Visitor location register, 808  
VLR, 808  
VM, 502  
Volatile memory, 989  
Voltage class, 64  
Voltage monitoring, 693  
VOP, 485, 989  
VSDC, 782  
VSI, 106

- Wafer, 575, 989  
Waiting time extension, 258, 267  
Waiting times ( $T = 1$ ), 265  
WAP, 854, 989  
Warm reset, 71, 221, 989  
Warpage, 636  
Watchdog, 96  
Waterfall model, 647  
WCDMA, 798  
Wear leveling, 461, 989  
Wear resistance of inks, 640  
Wear test for magnetic stripe, 640  
Web addresses, 1018  
Web server, 913  
WEF, 425  
White plastic, 569, 989  
White-light reflection hologram, 45  
Whitebox test, 650, 990  
Whitelist, 668, 732, 990  
WIB browser, 990  
Wideband code division multiple access, 798  
WIM, 854, 990  
Windows for Smart Cards, 990  
Wiping contacts, 739  
Wire bonding, 52, 54, 594  
Wireless identification module, 854  
WML, 990  
Workaround, 990  
Working EF, 423  
Working elementary file, 425  
WORM, 439  
    memory management, 468  
Wound coil, 591  
WRITE BINARY, 359  
Write commands, 358  
WRITE DATA, 392, 393  
WTLS, 854  
WWW addresses, 1018  
X-rays, 640  
X.25, 755  
X.509, 178, 900, 901, 990  
XML, 110, 990  
XOR checksum, 124  
Zentraler Kreditausschuss, 783  
ZKA, 783, 990  
    criteria, 663, 664