EMV®
Integrated Circuit Card Specifications for Payment Systems

Book 2

Security and Key Management

Version 4.4
October 2022

The EMV® Specifications are provided “AS IS” without warranties of any kind, and EMVCo neither assumes nor accepts any liability for any errors or omissions contained in these Specifications. EMVCO DISCLAIMS ALL REPRESENTATIONS AND WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AS TO THESE SPECIFICATIONS.

EMVCo makes no representations or warranties with respect to intellectual property rights of any third parties in or in relation to the Specifications. EMVCo undertakes no responsibility to determine whether any implementation of these Specifications may violate, infringe, or otherwise exercise the patent, copyright, trademark, trade secret, know-how, or other intellectual property rights of third parties, and thus any person who implements any part of these Specifications should consult an intellectual property attorney before any such implementation.

Without limiting the foregoing, the Specifications may provide for the use of public key encryption and other technology, which may be the subject matter of patents in several countries. Any party seeking to implement these Specifications is solely responsible for determining whether its activities require a license to any such technology, including for patents on public key encryption technology. EMVCo shall not be liable under any theory for any party’s infringement of any intellectual property rights in connection with these Specifications.

Revision Log – Version 4.4

The following changes have been made to Book 2 since the publication of Version 4.3. Numbering and cross references in this version have been updated to reflect changes introduced by the published bulletins.

Incorporated changes described in the following Specification Bulletins:

Specification Bulletin no. 104: Issuer guidance on TVR bit setting for CDA

Specification Bulletin no. 137: CDA

Specification Bulletin no. 144: Terminal Unpredictable Number generation

Specification Bulletin no. 162: AES Key Derivation Erratum

Specification Bulletin no. 163: Changes to PIN Pad requirements

Specification Bulletin no. 178, Third Edition: Tokenisation Data Objects – Payment Account Reference (PAR)

Specification Bulletin no. 185, Second Edition: Biometric Terminal Specification

Specification Bulletin no. 208: Clarification of Maximum Public Key Lengths

Specification Bulletin no. 231: Issuer Identification Number Extended (IINE)

Specification Bulletin no. 243: Introduction of XDA/ODE for EMV Specifications

Minor editorial clarifications and corrections, including those described in the following:

Specification Bulletin no. 243: Introduction of XDA/ODE for EMV Specifications

Contents

Part I – General

1 Scope 12
1.1 Changes in Version 4.4 12
1.2 Structure 12
1.3 Underlying Standards 13
1.4 Audience 13
2 Normative References 14
3 Definitions 17
4 Abbreviations, Notations, Conventions, and Terminology 25
4.1 Abbreviations 25
4.2 Notations 32
4.3 Data Element Format Conventions 34
4.4 Terminology 35

Part II – Security and Key Management Techniques

5 Static Data Authentication (SDA) 37
5.1 Keys and Certificates 39
5.1.1 Static Data to be Authenticated 42
5.1.2 Certification Revocation List 43
5.2 Retrieval of Certification Authority Public Key 44
5.3 Retrieval of Issuer Public Key 44
5.4 Verification of Signed Static Application Data 47
6 Offline Dynamic Data Authentication (DDA, CDA) 49
6.1 Keys and Certificates 53
6.1.1 Static Data To Be Authenticated 56
6.1.2 Certification Revocation List 56
6.2 Retrieval of Certification Authority Public Key 57
6.3 Retrieval of Issuer Public Key 57
6.4 Retrieval of ICC Public Key 59
6.5 Dynamic Data Authentication (DDA) 61
6.5.1 Dynamic Signature Generation 61
6.5.2 Dynamic Signature Verification 63
6.6 Combined DDA/Application Cryptogram Generation (CDA) 65

6.6.1 Dynamic Signature Generation 66
6.6.2 Dynamic Signature Verification 69
6.6.3 Sample CDA Flow 72
7 PIN and Biometric Data Encipherment Using RSA 76
7.1 Keys and Certificates 77
7.2 PIN Encipherment and Verification 80
7.3 Biometric Data Encipherment and Recovery 82
8 Application Cryptogram and Issuer Authentication 86
8.1 Application Cryptogram Generation 86
8.1.1 Data Selection 86
8.1.2 Application Cryptogram Algorithm 87
8.2 Issuer Authentication 88
8.2.1 ARPC Method 1 88
8.2.2 ARPC Method 2 89
8.3 Key Management 89
9 Secure Messaging 90
9.1 Secure Messaging Format 90
9.2 Secure Messaging for Integrity and Authentication 90
9.2.1 Command Data Field 90
9.2.2 MAC Session Key Derivation 92
9.2.3 MAC Computation 92
9.3 Secure Messaging for Confidentiality 94
9.3.1 Command Data Field 94
9.3.2 Encipherment Session Key Derivation 94
9.3.3 Encipherment/Decipherment 95
9.4 Key Management 95
10 CA Public Key Management Principles and Policies 96
11 Terminal Security and Key Management Requirements 97
11.1 Security Requirements for PIN Pads 97
11.2 Key Management Requirements 97
11.2.1 Certification Authority Public Key Introduction 97
11.2.2 Certification Authority Public Key Storage 100
11.2.3 Certification Authority Public Key Withdrawal 102
11.3 Unpredictable Number Generation 104
12 Offline Dynamic Data Authentication Using ECC (XDA) 106
12.1 Keys and Certificates 108
12.1.1 Certification Revocation List 109
12.2 Retrieval of Certification Authority Public Key 110
12.3 Authentication and Recovery of Issuer Public Key 111

12.4 Authentication and Recovery of ICC Public Key 114
12.5 XDA Signature Generation and Verification 118
12.5.1 Requesting, Generating, and Verifying an XDA Signature 118
12.5.2 Dynamic Signature Generation 118
12.5.3 Dynamic Signature Verification 121
12.5.4 Sample XDA Flow 121
13 Offline Data Encipherment Using ECC 124
13.1 Keys and Certificates 124
13.2 PIN Encipherment 125
13.3 PIN Decipherment and Verification 127
13.4 Biometric Data Encipherment 128
13.5 Biometric Data Decipherment and Recovery 131

Part III – Annexes

Annex A Security Mechanisms 133
A1 Symmetric Mechanisms 133
A1.1 Encipherment 133
A1.2 Message Authentication Code 134
A1.3 Session Key Derivation 137
A1.4 Master Key Derivation 138
A2 Asymmetric Mechanisms 141
A2.1 Digital Signature Scheme Using RSA 141
A2.2 Digital Signature Scheme Using ECC 143
A2.3 Encryption Scheme Using ECC 145
Annex B Approved Cryptographic Algorithms 151
B1 Symmetric Algorithms 151
B1.1 Data Encryption Standard (DES) 8-byte block cipher 151
B1.2 Advanced Encryption Standard (AES) 16-byte block cipher 151
B2 Asymmetric Algorithms 152
B2.1 RSA Algorithm 152
B2.2 Elliptic Curve Cryptography (ECC) 154
B2.3 Hash Algorithms (for ECC) 161
B2.4 Cryptographic Algorithm Suites (for ECC) 163
B2.5 Random Number Generation (for ECC) 164
B2.6 Integer Conversion Functions (for ECC) 165
B3 Hashing Algorithms (for RSA) 168
B3.1 Secure Hash Algorithm (SHA-1) 168
Annex C Informative References 169

Annex D Implementation Considerations 170
D1 Issuer and ICC RSA Public Key Length Considerations 170
D1.1 Issuer Public Key Restriction 170
D1.2 ICC Public Key Restriction 171
D2 Format 1 Secure Messaging Illustration 173
D2.1 Securing the Command APDU 173
D2.2 Encipherment 176
D2.3 MAC Computation 176
D3 Application Transaction Counter Considerations 178
D4 CDA Modes 179

Part IV – Common Core Definitions

Common Core Definitions 183
Changed Sections 183
6 Offline Dynamic Data Authentication 183
6.5 Dynamic Data Authentication (DDA) 183
6.6 Combined DDA/Application Cryptogram Generation (CDA) 184
8 Application Cryptogram and Issuer Authentication 185
8.1 Application Cryptogram Generation 185
8.2 Issuer Authentication 186
8.3 Key Management 186
9 Secure Messaging 187
9.1 Secure Messaging Format 187
9.2 Secure Messaging for Integrity and Authentication 187
9.3 Secure Messaging for Confidentiality 188
9.4 Key Management 188

Index 189

Tables

Table 1: Required ICC Data Elements for SDA 38
Table 2: Issuer Public Key Data To Be Signed by Certification Authority 40
Table 3: Static Application Data To Be Signed by Issuer 41
Table 4: Data Objects Required for SDA 42
Table 5: Minimum Data for Certificate Revocation List Entry 43
Table 6: Format of Data Recovered from Issuer Public Key Certificate 45
Table 7: Format of Data Recovered from Signed Static Application Data 47
Table 8: Required ICC Data Elements for Offline Dynamic Data Authentication 51
Table 9: Data Element Generated for Offline Dynamic Data Authentication 52
Table 10: Issuer Public Key Data To Be Signed by Certification Authority 54
Table 11: ICC Public Key Data To Be Signed by Issuer 55
Table 12: Data Objects Required for Public Key Authentication for Offline Dynamic Data Authentication 56
Table 13: Format of Data Recovered from Issuer Public Key Certificate 58
Table 14: Format of Data Recovered from ICC Public Key Certificate 60
Table 15: Dynamic Application Data To Be Signed 62
Table 16: Additional Data Objects Required for Dynamic Signature Generation and Verification 63
Table 17: Format of Data Recovered from Signed Dynamic Application Data 64
Table 18: Dynamic Application Data To Be Signed 68
Table 19: 32-38 Leftmost Bytes of ICC Dynamic Data 68
Table 20: Data Objects Included in Response to GENERATE AC for TC or ARQC 69
Table 21: Data Objects Included in Response to GENERATE AC for AAC 69
Table 22: Format of Data Recovered from Signed Dynamic Application Data 70
Table 23: ICC PIN Encipherment Public Key Data To Be Signed by Issuer 78
Table 24: Data Objects Required for Retrieval of ICC PIN Encipherment Public Key 79
Table 25: Data To Be Enciphered for PIN Encipherment 80
Table 26: Data To Be Enciphered for Biometric Encipherment 83
Table 27: Biometric Verification Data Template 84
Table 28: Recommended Minimum Set of Data Elements for Application Cryptogram Generation 87
Table 29: ECC Self-Signed Certification Authority Public Key & Related Data (Binary Encoding) 99
Table 30: Minimum Set of Certification Authority Public Key Related Data Elements To Be Stored in Terminal (RSA) 101
Table 31: Minimum Set of Certification Authority Public Key Related Data Elements To Be Stored in Terminal (ECC) 102
Table 32: Data Elements Used to Generate Terminal Unpredictable Number (UN) 104
Table 33: Data Element Generated for XDA 107
Table 34: ICC Data Required for Public Key Authentication for XDA and ECC ODE 109
Table 35: Issuer Public Key Certificate Tag '90' (ECC) 112
Table 36: ICC Public Key Certificate Tag '9F46' (XDA) and Tag '9F2D' (ODE) 115
Table 37: Dynamic Application Data To Be Signed (XDA) 119
Table 38: Format of XDA Signed Dynamic Application Data 120

Table 39: Data Objects Included in Response to GENERATE AC with XDA Signature 120
Table 40: Data To Be Enciphered for PIN Encipherment (ECC) 126
Table 41: Data To Be Enciphered for Biometric Encipherment 129
Table 42: Biometric Verification Data Template 130
Table 43: Upper Bounds for Size of Moduli 152
Table 44: Selected Elliptic Curves 158
Table 45: Parameters of Curve P-256 158
Table 46: Parameters of Curve P-521 159
Table 47: Recognised Hash Algorithms 161
Table 48: Cryptographic Algorithm Suite Indicators for ECC Signatures 163
Table 49: Cryptographic Algorithm Suite Indicators for Offline Data Encipherment (e.g. for PIN or Biometrics) 164
Table 50: Data Lengths in GENERATE AC Response 171
Table 51: CDA Modes 179

Table CCD 1: Data Objects in Response to GENERATE AC for TC or ARQC 184
Table CCD 2: Data Objects in Response to GENERATE AC for AAC 184
Table CCD 3: Data Elements for Application Cryptogram Generation 185

Figures

Figure 1: Diagram of SDA 37
Figure 2: Diagram of Offline Dynamic Data Authentication 50
Figure 3: CDA Sample Flow Part 1 of 3 73
Figure 4: CDA Sample Flow Part 2 of 3 74
Figure 5: CDA Sample Flow Part 3 of 3 75
Figure 6: Format 1 Command Data Field for Secure Messaging for Integrity and Authentication 91
Figure 7: Format 2 Command Data Field for Secure Messaging for Integrity and Authentication 91
Figure 8: Format 1 – Data Object for Confidentiality 94
Figure 9: Format 2 Command Data Field for Secure Messaging for Confidentiality 94
Figure 10: ECC Self-signed Certification Authority Public Key Diagram 98
Figure 11: Diagram of Offline Dynamic Data Authentication 106
Figure 12: XDA Sample Flow Part 1 of 2 122
Figure 13: XDA Sample Flow Part 2 of 2 123
Figure 14: Decimalisation for Master Key Derivation 139

Part I

General

1 Scope

This document, the *Integrated Circuit Card (ICC) Specifications for Payment Systems – Book 2, Security and Key Management*, describes the minimum security functionality required of integrated circuit cards (ICCs) and terminals to ensure correct operation and interoperability. Additional requirements and recommendations are provided with respect to the on-line communication between ICC and issuer and the management of cryptographic keys at terminal, issuer, and payment system level.

The *Integrated Circuit Card Specifications for Payment Systems* includes the following additional documents, all available on http://www.emvco.com:

- Book 1 – Application Independent ICC to Terminal Interface Requirements
- Book 3 – Application Specification
- Book 4 – Cardholder, Attendant, and Acquirer Interface Requirements

EMVCo also publishes security guidelines (see informative references 3 and 5).

1.1 Changes in Version 4.4

This release incorporates all relevant Specification Bulletins, Application Notes, amendments, etc., published up to the date of this release.

The Revision Log at the beginning of the Book provides additional detail about changes to this specification.

1.2 Structure

Book 2 consists of the following parts:

Part I – **General**
Part II – **Security and Key Management Techniques**
Part III – **Annexes**
Part IV – **Common Core Definitions**

Part I includes this introduction, as well as information applicable to all Books: normative references, definitions, abbreviations, notations, data element format convention, and terminology.

Part II covers:

- Offline static data authentication (SDA)
- Offline dynamic data authentication (DDA and CDA and XDA)
- Offline PIN encipherment
- Application cryptogram generation and issuer authentication
- Secure messaging
- Public key management principles and policies
- Terminal security and key management requirements

Part III (Annexes A-D) specifies the security mechanisms and the approved cryptographic algorithms required to implement the security functions specified, provides a list of informative references, and discusses implementation considerations.

Part IV defines an optional extension to be used when implementing the Common Core Definitions (CCD).

The Book also includes a revision log and an index.

1.3 Underlying Standards

This specification is based on the ISO/IEC 7816 series of standards and should be read in conjunction with those standards. However, if any of the provisions or definitions in this specification differ from those standards, the provisions herein shall take precedence.

1.4 Audience

This specification is intended for use by manufacturers of ICCs and terminals, system designers in payment systems, and financial institution staff responsible for implementing financial applications in ICCs.

2 Normative References

The following specifications and standards contain provisions that are referenced in these specifications. The latest version shall apply unless a publication date is explicitly stated.

EMV Contact Interface Specification	EMV Level 1 Specifications for Payment Systems, EMV Contact Interface Specification
EMV Tokenisation Framework	EMV Payment Tokenisation Specification – Technical Framework Framework specification for an interoperable Payment Tokenisation solution.
FIPS 202	SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions
IEEE P1363	Standard Specifications For Public-Key Cryptography
ISO 639-1	Codes for the representation of names of languages – Part 1: Alpha-2 Code **Note:** This standard is updated continuously by ISO. Additions/changes to ISO 639-1:1988: Codes for the Representation of Names of Languages are available on: http://www.loc.gov/standards/iso639-2/php/code_changes.php
ISO 3166	Codes for the representation of names of countries and their subdivisions
ISO 4217	Codes for the representation of currencies and funds
ISO/IEC 7812-1	Identification cards – Identification of issuers — Part 1: Numbering System
ISO/IEC 7813	Identification cards – Financial transaction cards
ISO/IEC 7816-4	Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange
ISO/IEC 7816-5	Identification cards — Integrated circuit cards — Part 5: Registration of application providers
ISO/IEC 7816-6	Identification cards – Integrated circuit cards – Part 6: Interindustry data elements for interchange

ISO/IEC 7816-11	Identification cards – Integrated circuit cards – Personal verification through biometric methods
ISO 8583:1987	Bank card originated messages – Interchange message specifications – Content for financial transactions
ISO 8583:1993	Financial transaction card originated messages – Interchange message specifications
ISO/IEC 8825-1	Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
ISO/IEC 8859	Information processing – 8-bit single-byte coded graphic character sets
ISO 9362	Banking – Banking telecommunication messages – Bank identifier codes
ISO 9564-1	Financial services – Personal Identification Number (PIN) management and security – Part 1: Basic principles and requirements for PINs in card-based systems
ISO/IEC 9796-2	Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Integer factorization based mechanisms
ISO/IEC 9797-1	Information technology – Security techniques – Message Authentication Codes – Part 1: Mechanisms using a block cipher
ISO/IEC 9797-2	Information technology – Security techniques – Message Authentication Codes (MACs) – Part 2: Mechanisms using a dedicated hash-function
ISO/IEC 10116	Information technology – Security techniques – Modes of operation for an n-bit block cipher
ISO/IEC 10118-3	Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions
ISO/IEC 11770-6	Information technology – Security techniques – Key management — Part 6: Key derivation
ISO 13616	Banking and related financial services – International bank account number (IBAN)

ISO/IEC 14888-3	Information technology – Security techniques – Digital signatures with appendix — Part 3: Discrete logarithm based mechanisms
ISO/IEC 15946-1	Information technology – Security techniques – Cryptographic techniques based on elliptic curves — Part 1: General
ISO/IEC 15946-5	Information technology – Security techniques – Cryptographic techniques based on elliptic curves — Part 5: Elliptic curve generation
ISO 16609	Banking – Requirements for message authentication using symmetric techniques
ISO/IEC 18031	Information technology – Security techniques – Random bit generation
ISO/IEC 18033-2	Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers
ISO/IEC 18033-3	Information technology – Security techniques – Encryption algorithms – Part 3: Block ciphers
ISO/IEC 19772	Information technology – Security techniques – Authenticated encryption
ISO/IEC 19785-3	Information technology – Common Biometric Exchange Formats Framework – Patron format specifications
ISO/IEC 19794	Information technology – Biometric data interchange formats
ISO/IEC 19794-2	Information technology – Biometric data interchange formats – Part 2: Finger minutiae data
SEC 1	Elliptic Curve Cryptography (available at http://www.secg.org)

3 Definitions

The following terms are used in one or more books of these specifications.

Application	The application protocol between the card and the terminal and its related set of data.
Application Authentication Cryptogram	An Application Cryptogram generated by the card when declining a transaction.
Application Cryptogram	A cryptogram generated by the card in response to a GENERATE AC command. See also: • Application Authentication Cryptogram • Authorisation Request Cryptogram • Transaction Certificate
Authentication	The provision of assurance of the claimed identity of an entity or of data origin.
Authorisation Request Cryptogram	An Application Cryptogram generated by the card when requesting online authorisation.
Authorisation Response Cryptogram	A cryptogram generated by the issuer in response to an Authorisation Request Cryptogram.
Biometric Data Block	A block of data with a specific format that contains information captured from a biometric capture device and that could be used as follows: • stored in the card as part of the biometric reference template • sent to the ICC in the data field of the PIN CHANGE/UNBLOCK command • sent to the ICC in the data field of the VERIFY command for offline biometric verification • sent online for verification The format of the BDB is outside the scope of this specification.

Biometric Reference Template	Biometric data stored in the card as reference. Data provided by a biometric capture device would be compared against the biometric reference template to determine a match.
Biometric Verification	The process of determining that the biometrics presented, such as finger, palm, iris, voice, or facial, are valid.
Byte	8 bits.
Card	A payment card as defined by a payment system.
Certificate	The public key and identity of an entity together with some other information, rendered unforgeable by signing with the private key of the certification authority which issued that certificate.
Certification Authority	Trusted third party that establishes a proof that links a public key and other relevant information to its owner.
Ciphertext	Enciphered information.
Combined DDA/Application Cryptogram Generation	A form of offline dynamic data authentication.
Command	A message sent by the terminal to the ICC that initiates an action and solicits a response from the ICC.
Command Chaining	A mechanism where consecutive command-response pairs can be chained.
Compromise	The breaching of secrecy or security.
Concatenation	Two elements are concatenated by appending the bytes from the second element to the end of the first. Bytes from each element are represented in the resulting string in the same sequence in which they were presented to the terminal by the ICC, that is, most significant byte first. Within each byte bits are ordered from most significant bit to least significant. A list of elements or objects may be concatenated by concatenating the first pair to form a new element, using that as the first element to concatenate with the next in the list, and so on.

Contact	A conducting element ensuring galvanic continuity between integrated circuit(s) and external interfacing equipment.
Cryptogram	Result of a cryptographic operation.
Cryptographic Algorithm	An algorithm that transforms data in order to hide or reveal its information content.
Data Integrity	The property that data has not been altered or destroyed in an unauthorised manner.
Decipherment	The reversal of a corresponding encipherment.
DEM1	A family of data encapsulation mechanisms defined in ISO/IEC 18033-2.
Digital Signature	An asymmetric cryptographic transformation of data that allows the recipient of the data to prove the origin and integrity of the data, and protect the sender and the recipient of the data against forgery by third parties, and the sender against forgery by the recipient.
Dynamic Data Authentication	A form of offline dynamic data authentication
Elliptic Curve Cryptography	Public key cryptography based on the algebraic structure of elliptic curves over finite fields.
Encipherment	The reversible transformation of data by a cryptographic algorithm to produce ciphertext.
Exclusive-OR	Binary addition with no carry, giving the following values: 0 + 0 = 0 0 + 1 = 1 1 + 0 = 1 1 + 1 = 0
Extended Data Authentication	A form of offline dynamic data authentication.
Facial Verification	The process of determining that the face presented is valid.
Financial Transaction	The act between a cardholder and a merchant or acquirer that results in the exchange of goods or services against payment.

Finger Verification	The process of determining that the finger presented is valid.
Function	A process accomplished by one or more commands and resultant actions that are used to perform all or part of a transaction.
Hash Function	A function that maps strings of bits to fixed-length strings of bits, satisfying the following two properties: • It is computationally infeasible to find for a given output an input which maps to this output. • It is computationally infeasible to find for a given input a second input that maps to the same output. Additionally, if the hash function is required to be collision-resistant, it must also satisfy the following property: • It is computationally infeasible to find any two distinct inputs that map to the same output.
Hash Result	The string of bits that is the output of a hash function.
I2OSP	An integer to octet string conversion primitive function defined in ISO/IEC 18033-2.
Integrated Circuit(s)	Electronic component(s) designed to perform processing and/or memory functions.
Integrated Circuit(s) Card	A card into which one or more integrated circuits are inserted to perform processing and memory functions.
Interface Device	That part of a terminal into which the ICC is inserted, including such mechanical and electrical devices as may be considered part of it.
Iris Verification	The process of determining that the iris presented is valid.
Issuer Action Code	Any of the following, which reflect the issuer-selected action to be taken upon analysis of the TVR: • Issuer Action Code – Default • Issuer Action Code – Denial • Issuer Action Code – Online

Kernel	The set of functions required to be present on every terminal implementing a specific interpreter. The kernel contains device drivers, interface routines, security and control functions, and the software for translating from the virtual machine language to the language used by the real machine. In other words, the kernel is the implementation of the virtual machine on a specific real machine.
Key	A sequence of symbols that controls the operation of a cryptographic transformation.
Key Introduction	The process of generating, distributing, and beginning use of a key pair.
Key Withdrawal	The process of removing a key from service as part of its revocation.
Keypad	Arrangement of numeric, command, and, where required, function and/or alphanumeric keys laid out in a specific manner.
Library	A set of high-level software functions with a published interface, providing general support for terminal programs and/or applications.
Logical Compromise	The compromise of a key through application of improved cryptanalytic techniques, increases in computing power, or combination of the two.
Magnetic Stripe	The stripe containing magnetically encoded information.
Message	A string of bytes sent by the terminal to the card or vice versa, excluding transmission-control characters.
Message Authentication Code	A symmetric cryptographic transformation of data that protects the sender and the recipient of the data against forgery by third parties.
Nibble	The four most significant or least significant bits of a byte.
Offline Data Encipherment	Offline encipherment of data, in particular for cardholder PIN and biometric data.
Padding	Appending extra bits to either side of a data string.
Palm Verification	The process of determining that the palm presented is valid.

Path	Concatenation of file identifiers without delimitation.
Payment System Environment	A logical construct within the ICC, the entry point to which is a Directory Definition File (DDF) named '1PAY.SYS.DDF01'. This DDF contains a Payment System Directory which in turn contains entries for one or more Application Definition Files (ADFs) which are formatted according to this specification.
Physical Compromise	The compromise of a key resulting from the fact that it has not been securely guarded, or a hardware security module has been stolen or accessed by unauthorised persons.
PIN Pad	Arrangement of numeric and command keys to be used for personal identification number (PIN) entry. Also known as a “PIN Entry Device” (PED).
Plaintext	Unenciphered information.
Potential Compromise	A condition where cryptanalytic techniques and/or computing power has advanced to the point that compromise of a key of a certain length is feasible or even likely.
Private Key	That key of an entity’s asymmetric key pair that should only be used by that entity. In the case of a digital signature scheme, the private key defines the signature function.
Public Key	That key of an entity’s asymmetric key pair that can be made public. In the case of a digital signature scheme, the public key defines the verification function.
Public Key Certificate	The public key information of an entity signed by the certification authority and thereby rendered unforgeable.
Response	A message returned by the ICC to the terminal after the processing of a command message received by the ICC.
RSA-KEM	A family of key encapsulation mechanisms defined in ISO/IEC 18033-2.
RSATransform	The RSA exponentiation that is used for encryption and decryption, and generating and verifying a signature.

Script	A command or a string of commands transmitted by the issuer to the terminal for the purpose of being sent serially to the ICC as commands.
Secret Key	A key used with symmetric cryptographic techniques and usable only by a set of specified entities.
Socket	An execution vector defined at a particular point in an application and assigned a unique number for reference.
Static Data Authentication	Offline static data authentication
Symmetric Cryptographic Technique	A cryptographic technique that uses the same secret key for both the originator's and recipient's transformation. Without knowledge of the secret key, it is computationally infeasible to compute either the originator's or the recipient's transformation.
Template	Value field of a constructed data object, defined to give a logical grouping of data objects.
Terminal	The device used in conjunction with the ICC at the point of transaction to perform a financial transaction. The terminal incorporates the interface device and may also include other components and interfaces such as host communications.
Terminal Action Code	Any of the following, which reflect the acquirer-selected action to be taken upon analysis of the TVR: • Terminal Action Code – Default • Terminal Action Code – Denial • Terminal Action Code – Online
Terminate Card Session	End the card session by deactivating the IFD contacts according to EMV Contact Interface Specification and displaying a message indicating that the ICC cannot be used to complete the transaction.
Terminate Transaction	Stop the current application and deactivate the card.
Transaction	An action taken by a terminal at the user's request. For a POS terminal, a transaction might be payment for goods, etc. A transaction selects among one or more applications as part of its processing flow.

Transaction Certificate	An Application Cryptogram generated by the card when accepting a transaction.
Virtual Machine	A theoretical microprocessor architecture that forms the basis for writing application programs in a specific interpreter software implementation.
Voice Verification	The process of determining that the voice presented is valid.

4 Abbreviations, Notations, Conventions, and Terminology

4.1 Abbreviations

a	Alphabetic (see section 4.3, Data Element Format Conventions)
AAC	Application Authentication Cryptogram
AAD	Additional Authenticated Data
AC	Application Cryptogram
ADF	Application Definition File
AEF	Application Elementary File
AES	Advanced Encryption Standard
AFL	Application File Locator
AID	Application Identifier
AIP	Application Interchange Profile
an	Alphanumeric (see section 4.3)
ans	Alphanumeric Special (see section 4.3)
APDU	Application Protocol Data Unit
API	Application Program Interface
ARC	Authorisation Response Code
ARPC	Authorisation Response Cryptogram
ARQC	Authorisation Request Cryptogram
ASI	Application Selection Indicator
ASN	Abstract Syntax Notation
ATC	Application Transaction Counter
ATM	Automated Teller Machine
ATR	Answer to Reset

AUC	Application Usage Control
b	Binary (see section 4.3)
BCD	Binary Coded Decimal
BDB	Biometric Data Block
BEK	Biometric Encryption Key
BER	Basic Encoding Rules (defined in ISO/IEC 8825-1)
BHT	Biometric Header Template
BIC	Bank Identifier Code
BIT	Biometric Information Template
BMK	Biometric MAC Key
CA	Certification Authority
CAD	Card Accepting Device
C-APDU	Command APDU
CBC	Cipher Block Chaining
CBEFF	Common Biometric Exchange Formats Framework
CCD	Common Core Definitions
CCI	Common Core Identifier
CCYYMMDD	Year (4 digits), Month, Day
CDA	Combined DDA/Application Cryptogram Generation
CDOL	Card Risk Management Data Object List
CID	Cryptogram Information Data
CLA	Class Byte of the Command Message
cn	Compressed Numeric (see section 4.3)
CPU	Central Processing Unit
CRL	Certificate Revocation List
CSU	Card Status Update
CV	Cryptogram Version
CV Rule	Cardholder Verification Rule

CVM	Cardholder Verification Method
CVR	Card Verification Results
DDA	Dynamic Data Authentication
DDF	Directory Definition File
DDOL	Dynamic Data Authentication Data Object List
DES	Data Encryption Standard
DF	Dedicated File
DIR	Directory
DOL	Data Object List
ECB	Electronic Code Book
EC-SDSA	Elliptic Curve Schnorr Digital Signature Algorithm
ECC	Elliptic Curve Cryptography
EF	Elementary File
EN	European Norm
FC	Format Code
FCI	File Control Information
Hex	Hexadecimal
HHMMSS	Hours, Minutes, Seconds
HMAC	Keyed-hash Message Authentication Code
I/O	Input/Output
IAC	Issuer Action Code (Denial, Default, Online)
IAD	Issuer Application Data
IBAN	International Bank Account Number
IC	Integrated Circuit
ICC	Integrated Circuit(s) Card
ICCD	Issuer Certified Card Data
IEC	International Electrotechnical Commission
IFD	Interface Device

IIN	Issuer Identification Number
IINE	Issuer Identification Number Extended
INS	Instruction Byte of Command Message
ISO	International Organization for Standardization
KD	Key Derivation
KDF	Key Derivation Function
K_M	Master Key
K_S	Session Key
L	Length
l.s.	Least Significant
Lc	Exact Length of Data Sent by the TAL in a Case 3 or 4 Command
LCOL	Lower Consecutive Offline Limit
L_{DD}	Length of the ICC Dynamic Data
Le	Maximum Length of Data Expected by the TAL in Response to a Case 2 or 4 Command
Lr	Length of Response Data Field
LRC	Longitudinal Redundancy Check
M	Mandatory
m.s.	Most Significant
MAC	Message Authentication Code
max.	Maximum
MF	Master File
MK	ICC Master Key for session key generation
MMDD	Month, Day
MMYY	Month, Year
n	Numeric (see section 4.3)
N_{CA}	Length of the Certification Authority Public Key Modulus
NF	Norme Française

N_{FIELD}	Length of a finite field element
N_{HASH}	Output length of a hash function
N_I	Length of the Issuer Public Key Modulus
N_{IC}	Length of the ICC Public Key Modulus
NIST	National Institute for Standards and Technology
N_{PE}	Length of the ICC PIN Encipherment Public Key Modulus
N_{SIG}	Length of an ECC Digital Signature
O	Optional
O/S	Operating System
ODA	Offline Data Authentication
ODE	Offline Data Encipherment
P1	Parameter 1
P2	Parameter 2
PAN	Primary Account Number
PAR	Payment Account Reference
PC	Personal Computer
P_{CA}	Certification Authority Public Key
PDOL	Processing Options Data Object List
P_I	Issuer Public Key
P_{IC}	ICC Public Key
PIN	Personal Identification Number
PIX	Proprietary Application Identifier Extension
POS	Point of Service
pos.	Position
PSE	Payment System Environment
R-APDU	Response APDU
RFU	Reserved for Future Use
RID	Registered Application Provider Identifier

RSA	Rivest, Shamir, Adleman Algorithm
S_{CA}	Certification Authority Private Key
SDA	Static Data Authentication
SDAD	Signed Dynamic Application Data
SFI	Short File Identifier
SHA-1	Secure Hash Algorithm 1
SHA-2	Secure Hash Algorithm 2 (includes SHA-256 and SHA-512)
SHA-256	Secure Hash Algorithm 256
SHA-3	Secure Hash Algorithm 3
S_I	Issuer Private Key
S_{IC}	ICC Private Key
SK	Session Key
SW1	Status Byte One
SW2	Status Byte Two
TAA	Terminal Action Analysis
TAC	Terminal Action Code(s) (Default, Denial, Online)
TAL	Terminal Application Layer
TC	Transaction Certificate
TDOL	Transaction Certificate Data Object List
TLV	Tag Length Value
TPDU	Transport Protocol Data Unit
TSI	Transaction Status Information
TVR	Terminal Verification Results
UCOL	Upper Consecutive Offline Limit
UL	Underwriters Laboratories Incorporated
UN	Unpredictable Number
var.	Variable (see section 4.3)
XDA	Extended Data Authentication

YYMM	Year, Month
YYMMDD	Year, Month, Day

4.2 Notations

'0' to '9' and 'A' to 'F'	16 hexadecimal characters
xx	Any value
A := B	A is assigned the value of B
A = B	Value of A is equal to the value of B
A ≡ B mod n	Integers A and B are congruent modulo the integer n, that is, there exists an integer d such that $(A - B) = dn$
A mod n	The reduction of the integer A modulo the integer n, that is, the unique integer r, $0 \leq r < n$, for which there exists an integer d such that $A = dn + r$
A / n	The integer division of A by n, that is, the unique integer d for which there exists an integer r, $0 \leq r < n$, such that $A = dn + r$
Y := ALG(K)[X]	Encipherment of a data block X with a block cipher as specified in section A1, using a secret key K
$X = ALG^{-1}(K)[Y]$	Decipherment of a data block Y with a block cipher as specified in section A1, using a secret key K
$Y := Sign\ (S_K)[X]$	The signing of a data block X with an asymmetric reversible algorithm as specified in section A2, using the private key S_K
$X = Recover(P_K)[Y]$	The recovery of the data block X with an asymmetric reversible algorithm as specified in section A2, using the public key P_K
C := (A || B)	The concatenation of an n-bit number A and an m-bit number B, which is defined as $C = 2^m A + B$.
Leftmost	Applies to a sequence of bits, bytes, or digits and used interchangeably with the term "most significant". If C = (A || B) as above, then A is the leftmost n bits of C.
Rightmost	Applies to a sequence of bits, bytes, or digits and used interchangeably with the term "least significant". If C = (A || B) as above, then B is the rightmost m bits of C.

H := Hash[MSG]	Hashing of a message MSG of arbitrary length using a 160-bit hash function
X ⊕ Y	The symbol '⊕' denotes bit-wise exclusive-OR and is defined as follows: X ⊕ Y — The bit-wise exclusive-OR of the data blocks X and Y. If one data block is shorter than the other, then it is first padded to the left with sufficient binary zeros to make it the same length as the other.
MIN (x, y)	The smaller of values x and y.

4.3 Data Element Format Conventions

The EMV specifications use the following data element formats:

a	Alphabetic data elements contain a single character per byte. The permitted characters are alphabetic only (a to z and A to Z, upper and lower case).
an	Alphanumeric data elements contain a single character per byte. The permitted characters are alphabetic (a to z and A to Z, upper and lower case) and numeric (0 to 9). There is one exception: The permitted characters for Payment Account Reference are alphabetic *upper case* (A to Z) and numeric (0 to 9).
ans	Alphanumeric Special data elements contain a single character per byte. The permitted characters and their coding are shown in the Common Character Set table in Book 4 Annex B. There is one exception: The permitted characters for Application Preferred Name are the non-control characters defined in the ISO/IEC 8859 part designated in the Issuer Code Table Index associated with the Application Preferred Name.
b	These data elements consist of either unsigned binary numbers or bit combinations that are defined elsewhere in the specification. Binary example: The Application Transaction Counter (ATC) is defined as “b” with a length of two bytes. An ATC value of 19 is stored as Hex '00 13'. Bit combination example: Processing Options Data Object List (PDOL) is defined as “b” with the format shown in Book 3 section 5.4.
cn	Compressed numeric data elements consist of two numeric digits (having values in the range Hex '0'–'9') per byte. These data elements are left justified and padded with trailing hexadecimal 'F's. Example: The Application Primary Account Number (PAN) is defined as “cn” with a length of up to ten bytes. A value of 1234567890123 may be stored in the Application PAN as Hex '12 34 56 78 90 12 3F FF' with a length of 8.
n	Numeric data elements consist of two numeric digits (having values in the range Hex '0' – '9') per byte. These digits are right justified and padded with leading hexadecimal zeroes. Other specifications sometimes refer to this data format as Binary Coded Decimal (“BCD”) or unsigned packed. Example: Amount, Authorised (Numeric) is defined as “n 12” with a length of six bytes. A value of 12345 is stored in Amount, Authorised (Numeric) as Hex '00 00 00 01 23 45'.
var.	Variable data elements are variable length and may contain any bit combination. Additional information on the formats of specific variable data elements is available elsewhere.

4.4 Terminology

business agreement	An agreement reached between a payment system and its business partner(s).
proprietary	Not defined in this specification and/or outside the scope of this specification
shall	Denotes a mandatory requirement
should	Denotes a recommendation

Part II

Security and Key Management Techniques

5 Static Data Authentication (SDA)

This section describes SDA, an offline data authentication mechanism that utilises RSA public key cryptography.

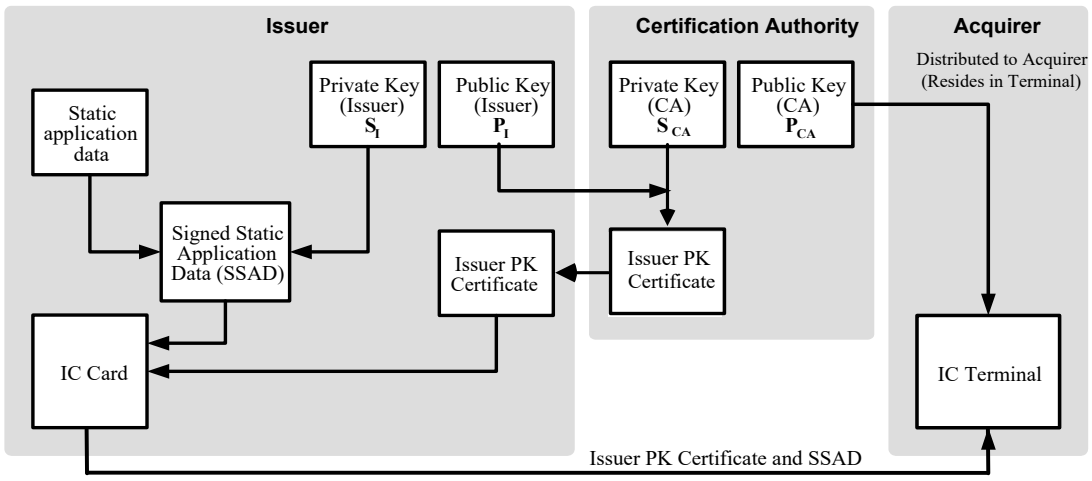
Offline static data authentication is performed by the terminal using a digital signature scheme based on public key techniques to confirm the legitimacy of critical ICC-resident static data. This detects unauthorised alteration of data after personalisation.

The only form of offline static data authentication defined is Static Data Authentication (SDA) that verifies the data identified by the Application File Locator (AFL) and by the optional Static Data Authentication Tag List.

SDA requires the existence of a certification authority, which is a highly secure cryptographic facility that 'signs' the issuer's public keys.

Every terminal conforming to this specification shall contain the appropriate certification authority's public key(s) for every application recognised by the terminal.

This specification permits multiple AIDs to share the same 'set' of Certification Authority Public Keys. The relationship between the data and the cryptographic keys is shown in Figure 1.



Card provides to Terminal:

- Issuer PK Certificate (P_I signed by CA using S_{CA})
- Signed Static Application Data (SSAD) (signed by the Issuer using S_I)

Terminal:

- Uses P_{CA} to verify that the Issuer's P_I was signed by the CA
- Uses P_I to verify that the Card's SSAD was signed by the Issuer

Figure 1: Diagram of SDA

ICCs that support SDA shall contain the data elements listed in Table 1:

Required Data Element	Length	Description
Certification Authority Public Key Index	1	Contains a binary number that indicates which of the application's Certification Authority Public Keys and its associated algorithm that reside in the terminal is to be used with this ICC.
Issuer Public Key Certificate	var.	Provided by the appropriate certification authority to the card issuer. When the terminal verifies this data element, it authenticates the Issuer Public Key plus additional data as described in section 5.3.
Signed Static Application Data	var.	Generated by the issuer using the private key that corresponds to the public key authenticated in the Issuer Public Key Certificate. It is a digital signature covering critical ICC-resident static data elements, as described in section 5.4.
Issuer Public Key Remainder	var.	The presence of this data element in the ICC is conditional. See section 5.1 for further explanation.
Issuer Public Key Exponent	var.	Provided by the issuer. See section 5.1 for further explanation.

Table 1: Required ICC Data Elements for SDA

To support SDA, each terminal shall be able to store six Certification Authority Public Keys per Registered Application Provider Identifier (RID) and shall associate with each such key the key-related information to be used with the key (so that terminals can in the future support multiple algorithms and allow an evolutionary transition from one to another, as discussed in section 11.2.2). The terminal shall be able to locate any such key (and the key-related information) given the RID and Certification Authority Public Key Index as provided by the ICC.

SDA shall use a reversible algorithm as specified in section A2.1 and section B2. Section 5.1 contains an overview of the keys and certificates involved in the SDA process, and sections 5.2 to 5.4 specify the three main steps in the process, namely:

- Retrieval of the Certification Authority Public Key by the terminal
- Retrieval of the Issuer Public Key by the terminal
- Verification of the Signed Static Application Data by the terminal

If SDA fails then the terminal shall set the 'SDA failed' bit in the Terminal Verification Results (TVR) to 1.

5.1 Keys and Certificates

To support SDA, an ICC shall contain the Signed Static Application Data, which is signed with the Issuer Private Key. The Issuer Public Key shall be stored on the ICC with a public key certificate.

The bit length of all moduli shall be a multiple of 8, the leftmost bit of its leftmost byte being 1. All lengths are given in bytes.

The signature scheme specified in section A2.1 is applied to the data specified in Table 2 using the Certification Authority Private Key S_{CA} in order to obtain the Issuer Public Key Certificate.

The public key pair of the certification authority has a public key modulus of N_{CA} bytes, where $N_{CA} \leq 248$. The Certification Authority Public Key Exponent shall be equal to 3 or $2^{16} + 1$.

The signature scheme specified in section A2.1 is applied to the data specified in Table 3 using the Issuer Private Key S_I in order to obtain the Signed Static Application Data.

The public key pair of the issuer has an Issuer Public Key Modulus of N_I bytes, where $N_I \leq N_{CA} \leq 248$. If $N_I > (N_{CA} - 36)$, the Issuer Public Key Modulus is split into two parts, namely:

- the Leftmost Digits of the Issuer Public Key, consisting of the $N_{CA} - 36$ most significant bytes of the modulus, and
- the Issuer Public Key Remainder, consisting of the remaining $N_I - (N_{CA} - 36)$ least significant bytes of the modulus.

The Issuer Public Key Exponent shall be equal to 3 or $2^{16} + 1$.

All the information necessary for SDA is specified in Table 4 and stored in the ICC. With the exception of the RID, which can be obtained from the Application Identifier (AID; see Book 1 section 12.2.1), this information may be retrieved with the READ RECORD command. If any of this data is missing, SDA has failed.

Field Name	Length	Description	Format
Certificate Format	1	Hex value '02'	b
Issuer Identifier	4	Leftmost 3-8 digits from the Primary Account Number (PAN) (padded to the right with Hex 'F's)	cn 8
Certificate Expiration Date	2	MMYY after which this certificate is invalid	n 4
Certificate Serial Number	3	Binary number unique to this certificate assigned by the certification authority	b
Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme [1]	b
Issuer Public Key Algorithm Indicator	1	Identifies the digital signature algorithm to be used with the Issuer Public Key [1]	b
Issuer Public Key Length	1	Identifies the length of the Issuer Public Key Modulus in bytes	b
Issuer Public Key Exponent Length	1	Identifies the length of the Issuer Public Key Exponent in bytes	b
Issuer Public Key or Leftmost Digits of the Issuer Public Key	$N_{CA} - 36$	If $N_I \leq N_{CA} - 36$, consists of the full Issuer Public Key padded to the right with $N_{CA} - 36 - N_I$ bytes of value 'BB' If $N_I > N_{CA} - 36$, consists of the $N_{CA} - 36$ most significant bytes of the Issuer Public Key [2]	b
Issuer Public Key Remainder	0 or $N_I - N_{CA} + 36$	Present only if $N_I > N_{CA} - 36$ and consists of the $N_I - N_{CA} + 36$ least significant bytes of the Issuer Public Key.	b
Issuer Public Key Exponent	1 or 3	Issuer Public Key Exponent equal to 3 or $2^{16} + 1$	b

Table 2: Issuer Public Key Data To Be Signed by Certification Authority (i.e., input to the hash algorithm)

[1] See Annex B for specific values assigned to approved algorithms.

[2] As can be seen in section A2.1, $N_{CA} - 22$ bytes of the data signed are retrieved from the signature. Since the length of the first through the eighth data elements in Table 2 is 14 bytes, there are $N_{CA} - 22 - 14 = N_{CA} - 36$ bytes remaining in the signature to store the Issuer Public Key Modulus.

Field Name	Length	Description	Format
Signed Data Format	1	Hex Value '03'	b
Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme [3]	b
Data Authentication Code	2	Issuer-assigned code	b
Pad Pattern	$N_I - 26$	Pad pattern consisting of $N_I - 26$ bytes of value 'BB' [4]	b
Static Data to be Authenticated	var.	Static data to be authenticated as specified in Book 3 section 10.3 (see also section 5.1.1)	—

Table 3: Static Application Data To Be Signed by Issuer (i.e., input to the hash algorithm)

[3] See Annex B for specific values assigned to approved algorithms.

[4] As can be seen in section A2.1, $N_I - 22$ bytes of the data signed are retrieved from the signature. Since the length of the first through the third data elements in Table 3 is 4 bytes, there are $N_I - 22 - 4 = N_I - 26$ bytes left for the data to be stored in the signature.

5.1.1 Static Data to be Authenticated

Input to the authentication process is formed from the records identified by the AFL, followed by the value of the Application Interchange Profile (AIP), if identified by the optional Static Data Authentication Tag List (tag '9F4A'). If present, the Static Data Authentication Tag List shall only contain the tag '82' identifying the AIP.

Tag	Length	Value	Format
—	5	Registered Application Provider Identifier (RID)	b
'8F'	1	Certification Authority Public Key Index	b
'90'	N_{CA}	Issuer Public Key Certificate	b
'92'	$N_I - N_{CA} + 36$	Issuer Public Key Remainder, if present	b
'9F32'	1 or 3	Issuer Public Key Exponent	b
'93'	N_I	Signed Static Application Data	b
—	var.	Static data to be authenticated as specified in Book 3 section 10.3 (see also section 5.1.1)	—

Table 4: Data Objects Required for SDA

5.1.2 Certification Revocation List

The terminal may support a Certification Revocation List (CRL) that lists the Issuer Public Key Certificates that payment systems have revoked. If, during SDA, a concatenation of the RID and Certification Authority Public Key Index from the card and the Certificate Serial Number recovered from the Issuer Public Key Certificate is on this list, SDA fails as described in section 5.3 step 10.

At a minimum each entry in the CRL shall contain the following data:

Name	Description	Format	Length
Registered Application Provider Identifier (RID)	Identifiers the application provider	b	5
Certification Authority Public Key Index	Identifies the public key in conjunction with the RID	b	1
Certificate Serial Number	Number unique to this certificate assigned by the certification authority	b	3
Additional Data	Optional terminal proprietary data, such as the date the certificate was added to the revocation list	b	var.

Table 5: Minimum Data for Certificate Revocation List Entry

Additional data such as the date the certificate was added to the CRL may be included in the CRL entry.

The terminal shall be able to support at least thirty entries in the CRL for each RID for which the terminal has CA Public Keys.

The terminal shall be able to update the CRL as requested by the acquirer. The payment systems provide these updates to the acquirer. A reliable method of maintaining the CRL is defined by the terminal vendor and the acquirer and should meet the security requirements of the acquirer. It is the responsibility of the payment system to ensure that the number of revoked certificates does not exceed the maximum number of entries that terminals are required to support and the responsibility of the acquirer to ensure that appropriate entries are deleted in order to make way for new entries.

5.2 Retrieval of Certification Authority Public Key

The terminal reads the Certification Authority Public Key Index. Using this index and the RID, the terminal shall identify and retrieve the terminal-stored Certification Authority Public Key Modulus and Exponent and the associated key-related information, and the corresponding algorithm to be used. If the terminal does not have the key stored associated with this index and RID, SDA has failed.

5.3 Retrieval of Issuer Public Key

1. If the Issuer Public Key Certificate has a length different from the length of the Certification Authority Public Key Modulus obtained in the previous section, SDA has failed.
2. In order to obtain the recovered data specified in Table 6, apply the recovery function specified in section A2.1 to the Issuer Public Key Certificate using the Certification Authority Public Key in conjunction with the corresponding algorithm. If the Recovered Data Trailer is not equal to 'BC', SDA has failed.

Field Name	Length	Description	Format
Recovered Data Header	1	Hex Value '6A'	b
Certificate Format	1	Hex Value '02'	b
Issuer Identifier	4	Leftmost 3-8 digits from the PAN (padded to the right with Hex 'F's)	cn 8
Certificate Expiration Date	2	MMYY after which this certificate is invalid	n 4
Certificate Serial Number	3	Binary number unique to this certificate assigned by the certification authority	b
Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme [5]	b
Issuer Public Key Algorithm Indicator	1	Identifies the digital signature algorithm to be used with the Issuer Public Key [5]	b
Issuer Public Key Length	1	Identifies the length of the Issuer Public Key Modulus in bytes	b
Issuer Public Key Exponent Length	1	Identifies the length of the Issuer Public Key Exponent in bytes	b
Issuer Public Key or Leftmost Digits of the Issuer Public Key	N_{CA} –36	If $N_I \leq N_{CA} - 36$, consists of the full Issuer Public Key padded to the right with $N_{CA} - 36 - N_I$ bytes of value 'BB' If $N_I > N_{CA} - 36$, consists of the $N_{CA} - 36$ most significant bytes of the Issuer Public Key [6]	b
Hash Result	20	Hash of the Issuer Public Key and its related information	b
Recovered Data Trailer	1	Hex value 'BC'	b

Table 6: Format of Data Recovered from Issuer Public Key Certificate

3. Check the Recovered Data Header. If it is not '6A', SDA has failed.
4. Check the Certificate Format. If it is not '02', SDA has failed.

[5] See Annex B for specific values assigned to approved algorithms.

[6] As can be seen in section A2.1, $N_{CA} - 22$ bytes of the data signed are retrieved from the signature. Since the length of the second through the ninth data elements in Table 6 is 14 bytes, there are $N_{CA} - 22 - 14 = N_{CA} - 36$ bytes left for the data to be stored in the signature.

5. Concatenate from left to right the second to the tenth data elements in Table 6 (that is, Certificate Format through Issuer Public Key or Leftmost Digits of the Issuer Public Key), followed by the Issuer Public Key Remainder (if present), and finally the Issuer Public Key Exponent.
6. Apply the indicated hash algorithm (derived from the Hash Algorithm Indicator) to the result of the concatenation of the previous step to produce the hash result.
7. Compare the calculated hash result from the previous step with the recovered Hash Result. If they are not the same, SDA has failed.
8. Verify that the Issuer Identifier matches the leftmost 3-8 PAN digits (allowing for the possible padding of the Issuer Identifier with hexadecimal 'F's). If not, SDA has failed.
9. Verify that the last day of the month specified in the Certificate Expiration Date is equal to or later than today's date. If the Certificate Expiration Date is earlier than today's date, the certificate has expired, in which case SDA has failed.
10. Verify that the concatenation of RID, Certification Authority Public Key Index, and Certificate Serial Number is valid. If not, SDA has failed.[7]
11. If the Issuer Public Key Algorithm Indicator is not recognised, SDA has failed.
12. If all the checks above are correct, concatenate the Leftmost Digits of the Issuer Public Key and the Issuer Public Key Remainder (if present) to obtain the Issuer Public Key Modulus, and continue with the next steps for the verification of the Signed Static Application Data.

[7] This step is optional and is to allow the revocation of the Issuer Public Key Certificate against a Certification Revocation List that may be kept by the terminal (see section 5.1.2).

5.4 Verification of Signed Static Application Data

1. If the Signed Static Application Data has a length different from the length of the Issuer Public Key Modulus, SDA has failed.
2. In order to obtain the Recovered Data specified in Table 7, apply the recovery function specified in section A2.1 on the Signed Static Application Data using the Issuer Public Key in conjunction with the corresponding algorithm. If the Recovered Data Trailer is not equal to 'BC', SDA has failed.

Field Name	Length	Description	Format
Recovered Data Header	1	Hex value '6A'	b
Signed Data Format	1	Hex value '03'	b
Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme [8]	b
Data Authentication Code	2	Issuer-assigned code	b
Pad Pattern	$N_I - 26$	Pad pattern consisting of $N_I - 26$ bytes of value 'BB' [9]	b
Hash Result	20	Hash of the Static Application Data to be authenticated	b
Recovered Data Trailer	1	Hex Value 'BC'	b

Table 7: Format of Data Recovered from Signed Static Application Data

3. Check the Recovered Data Header. If it is not '6A', SDA has failed.
4. Check the Signed Data Format. If it is not '03', SDA has failed.
5. Concatenate from left to right the second to the fifth data elements in Table 7 (that is, Signed Data Format through Pad Pattern), followed by the static data to be authenticated as specified in Book 3 section 10.3. If the Static Data Authentication Tag List is present and contains tags other than '82', then SDA has failed.
6. Apply the indicated hash algorithm (derived from the Hash Algorithm Indicator) to the result of the concatenation of the previous step to produce the hash result.
7. Compare the calculated hash result from the previous step with the recovered Hash Result. If they are not the same, SDA has failed.

[8] See Annex B for specific values assigned to approved algorithms.

[9] As can be seen in section A2.1, $N_I - 22$ bytes of the data signed are retrieved from the signature. Since the length of the second through the fourth data elements in Table 7 is 4 bytes, there are $N_I - 22 - 4 = N_I - 26$ bytes left for the data to be stored in the signature.

If all of the above steps were executed successfully, SDA was successful. The Data Authentication Code recovered in Table 7 shall be stored in tag '9F45'.

6 Offline Dynamic Data Authentication (DDA, CDA)

This section describes DDA and CDA, two offline dynamic data authentication mechanisms that utilise RSA public key cryptography.

Offline dynamic data authentication is performed by the terminal using a digital signature scheme based on public key techniques to authenticate the ICC and confirm the legitimacy of critical ICC-resident/generated data and data received from the terminal. This precludes the counterfeiting of any such card.

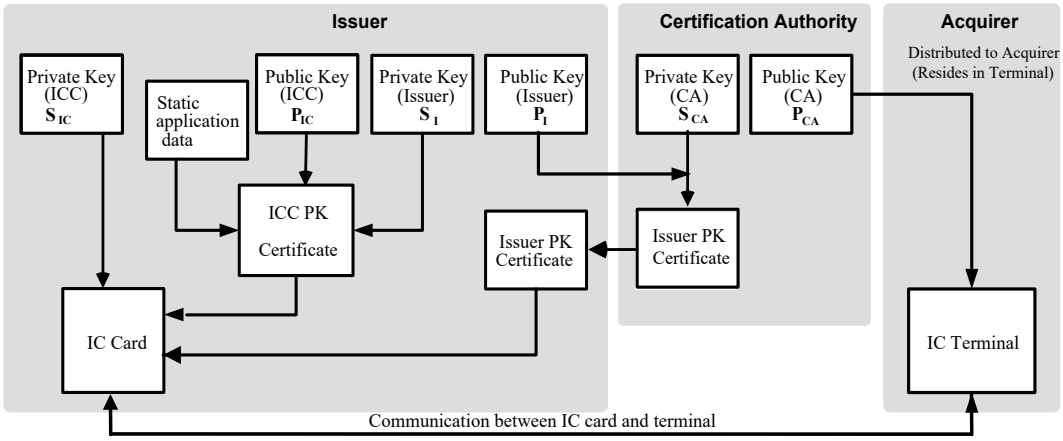
Two forms of RSA offline dynamic data authentication exist:

- Dynamic Data Authentication (DDA) executed before card action analysis, where the ICC generates a digital signature on ICC-resident/generated data identified by the ICC Dynamic Data and data received from the terminal identified by the Dynamic Data Authentication Data Object List (DDOL).
- Combined Dynamic Data Authentication/Application Cryptogram Generation (CDA) executed at issuance of the first and second GENERATE AC commands. In the case of a Transaction Certificate (TC) or Authorisation Request Cryptogram (ARQC), the ICC generates a digital signature on ICC-resident/generated data identified by the ICC Dynamic Data, which contains the TC or ARQC, and an Unpredictable Number generated by the terminal[10].

The AIP denotes the options supported by the ICC.

Offline dynamic data authentication requires the existence of a certification authority, a highly secure cryptographic facility that 'signs' the Issuer's Public Keys. Every terminal conforming to this specification shall contain the appropriate certification authority's public key(s) for every application recognised by the terminal. This specification permits multiple AIDs to share the same 'set' of Certification Authority Public Keys. The relationship between the data and the cryptographic keys is shown in Figure 2.

[10] In order to ensure that the ICC uses the correct value for the Unpredictable Number, the Issuer needs to ensure that both CDOL1 and CDOL2 contain tag '9F37'.



Card provides to Terminal:

- Issuer PK Certificate (P_I signed by the CA S_{CA})
- ICC PK Certificate (P_{IC} and static application data signed by Issuer S_I)
- Card and terminal dynamic data and digital signature (dynamic data signed by Card S_{IC})

Terminal:

- Uses P_{CA} to verify that the Issuer's P_I was signed by CA
- Uses P_I to verify that Card P_{IC} and static application data were signed by Issuer
- Uses P_{IC} to verify the card's signature on the dynamic data

Figure 2: Diagram of Offline Dynamic Data Authentication

ICCs that support DDA or CDA shall contain the data elements listed in Table 8:

Required Data Element	Length	Description
Certification Authority Public Key Index	1	Contains a binary number that indicates which of the application's Certification Authority Public Keys and its associated algorithm that reside in the terminal is to be used with this ICC.
Issuer Public Key Certificate	var.	Provided by the appropriate certification authority to the card issuer. When the terminal verifies this data element, it authenticates the Issuer Public Key plus additional data as described in section 6.3.
ICC Public Key Certificate	var.	Provided by the issuer to the ICC. When the terminal verifies this data element, it authenticates the ICC Public Key plus additional data as described in section 6.4.
Issuer Public Key Remainder	var.	See section 6.4 for further explanation.
Issuer Public Key Exponent	var.	Provided by the issuer. See section 6.4 for further explanation.
ICC Public Key Remainder	var.	See section 6.4 for further explanation.
ICC Public Key Exponent	var.	Provided by the issuer. See section 6.4 for further explanation.
ICC Private Key	var.	ICC internal. Used to generate the Signed Dynamic Application Data as described in sections 6.5 and 6.6.

Table 8: Required ICC Data Elements for Offline Dynamic Data Authentication

ICCs that support DDA or CDA shall generate the data element listed in Table 9:

Data Element	Length	Description
Signed Dynamic Application Data	var.	Generated by the ICC using the private key that corresponds to the public key authenticated in the ICC Public Key Certificate. This data element is a digital signature covering critical ICC-resident/generated and terminal data elements, as described in sections 6.5 and 6.6.

Table 9: Data Element Generated for Offline Dynamic Data Authentication

To support offline dynamic data authentication, each terminal shall be able to store six Certification Authority Public Keys per RID and shall associate with each such key the key-related information to be used with the key (so that terminals can in the future support multiple algorithms and allow an evolutionary transition from one to another, see section 11.2.2). The terminal shall be able to locate any such key (and key-related information) given the RID and Certification Authority Public Key Index as provided by the ICC.

DDA and CDA shall use a reversible algorithm as specified in section A2.1 and section B2. Section 11.2 contains an overview of the keys and certificates involved in the offline dynamic data authentication process. Sections 6.2 to 6.4 specify the initial steps in the process, namely:

- Retrieval of the Certification Authority Public Key by the terminal.
- Retrieval of the Issuer Public Key by the terminal.
- Retrieval of the ICC Public Key by the terminal.

If DDA or CDA fails then the TVR bit indicating failure of the attempted method shall be set as follows:

- If the attempted method is DDA then the terminal shall set the 'DDA failed' bit in the TVR to 1.
- If the attempted method is CDA then the terminal shall set the 'CDA failed' bit in the TVR to 1.

Sections 6.5 and 6.6 specify the dynamic signature generation and verification processes for each method.

6.1 Keys and Certificates

To support offline dynamic data authentication, an ICC shall own its own unique public key pair consisting of a private signature key and the corresponding public verification key. The ICC Public Key shall be stored on the ICC in a public key certificate.

More precisely, a three-layer public key certification scheme is used. Each ICC Public Key is certified by its issuer, and the certification authority certifies the Issuer Public Key. This implies that, for the verification of an ICC signature, the terminal first needs to verify two certificates in order to retrieve and authenticate the ICC Public Key, which is then employed to verify the ICC's dynamic signature.

The bit length of all moduli shall be a multiple of 8, the leftmost bit of its leftmost byte being 1. All lengths are given in bytes.

The signature scheme as specified in section A2.1 is applied on the data in Table 10 and on the data in Table 11 using the Certification Authority Private Key S_{CA} and the Issuer Private Key S_I in order to obtain the Issuer Public Key Certificate and ICC Public Key Certificate, respectively.

The public key pair of the certification authority has a Certification Authority Public Key Modulus of N_{CA} bytes, where $N_{CA} \leq 248$. The Certification Authority Public Key Exponent shall be equal to 3 or $2^{16} + 1$.

The public key pair of the issuer has a Public Key Modulus of N_I bytes, where $N_I \leq N_{CA} \leq 248$. N_I may be restricted to less than N_{CA}, see section D1.1 for further details. If $N_I > (N_{CA} - 36)$, the Issuer Public Key Modulus is divided into two parts, one part consisting of the $N_{CA} - 36$ most significant bytes of the modulus (the Leftmost Digits of the Issuer Public Key) and a second part consisting of the remaining $N_I - (N_{CA} - 36)$ least significant bytes of the modulus (the Issuer Public Key Remainder). The Issuer Public Key Exponent shall be equal to 3 or $2^{16} + 1$.

The public key pair of the ICC has an ICC Public Key Modulus of N_{IC} bytes, where $N_{IC} \leq N_I \leq N_{CA} \leq 248$. N_{IC} may be restricted to less than N_I, see section D1.2 for further details. If $N_{IC} > (N_I - 42)$, the ICC Public Key Modulus is divided into two parts, one part consisting of the $N_I - 42$ most significant bytes of the modulus (the Leftmost Digits of the ICC Public Key) and a second part consisting of the remaining $N_{IC} - (N_I - 42)$ least significant bytes of the modulus (the ICC Public Key Remainder). The ICC Public Key Exponent shall be equal to 3 or $2^{16} + 1$.

To execute offline dynamic data authentication, the terminal shall first retrieve and authenticate the ICC Public Key (this process is called ICC Public Key authentication). All the information necessary for ICC Public Key authentication is specified in Table 12 and stored in the ICC. With the exception of the RID, which can be obtained from the AID, this information may be retrieved with the READ RECORD command. If any of this data is missing, offline dynamic data authentication has failed.

Field Name	Length	Description	Format
Certificate Format	1	Hex value '02'	b
Issuer Identifier	4	Leftmost 3-8 digits from the PAN (padded to the right with Hex 'F's)	cn 8
Certificate Expiration Date	2	MMYY after which this certificate is invalid	n 4
Certificate Serial Number	3	Binary number unique to this certificate assigned by the certification authority	b
Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme [11]	b
Issuer Public Key Algorithm Indicator	1	Identifies the digital signature algorithm to be used with the Issuer Public Key [11]	b
Issuer Public Key Length	1	Identifies the length of the Issuer Public Key Modulus in bytes	b
Issuer Public Key Exponent Length	1	Identifies the length of the Issuer Public Key Exponent in bytes	b
Issuer Public Key or Leftmost Digits of the Issuer Public Key	$N_{CA} - 36$	If $N_I \leq N_{CA} - 36$, consists of the full Issuer Public Key padded to the right with $N_{CA} - 36 - N_I$ bytes of value 'BB' If $N_I > N_{CA} - 36$, consists of the $N_{CA} - 36$ most significant bytes of the Issuer Public Key [12]	b
Issuer Public Key Remainder	0 or $N_I - N_{CA} + 36$	Present only if $N_I > N_{CA} - 36$ and consists of the $N_I - N_{CA} + 36$ least significant bytes of the Issuer Public Key	b
Issuer Public Key Exponent	1 or 3	Issuer Public Key Exponent equal to 3 or $2^{16} + 1$	b

Table 10: Issuer Public Key Data To Be Signed by Certification Authority (i.e., input to the hash algorithm)

[11] See Annex B for specific values assigned to approved algorithms.

[12] As can be seen in section A2.1, $N_{CA} - 22$ bytes of the data signed are retrieved from the signature. Since the length of the first through the eighth data elements in Table 10 is 14 bytes, there are $N_{CA} - 22 - 14 = N_{CA} - 36$ bytes left for the data to be stored in the signature.

Field Name	Length	Description	Format
Certificate Format	1	Hex value '04'	b
Application PAN	10	PAN (padded to the right with Hex 'F's)	cn 20
Certificate Expiration Date	2	MMYY after which this certificate is invalid	n 4
Certificate Serial Number	3	Binary number unique to this certificate assigned by the issuer	b
Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme [13]	b
ICC Public Key Algorithm Indicator	1	Identifies the digital signature algorithm to be used with the ICC Public Key [13]	b
ICC Public Key Length	1	Identifies the length of the ICC Public Key Modulus in bytes	b
ICC Public Key Exponent Length	1	Identifies the length of the ICC Public Key Exponent in bytes	b
ICC Public Key or Leftmost Digits of the ICC Public Key	$N_I - 42$	If $N_{IC} \leq N_I - 42$, consists of the full ICC Public Key padded to the right with $N_I - 42 - N_{IC}$ bytes of value 'BB' If $N_{IC} > N_I - 42$, consists of the $N_I - 42$ most significant bytes of the ICC Public Key [14]	b
ICC Public Key Remainder	0 or $N_{IC} - N_I + 42$	Present only if $N_{IC} > N_I - 42$ and consists of the $N_{IC} - N_I + 42$ least significant bytes of the ICC Public Key	b
ICC Public Key Exponent	1 or 3	ICC Public Key Exponent equal to 3 or $2^{16} + 1$	b
Static Data to be Authenticated	var.	Static data to be authenticated as specified in Book 3 section 10.3 (see also section 6.1.1)	b

Table 11: ICC Public Key Data To Be Signed by Issuer (i.e., input to the hash algorithm)

[13] See Annex B for specific values assigned to approved algorithms.

[14] As can be seen in section A2.1, $N_I - 22$ bytes of the data signed are retrieved from the signature. Since the length of the first through the eighth data elements in Table 11 is 20 bytes, there are $N_I - 22 - 20 = N_I - 42$ bytes left for the data to be stored in the signature.

6.1.1 Static Data To Be Authenticated

Input to the authentication process is formed from the records identified by the AFL, followed by the value of the AIP, if identified by the optional Static Data Authentication Tag List (tag '9F4A'). If present, the Static Data Authentication Tag List shall only contain the tag '82' identifying the AIP.

Tag	Length	Value	Format
—	5	Registered Application Provider Identifier (RID)	b
'8F'	1	Certification Authority Public Key Index	b
'90'	N_{CA}	Issuer Public Key Certificate	b
'92'	$N_I - N_{CA} + 36$	Issuer Public Key Remainder, if present	b
'9F32'	1 or 3	Issuer Public Key Exponent	b
'9F46'	N_I	ICC Public Key Certificate	b
'9F48'	$N_{IC} - N_I + 42$	ICC Public Key Remainder, if present	b
'9F47'	1 or 3	ICC Public Key Exponent	b
—	var.	Static data to be authenticated as specified in Book 3 section 10.3 (see also section 6.1.1)	—

Table 12: Data Objects Required for Public Key Authentication for Offline Dynamic Data Authentication

6.1.2 Certification Revocation List

The terminal may support a Certification Revocation List (CRL) that lists the Issuer Public Key Certificates that payment systems have revoked. If, during dynamic data authentication (DDA or CDA), a concatenation of the RID and Certification Authority Public Key Index from the card and the Certificate Serial Number recovered from the Issuer Public Key Certificate is on this list, dynamic data authentication fails as described in section 6.3 step 10.

The requirements for the CRL are listed in section 5.1.2.

6.2 Retrieval of Certification Authority Public Key

The terminal reads the Certification Authority Public Key Index. Using this index and the RID, the terminal can identify and retrieve the terminal-stored Certification Authority Public Key Modulus and Exponent and associated key-related information, and the corresponding algorithm to be used. If the terminal does not have the key stored associated with this index and RID, offline dynamic data authentication has failed.

In the case of CDA, this step shall be performed before Terminal Action Analysis. (Refer to individual contactless kernel specifications for specific contactless requirements.)

6.3 Retrieval of Issuer Public Key

1. If the Issuer Public Key Certificate has a length different from the length of the Certification Authority Public Key Modulus obtained in the previous section, offline dynamic data authentication has failed.
2. In order to obtain the recovered data specified in Table 13, apply the recovery function as specified in section A2.1 on the Issuer Public Key Certificate using the Certification Authority Public Key in conjunction with the corresponding algorithm. If the Recovered Data Trailer is not equal to 'BC', offline dynamic data authentication has failed.

Field Name	Length	Description	Format
Recovered Data Header	1	Hex value '6A'	b
Certificate Format	1	Hex value '02'	b
Issuer Identifier	4	Leftmost 3-8 digits from the PAN (padded to the right with Hex 'F's)	cn 8
Certificate Expiration Date	2	MMYY after which this certificate is invalid	n 4
Certificate Serial Number	3	Binary number unique to this certificate assigned by the certification authority	b
Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme [15]	b
Issuer Public Key Algorithm Indicator	1	Identifies the digital signature algorithm to be used with the Issuer Public Key [15]	b
Issuer Public Key Length	1	Identifies the length of the Issuer Public Key Modulus in bytes	b
Issuer Public Key Exponent Length	1	Identifies the length of the Issuer Public Key Exponent in bytes	b
Issuer Public Key or Leftmost Digits of the Issuer Public Key	$N_{CA} - 36$	If $N_I \leq N_{CA} - 36$, consists of the full Issuer Public Key padded to the right with $N_{CA} - 36 - N_I$ bytes of value 'BB' If $N_I > N_{CA} - 36$, consists of the $N_{CA} - 36$ most significant bytes of the Issuer Public Key [16]	b
Hash Result	20	Hash of the Issuer Public Key and its related information	b
Recovered Data Trailer	1	Hex value 'BC'	b

Table 13: Format of Data Recovered from Issuer Public Key Certificate

3. Check the Recovered Data Header. If it is not '6A', offline dynamic data authentication has failed.
4. Check the Certificate Format. If it is not '02', offline dynamic data authentication has failed.

[15] See Annex B for specific values assigned to approved algorithms.

[16] As can be seen in section A2.1, $N_{CA} - 22$ bytes of the data signed are retrieved from the signature. Since the length of the second through the ninth data elements in Table 13 is 14 bytes, there are $N_{CA} - 22 - 14 = N_{CA} - 36$ bytes left for the data to be stored in the signature.

5. Concatenate from left to right the second to the tenth data elements in Table 13 (that is, Certificate Format through Issuer Public Key or Leftmost Digits of the Issuer Public Key), followed by the Issuer Public Key Remainder (if present), and finally the Issuer Public Key Exponent.
6. Apply the indicated hash algorithm (derived from the Hash Algorithm Indicator) to the result of the concatenation of the previous step to produce the hash result.
7. Compare the calculated hash result from the previous step with the recovered Hash Result. If they are not the same, offline dynamic data authentication has failed.
8. Verify that the Issuer Identifier matches the leftmost 3-8 PAN digits (allowing for the possible padding of the Issuer Identifier with hexadecimal 'F's). If not, offline dynamic data authentication has failed.
9. Verify that the last day of the month specified in the Certificate Expiration Date is equal to or later than today's date. If the Certificate Expiration Date is earlier than today's date, the certificate has expired, in which case offline dynamic data authentication has failed.
10. Verify that the concatenation of RID, Certification Public Key Index, and Certificate Serial Number is valid. If not, offline dynamic data authentication has failed.[17]
11. If the Issuer Public Key Algorithm Indicator is not recognised, offline dynamic data authentication has failed.
12. If all the checks above are correct, concatenate the Leftmost Digits of the Issuer Public Key and the Issuer Public Key Remainder (if present) to obtain the Issuer Public Key Modulus, and continue with the next steps for the retrieval of the ICC Public Key.

6.4 Retrieval of ICC Public Key

1. If the ICC Public Key Certificate has a length different from the length of the Issuer Public Key Modulus obtained in the previous section, offline dynamic data authentication has failed.
2. In order to obtain the recovered data specified in Table 14, apply the recovery function as specified in section A2.1 on the ICC Public Key Certificate using the Issuer Public Key in conjunction with the corresponding algorithm. If the Recovered Data Trailer is not equal to 'BC', offline dynamic data authentication has failed.

[17] This step is optional and is to allow the revocation of the Issuer Public Key Certificate against a Certification Revocation List that may be kept by the terminal (see section 6.1.2).

Field Name	Length	Description	Format
Recovered Data Header	1	Hex Value '6A'	b
Certificate Format	1	Hex Value '04'	b
Application PAN	10	PAN (padded to the right with Hex 'F's)	cn 20
Certificate Expiration Date	2	MMYY after which this certificate is invalid	n 4
Certificate Serial Number	3	Binary number unique to this certificate assigned by the issuer	b
Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme [18]	b
ICC Public Key Algorithm Indicator	1	Identifies the digital signature algorithm to be used with the ICC Public Key [18]	b
ICC Public Key Length	1	Identifies the length of the ICC Public Key Modulus in bytes	b
ICC Public Key Exponent Length	1	Identifies the length of the ICC Public Key Exponent in bytes	b
ICC Public Key or Leftmost Digits of the ICC Public Key	$N_I - 42$	If $N_{IC} \leq N_I - 42$, consists of the full ICC Public Key padded to the right with $N_I - 42 - N_{IC}$ bytes of value 'BB' [19] If $N_{IC} > N_I - 42$, consists of the $N_I - 42$ most significant bytes of the ICC Public Key	b
Hash Result	20	Hash of the ICC Public Key and its related information	b
Recovered Data Trailer	1	Hex Value 'BC'	b

Table 14: Format of Data Recovered from ICC Public Key Certificate

3. Check the Recovered Data Header. If it is not '6A', offline dynamic data authentication has failed.

[18] See Annex B for specific values assigned to approved algorithms.

[19] As can be seen in section A2.1, $N_I - 22$ bytes of the data signed are retrieved from the signature. Since the length of the second through the ninth data elements in Table 14 is 20 bytes, there are $N_I - 22 - 20 = N_I - 42$ bytes left for the data to be stored in the signature.

4. Check the Certificate Format. If it is not '04', offline dynamic data authentication has failed.
5. Concatenate from left to right the second to the tenth data elements in Table 14 (that is, Certificate Format through ICC Public Key or Leftmost Digits of the ICC Public Key), followed by the ICC Public Key Remainder (if present), the ICC Public Key Exponent, and finally the static data to be authenticated specified in Book 3 section 10.3. If the Static Data Authentication Tag List is present and contains tags other than '82', then offline dynamic data authentication has failed.
6. Apply the indicated hash algorithm (derived from the Hash Algorithm Indicator) to the result of the concatenation of the previous step to produce the hash result.
7. Compare the calculated hash result from the previous step with the recovered Hash Result. If they are not the same, offline dynamic data authentication has failed.
8. Compare the recovered PAN to the Application PAN read from the ICC. If they are not the same, offline dynamic data authentication has failed.
9. Verify that the last day of the month specified in the Certificate Expiration Date is equal to or later than today's date. If not, offline dynamic data authentication has failed.
10. If the ICC Public Key Algorithm Indicator is not recognised, offline dynamic data authentication has failed.
11. If all the checks above are correct, concatenate the Leftmost Digits of the ICC Public Key and the ICC Public Key Remainder (if present) to obtain the ICC Public Key Modulus, and continue with the actual offline dynamic data authentication described in the two sections below.

6.5 Dynamic Data Authentication (DDA)

6.5.1 Dynamic Signature Generation

The generation of the dynamic signature takes place in the following steps.

1. The terminal issues an INTERNAL AUTHENTICATE command including the concatenation of the data elements specified by the DDOL according to the rules specified in Book 3 section 5.4.

   The ICC may contain the DDOL, but there shall be a default DDOL in the terminal, specified by the payment system, for use in case the DDOL is not present in the ICC.

   It is mandatory that the DDOL contains the Unpredictable Number generated by the terminal (tag '9F37', 4 bytes binary).

   If any of the following cases occurs, DDA has failed.

   - The ICC does not contain a DDOL and the terminal does not contain a default DDOL.
   - The DDOL in the ICC does not include the Unpredictable Number.

- The ICC does not contain a DDOL and the default DDOL in the terminal does not include the Unpredictable Number.

2. The ICC generates a digital signature as described in section A2.1 on the data specified in Table 15 using its ICC Private Key S_{IC} in conjunction with the corresponding algorithm. The result is called the Signed Dynamic Application Data.

Field Name	Length	Description	Format
Signed Data Format	1	Hex value '05'	b
Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result [20]	b
ICC Dynamic Data Length	1	Identifies the length L_{DD} of the ICC Dynamic Data in bytes	b
ICC Dynamic Data	L_{DD}	Dynamic data generated by and/or stored in the ICC	—
Pad Pattern	$N_{IC} - L_{DD} - 25$	($N_{IC} - L_{DD} - 25$) padding bytes of value 'BB' [21]	b
Terminal Dynamic Data	var.	Concatenation of the data elements specified by the DDOL	—

Table 15: Dynamic Application Data To Be Signed (i.e., input to the hash algorithm)

The length L_{DD} of the ICC Dynamic Data satisfies $0 \le L_{DD} \le N_{IC} - 25$. The 3-9 leftmost bytes of the ICC Dynamic Data shall consist of the 1-byte length of the ICC Dynamic Number, followed by the 2-8 byte value of the ICC Dynamic Number (tag '9F4C', 2-8 bytes binary). The ICC Dynamic Number is a time-variant parameter generated by the ICC (it can for example be an unpredictable number or a counter incremented each time the ICC receives an INTERNAL AUTHENTICATE command).

In addition to those specified in Table 12, the data objects necessary for DDA are specified in Table 16.

[20] See Annex B for specific values assigned to approved algorithms.

[21] As can be seen in section A2.1, $N_{IC} - 22$ bytes of the data signed is recovered from the signature. Since the length of the first three data elements in Table 15 is three bytes, there are $N_{IC} - L_{DD} - 22 - 3 = N_{IC} - L_{DD} - 25$ bytes remaining for the data to be stored in the signature.

Tag	Length	Value	Format
'9F4B'	N_{IC}	Signed Dynamic Application Data	b
'9F49'	var.	DDOL	b

Table 16: Additional Data Objects Required for Dynamic Signature Generation and Verification

6.5.2 Dynamic Signature Verification

In this section it is assumed that the terminal has successfully retrieved the ICC Public Key. The verification of the dynamic signature takes place in the following steps.

1. If the Signed Dynamic Application Data has a length different from the length of the ICC Public Key Modulus, DDA has failed.
2. To obtain the recovered data specified in Table 17, apply the recovery function as specified in section A2.1 on the Signed Dynamic Application Data using the ICC Public Key in conjunction with the corresponding algorithm. If the Recovered Data Trailer is not equal to 'BC', DDA has failed.

Field Name	Length	Description	Format
Recovered Data Header	1	Hex value '6A'	b
Signed Data Format	1	Hex value '05'	b
Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme [22]	b
ICC Dynamic Data Length	1	Identifies the length of the ICC Dynamic Data in bytes	b
ICC Dynamic Data	L_{DD}	Dynamic data generated by and/or stored in the ICC	—
Pad Pattern	$N_{IC} - L_{DD} - 25$	($N_{IC} - L_{DD} - 25$) padding bytes of value 'BB' [23]	b
Hash Result	20	Hash of the Dynamic Application Data and its related information	b
Recovered Data Trailer	1	Hex value 'BC'	b

Table 17: Format of Data Recovered from Signed Dynamic Application Data

3. Check the Recovered Data Header. If it is not '6A', DDA has failed.
4. Check the Signed Data Format. If it is not '05', DDA has failed.
5. Concatenate from left to right the second to the sixth data elements in Table 17 (that is, Signed Data Format through Pad Pattern), followed by the data elements specified by the DDOL.
6. Apply the indicated hash algorithm (derived from the Hash Algorithm Indicator) to the result of the concatenation of the previous step to produce the hash result.
7. Compare the calculated hash result from the previous step with the recovered Hash Result. If they are not the same, DDA has failed.

If all the above steps were executed successfully, DDA was successful. The ICC Dynamic Number contained in the ICC Dynamic Data recovered in Table 17 shall be stored in tag '9F4C'.

[22] See Annex B for specific values assigned to approved algorithms.

[23] As can be seen in section A2.1, $N_{IC} - 22$ bytes of the data signed are retrieved from the signature. Since the length of the second through the fourth data elements in Table 17 is 3 bytes, there are $N_{IC} - L_{DD} - 22 - 3 = N_{IC} - L_{DD} - 25$ bytes left for the data to be stored in the signature.

6.6 Combined DDA/Application Cryptogram Generation (CDA)

CDA consists of a dynamic signature generated by the ICC (similar to DDA but including Application Cryptogram (AC) generation) followed by verification of the signature by the terminal.

It is applicable to both the first and second GENERATE AC commands and requires the retrieval of the relevant public keys as described in sections 6.2, 6.3, and 6.4. Since the public keys are not required until the CDA signature is verified as part of processing the response to the first GENERATE AC, retrieval of the public keys may happen any time before verifying the CDA signature.

During retrieval of the public keys, errors may result in CDA failure (TVR bit for 'CDA failed' is set to 1). These errors include but are not limited to failure of public key retrieval and invalid format of records to be authenticated (see Book 3 section 10.3). All terminals shall verify before Terminal Action Analysis that they contain the Certification Authority Public Key identified by the card. See section 6.2. (Refer to individual contactless kernel specifications for specific contactless requirements).

For the first GENERATE AC command, and for the second GENERATE AC command in the case 'unable to go online', the cryptogram type requested by the terminal is always determined by the final Terminal Action Analysis preceding the GENERATE AC command. If any of the above errors are detected prior to the final Terminal Action Analysis, then the terminal shall not request CDA in the GENERATE AC command. When the GENERATE AC command is issued with a CDA request, then if any of the above errors are detected subsequently, the eventual result will be an offline decline in accordance with the paragraphs beginning "If CDA fails in conjunction" in Book 4 section 6.3.2.

In sections 6.1.1 and 6.6.2 it is assumed that:

- Both the ICC and the terminal support CDA.
- The cryptogram to be requested is not an Application Authentication Cryptogram (AAC), i.e. Terminal Action Analysis has not resulted in offline decline.
- The TVR bit for 'CDA failed' is not set to 1 prior to final Terminal Action Analysis.
- Except when returning an AAC, the ICC always replies with a CDA signature when requested by the terminal.

In the case of the first GENERATE AC command:

- When requesting an ARQC, the terminal may request it with or without a CDA signature. When an ARQC is requested without a CDA signature, then the terminal shall set the TVR bit for 'Offline data authentication was not performed' to 1[24] prior to issuance of the GENERATE AC command. When an ARQC is requested without a CDA signature, the processes described in sections 6.6.1 and 6.6.2 are not performed.
- When requesting a TC, the terminal shall request it with a CDA signature.
- When requesting an AAC, the terminal shall request it without a CDA signature.

In the case of the second GENERATE AC command:

- The terminal shall set the TVR bit for 'Offline data authentication was not performed' to 0[25] prior to issuance of the GENERATE AC command. If the terminal is processing the transaction as 'unable to go online' then the TVR bit setting shall be done before the associated terminal action analysis.
- When requesting a TC:
  - If the terminal is processing the transaction as 'unable to go online' (and the result of terminal action analysis is to request a TC), then the terminal shall request a TC with a CDA signature.
  - If the terminal is not processing the transaction as 'unable to go online', then the terminal may request the TC with or without a CDA signature.
- When requesting an AAC, the terminal shall request it without a CDA signature.

6.6.1 Dynamic Signature Generation

The generation of the combined dynamic signature and Application Cryptogram takes place in the following steps.

1. The terminal issues a first or second GENERATE AC command with the 'CDA signature requested' bits in the GENERATE AC command set to 10 according to Book 3 sections 6.5.5.2 and 9.3.
2. If the ICC is to respond with a TC or ARQC, the ICC performs the following steps:
   a) The ICC generates the TC or ARQC.
   b) The ICC applies the hash algorithm specified by the Hash Algorithm Indicator to the concatenation from left to right of the following data elements:

      In the case of the first GENERATE AC command:

[24] This updated TVR is used if requested in CDOL1.

[25] This updated TVR is used if requested in CDOL2.

- The values of the data elements specified by, and in the order they appear in the PDOL, and sent by the terminal in the GET PROCESSING OPTIONS command.[26]
- The values of the data elements specified by, and in the order they appear in the CDOL1, and sent by the terminal in the first GENERATE AC command.[26]
- The tags, lengths, and values of the data elements returned by the ICC in the response to the GENERATE AC command in the order they are returned, with the exception of the Signed Dynamic Application Data.

In the case of the second GENERATE AC command:

- The values of the data elements specified by, and in the order they appear in the PDOL, and sent by the terminal in the GET PROCESSING OPTIONS command.[26]
- The values of the data elements specified by, and in the order they appear in the CDOL1, and sent by the terminal in the first GENERATE AC command.[26]
- The values of the data elements specified by, and in the order they appear in the CDOL2, and sent by the terminal in the second GENERATE AC command.
- The tags, lengths, and values of the data elements returned by the ICC in the response to the GENERATE AC command in the order they are returned, with the exception of the Signed Dynamic Application Data.

The 20-byte result is called the Transaction Data Hash Code.

c) The ICC applies the digital signature scheme as specified in section A2.1 on the data specified in Table 18 using its ICC Private Key S_{IC} in conjunction with the corresponding algorithm. The result is called the Signed Dynamic Application Data.

[26] At the time of issuance of the command, the terminal is required to store the values of these data elements to later perform the signature verification process as specified in section 6.6.2.

Field Name	Length	Description	Format
Signed Data Format	1	Hex Value '05'	b
Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result [27]	b
ICC Dynamic Data Length	1	Identifies the length L_{DD} of the ICC Dynamic Data in bytes	b
ICC Dynamic Data	L_{DD}	Dynamic data generated by and/or stored in the ICC (See Table 19)	—
Pad Pattern	$N_{IC} - L_{DD} - 25$	($N_{IC} - L_{DD} - 25$) padding bytes of value 'BB' [28]	b
Unpredictable Number	4	Unpredictable Number generated by the terminal	b

Table 18: Dynamic Application Data To Be Signed (i.e., input to the hash algorithm in section A2.1.2)

The length L_{DD} of the ICC Dynamic Data satisfies $0 \le L_{DD} \le N_{IC} - 25$. The 32-38 leftmost bytes of the ICC Dynamic Data shall consist of the concatenation of the data specified in Table 19.

Length	Value	Format
1	ICC Dynamic Number Length	b
2-8	ICC Dynamic Number	b
1	Cryptogram Information Data	b
8	TC or ARQC	b
20	Transaction Data Hash Code	b

Table 19: 32-38 Leftmost Bytes of ICC Dynamic Data

The ICC Dynamic Number is a time-variant parameter generated by the ICC (it can for example be an unpredictable number or a counter incremented each time the ICC receives the first GENERATE AC command during a transaction).

[27] See Annex B for specific values assigned to approved algorithms.

[28] As can be seen in section A2.1, $N_{IC} - 22$ bytes of the data signed is recovered from the signature. Since the length of the first three data elements in Table 18 is three bytes, there are $N_{IC} - L_{DD} - 22 - 3 = N_{IC} - L_{DD} - 25$ bytes remaining for the data to be stored in the signature.

The ICC response to the first GENERATE AC command shall be coded according to format 2 as specified in Book 3 section 6.5.5.4 (constructed data object with tag '77') and shall contain at least the mandatory data objects (TLV coded in the response) specified in Table 20, and optionally the Issuer Application Data.

Tag	Length	Value	Presence
'9F27'	1	Cryptogram Information Data	M
'9F36'	2	Application Transaction Counter	M
'9F4B'	N_{IC}	Signed Dynamic Application Data	M
'9F10'	var. up to 32	Issuer Application Data	O

Table 20: Data Objects Included in Response to GENERATE AC for TC or ARQC

3. If the ICC responds with an AAC, the ICC response shall be coded according to either format 1 or format 2 as specified in Book 3 section 6.5.5.4 and shall contain at least the mandatory data elements specified in Table 21, and optionally the Issuer Application Data.

Tag	Length	Value	Presence
'9F27'	1	Cryptogram Information Data	M
'9F36'	2	Application Transaction Counter	M
'9F26'	8	AAC	M
'9F10'	var. up to 32	Issuer Application Data	O

Table 21: Data Objects Included in Response to GENERATE AC for AAC

6.6.2 Dynamic Signature Verification

In this section it is assumed that the terminal has successfully retrieved the ICC Public Key as described in sections 6.2, 6.3, and 6.4.

On receiving the GENERATE AC response, the terminal determines the type of Application Cryptogram by inspecting the cleartext CID in the response.

If the ICC has responded with an AAC, then the terminal shall decline the transaction.

If the ICC has responded with a TC or ARQC, the terminal retrieves from the response the data objects specified in Table 20 and executes the following steps:

1. If the Signed Dynamic Application Data has a length different from the length of the ICC Public Key Modulus, CDA has failed.

2. To obtain the recovered data specified in Table 22, apply the recovery function as specified in section A2.1 on the Signed Dynamic Application Data using the ICC Public Key in conjunction with the corresponding algorithm. If the Recovered Data Trailer is not equal to 'BC', CDA has failed.

Field Name	Length	Description	Format
Recovered Data Header	1	Hex Value '6A'	b
Signed Data Format	1	Hex Value '05'	b
Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme [29]	b
ICC Dynamic Data Length	1	Identifies the length of the ICC Dynamic Data in bytes	b
ICC Dynamic Data	L_{DD}	Dynamic data generated by and/or stored in the ICC	—
Pad Pattern	$N_{IC} - L_{DD} - 25$	($N_{IC} - L_{DD} - 25$) padding bytes of value 'BB' [30]	b
Hash Result	20	Hash of the Dynamic Application Data and its related information	b
Recovered Data Trailer	1	Hex Value 'BC'	b

Table 22: Format of Data Recovered from Signed Dynamic Application Data

3. Check the Recovered Data Header. If it is not '6A', CDA has failed.
4. Check the Signed Data Format. If it is not '05', CDA has failed.
5. Retrieve from the ICC Dynamic Data the data specified in Table 19.
6. Check that the Cryptogram Information Data retrieved from the ICC Dynamic Data is equal to the Cryptogram Information Data obtained from the response to the GENERATE AC command. If this is not the case, CDA has failed.
7. Concatenate from left to right the second to the sixth data elements in Table 22 (that is, Signed Data Format through Pad Pattern), followed by the Unpredictable Number.

[29] See Annex B for specific values assigned to approved algorithms.

[30] As can be seen in section A2.1, $N_{IC} - 22$ bytes of the data signed are retrieved from the signature. Since the length of the second through the fourth data elements in Table 22 is 3 bytes, there are $N_{IC} - L_{DD} - 22 - 3 = N_{IC} - L_{DD} - 25$ bytes left for the data to be stored in the signature.

8. Apply the indicated hash algorithm (derived from the Hash Algorithm Indicator) to the result of the concatenation of the previous step to produce the hash result.
9. Compare the calculated hash result from the previous step with the recovered Hash Result. If they are not the same, CDA has failed.
10. Concatenate from left to right the values of the following data elements:

    In the case of the first GENERATE AC command:

    - The values of the data elements specified by, and in the order they appear in the PDOL, and sent by the terminal in the GET PROCESSING OPTIONS command.
    - The values of the data elements specified by, and in the order they appear in the CDOL1, and sent by the terminal in the first GENERATE AC command.
    - The tags, lengths, and values of the data elements returned by the ICC in the response to the GENERATE AC command in the order they are returned, with the exception of the Signed Dynamic Application Data.

    In the case of the second GENERATE AC command:

    - The values of the data elements specified by, and in the order they appear in the PDOL, and sent by the terminal in the GET PROCESSING OPTIONS command.
    - The values of the data elements specified by, and in the order they appear in the CDOL1, and sent by the terminal in the first GENERATE AC command.
    - The values of the data elements specified by, and in the order they appear in the CDOL2, and sent by the terminal in the second GENERATE AC command.
    - The tags, lengths, and values of the data elements returned by the ICC in the response to the GENERATE AC command in the order they are returned, with the exception of the Signed Dynamic Application Data.
11. Apply the indicated hash algorithm (derived from the Hash Algorithm Indicator) to the result of the concatenation of the previous step to produce the Transaction Data Hash Code.
12. Compare the calculated Transaction Data Hash Code from the previous step with the Transaction Data Hash Code retrieved from the ICC Dynamic Data in step 5. If they are not the same, CDA has failed.

If all the above steps were executed successfully, CDA was successful. The ICC Dynamic Number and the ARQC or TC contained in the ICC Dynamic Data recovered in Table 19 shall be stored in tag '9F4C' and in tag '9F26', respectively.

6.6.3 Sample CDA Flow

The figures on the following pages are an example of how a terminal might perform CDA. This sample flow provides a generalised illustration of the concepts of CDA. It does not necessarily contain all required steps and does not show parallel processing (for example, overlapping certificate recovery and signature generation). If any discrepancies are found between the text and flow, the text shall be followed.

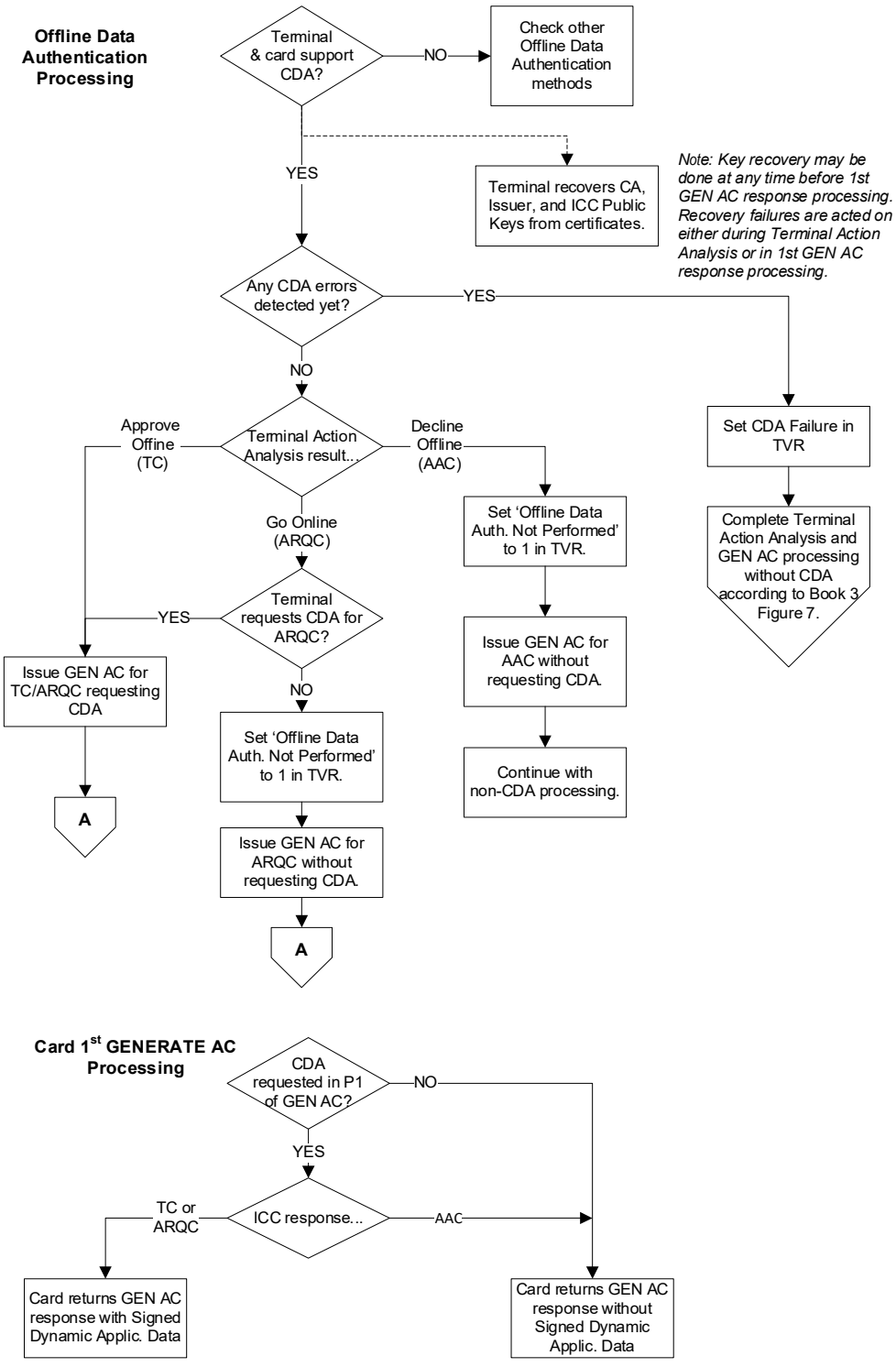


Figure 3: CDA Sample Flow Part 1 of 3

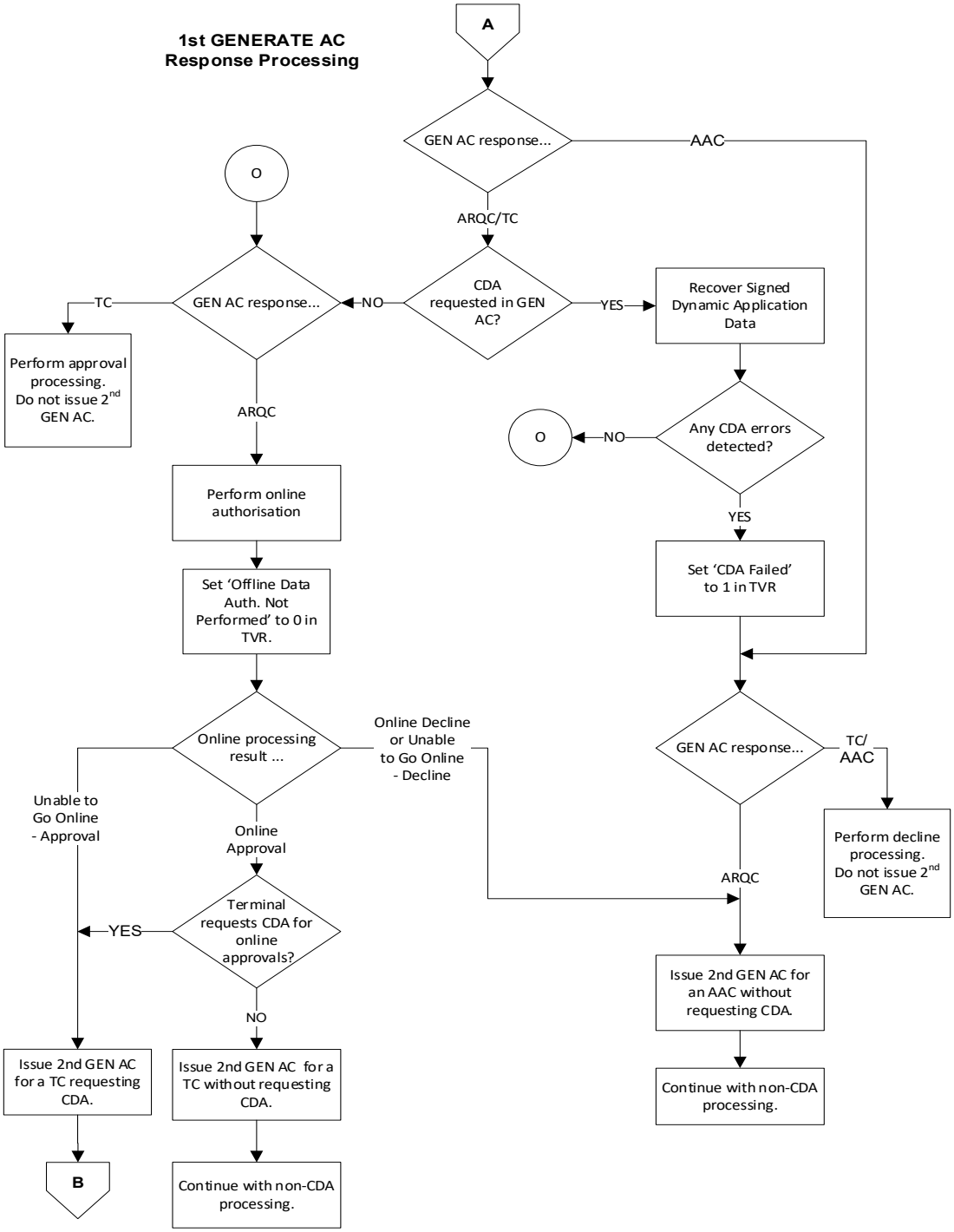


Figure 4: CDA Sample Flow Part 2 of 3

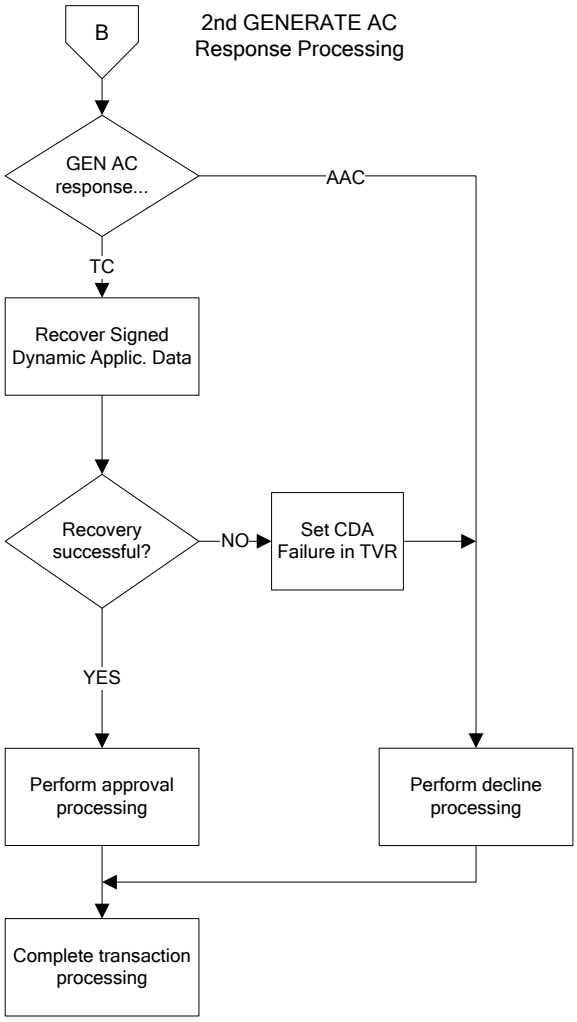


Figure 5: CDA Sample Flow Part 3 of 3

7 PIN and Biometric Data Encipherment Using RSA

This section relates only to PIN and Biometric Data Encipherment using RSA.

If supported, Personal Identification Number (PIN) encipherment for offline PIN verification is performed by the terminal using an RSA based encipherment mechanism in order to ensure the secure transfer of a PIN from a CVM capture device to the ICC.

More precisely, the ICC shall own a public key pair associated with PIN encipherment. The public key is then used by the PIN pad or a secure component of the terminal (other than the PIN pad) to encipher the PIN, and the private key is used by the ICC to decipher the enciphered PIN for verification.

The PIN block used in the data field to be enciphered shall be 8 bytes as shown in Book 3 section 6.5.12.

If offline Biometric CVM is supported, the biometric data encipherment is performed by the terminal using both symmetric and asymmetric based encipherment mechanisms in order to ensure the secure transfer of biometric data from the biometric capture device to the ICC.

More precisely, the ICC shall own a public/private key pair associated with biometric data encipherment. The public key is then used by the Biometric Processing Application, as shown in Book 4 Figure 7, or a secure component of the terminal (other than Biometric Processing Application) to encipher a seed, which generates the 128-bit AES key used to encrypt the biometric data, and the private key is used by the ICC to decipher the seed, which generates the 128-bit AES key used to decrypt the Enciphered Biometric Data for verification.

7.1 Keys and Certificates

If offline PIN encipherment or offline biometric data encipherment is supported, the ICC shall own a unique public key pair consisting of a public encipherment key and the corresponding private decipherment key. This specification allows the following two possibilities.

1. The ICC owns a specific ICC PIN Encipherment Private and Public Key. The ICC PIN Encipherment Public Key shall be stored on the ICC in a public key certificate in exactly the same way as for the ICC Public Key for offline dynamic data authentication as specified in section 6.[31]

   The ICC PIN encipherment public key pair has an ICC PIN Encipherment Public Key Modulus of N_{PE} bytes, where $N_{PE} \leq N_I \leq N_{CA} \leq 248$, N_I being the length of the Issuer Public Key Modulus (see section 6.1). If $N_{PE} > (N_I - 42)$, the ICC PIN Encipherment Public Key Modulus is divided into two parts, one part consisting of the $N_I - 42$ most significant bytes of the modulus (the Leftmost Digits of the ICC PIN Encipherment Public Key) and a second part consisting of the remaining $N_{PE} - (N_I - 42)$ least significant bytes of the modulus (the ICC PIN Encipherment Public Key Remainder).

   The ICC PIN Encipherment Public Key Exponent shall be equal to 3 or $2^{16} + 1$.

   The ICC PIN Encipherment Public Key Certificate is obtained by applying the digital signature scheme as specified in section A2.1 on the data in Table 23 using the Issuer Private Key.

   The terminal shall use the ICC PIN Encipherment key pair also for biometric data encipherment.

[31] In the case that the ICC owns a specific ICC PIN Encipherment Public Key, the format of the data recovered from the certificate is exactly the same as for dynamic data authentication (see Table 14 in section 6.4) however the data that is input to the hash algorithm when computing the certificate (see Table 23) does not include the Static Data to be Authenticated. Hence all the verification steps specified in section 6.4 are performed except in step 5 the static data to be authenticated is not included in the concatenation of data elements to be hashed in step 6.

Field Name	Length	Description	Format
Certificate Format	1	Hex Value '04'	b
Application PAN	10	PAN (padded to the right with Hex 'F's)	cn 20
Certificate Expiration Date	2	MMYY after which this certificate is invalid	n 4
Certificate Serial Number	3	Binary number unique to this certificate assigned by the issuer	b
Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme [32]	b
ICC PIN Encipherment Public Key Algorithm Indicator	1	Identifies the digital signature algorithm to be used with the ICC PIN Encipherment Public Key [32]	b
ICC PIN Encipherment Public Key Length	1	Identifies the length of the ICC PIN Encipherment Public Key Modulus in bytes	b
ICC PIN Encipherment Public Key Exponent Length	1	Identifies the length of the ICC PIN Encipherment Public Key Exponent in bytes	b
ICC PIN Encipherment Public Key or Leftmost Digits of the ICC PIN Encipherment Public Key	$N_I - 42$	If $N_{PE} \le N_I - 42$, consists of the full ICC PIN Encipherment Public Key padded to the right with $N_I - 42 - N_{PE}$ bytes of value 'BB' If $N_{PE} > N_I - 42$, consists of the $N_I - 42$ most significant bytes of the ICC PIN Encipherment Public Key [33]	b
ICC PIN Encipherment Public Key Remainder	0 or $N_{PE} - N_I + 42$	Present only if $N_{PE} > N_I - 42$ and consists of the $N_{PE} - N_I + 42$ least significant bytes of the ICC PIN Encipherment Public Key	b
ICC PIN Encipherment Public Key Exponent	1 or 3	ICC PIN Encipherment Public Key Exponent equal to 3 or $2^{16} + 1$	b

Table 23: ICC PIN Encipherment Public Key Data To Be Signed by Issuer (i.e. input to the hash algorithm)

[32] See Annex B for specific values assigned to approved algorithms.

[33] As can be seen in section A2.1, $N_I - 22$ bytes of the data signed are retrieved from the signature. Since the length of the first through the eighth data elements in Table 23 is 20 bytes, there are $N_I - 22 - 20 = N_I - 42$ bytes left for the data to be stored in the signature.

2. The ICC does not own a specific ICC PIN encipherment public key pair, but owns an ICC public key pair for offline dynamic data authentication as specified in section 6.1. This key pair can then be used for PIN encipherment and biometric data encipherment. The ICC Public Key is stored on the ICC in a public key certificate as specified in section 6.1.

The first step of PIN encipherment or biometric data encipherment shall be the retrieval of the public key to be used by the terminal for the encipherment of the PIN or biometric data. This process takes place as follows.

1. If the terminal has obtained all the data objects specified in Table 24 from the ICC, then the terminal retrieves the ICC PIN Encipherment Public Key in exactly the same way as it retrieves the ICC Public Key for offline dynamic data authentication (see section 6).
2. If the terminal has not obtained all the data objects specified in Table 24, but has obtained all the data objects specified in Table 12, then the terminal retrieves the ICC Public Key as described in section 6.
3. If the conditions under points 1 and 2 above are not satisfied or if as described in section 6.1.2 for dynamic data authentication, the Issuer Public Key Certificate has been revoked, then PIN encipherment has failed and the Offline Enciphered PIN CVM has failed, or biometric data encipherment has failed and the Offline Biometric CVM has failed.

Tag	Length	Value	Format
—	5	Registered Application Provider Identifier (RID)	b
'8F'	1	Certification Authority Public Key Index	b
'90'	N_{CA}	Issuer Public Key Certificate	b
'92'	$N_I - N_{CA} + 36$	Issuer Public Key Remainder, if present	b
'9F32'	1 or 3	Issuer Public Key Exponent	b
'9F2D'	N_I	ICC PIN Encipherment Public Key Certificate	b
'9F2E'	1 or 3	ICC PIN Encipherment Public Key Exponent	b
'9F2F'	$N_{PE} - N_I + 42$	ICC PIN Encipherment Public Key Remainder, if present	b

Table 24: Data Objects Required for Retrieval of ICC PIN Encipherment Public Key

7.2 PIN Encipherment and Verification

The exchange and verification of an enciphered PIN between terminal and ICC takes place in the following steps.

1. The PIN is entered in plaintext format on the PIN pad and a PIN block is constructed as defined in Book 3 section 6.5.12.
2. The terminal issues a GET CHALLENGE command to the ICC to obtain an 8-byte unpredictable number from the ICC. When the response to the GET CHALLENGE command is anything other than an 8 byte data value with SW1 SW2 = '9000', then the terminal shall consider that the Offline Enciphered PIN CVM has failed.
3. The terminal generates random padding consisting of N – 17 bytes, where N is the length in bytes of the public key to be used for PIN encipherment retrieved as specified in section 7.1 (hence N = N_{PE} or N = N_{IC}). The value of the random padding shall be unpredictable from the perspective of an attacker (even given knowledge of previous values) and prior to encipherment shall only exist in hardware suitable for protecting the plaintext PIN. For each encryption all values should be equally likely to be generated. This may be achieved using a Random Number Generator compliant with ISO/IEC 18031 and tested using NIST SP 800-22A (see Annex C reference 4).
4. Using the PIN Encipherment Public Key or the ICC Public Key retrieved as specified in section 7.1, the terminal applies the RSA Recovery Function as specified in section B2.1.3 to the data specified in Table 25 in order to obtain the Enciphered PIN Data.

Field Name	Length	Description	Format
Data Header	1	Hex Value '7F'	b
PIN Block	8	PIN in PIN Block	b
ICC Unpredictable Number	8	Unpredictable number obtained from the ICC with the GET CHALLENGE command	b
Random Padding	N_{IC} – 17	Random Padding generated by the terminal	b

Table 25: Data To Be Enciphered for PIN Encipherment

5. The terminal issues a VERIFY command including the Enciphered PIN Data obtained in the previous step.
6. With the ICC Private Key, the ICC applies the RSA Signing Function as specified in section B2.1.2 to the Enciphered PIN Data in order to recover the plaintext data specified in Table 25.

7. The ICC verifies whether the ICC Unpredictable Number recovered is equal to the ICC Unpredictable Number generated by the ICC with the GET CHALLENGE command. If this is not the case, Offline Enciphered PIN has failed.[34]
8. The ICC verifies whether the Data Header recovered is equal to '7F'. If this is not the case, Offline Enciphered PIN has failed.[34]
9. The ICC verifies whether the PIN included in the recovered PIN Block corresponds with the PIN stored in the ICC. If this is not the case, PIN verification has failed.[34]

If all the above steps were executed successfully, then enciphered PIN verification was successful. The ordering of steps 1 to 3 in section 7.2 is representative, not mandatory. Key retrieval (as described in section 7.1) and steps 1 to 3 in section 7.2 can be conducted in any order, provided they are all complete before the terminal applies the RSA Recovery Function (as described in step 4 of section 7.2).

[34] When recovery of the PIN Block has failed, the ICC should return SW1 SW2 = '6983' or '6984' as described in Book 3 section 10.5.1 to indicate that Offline Enciphered PIN processing has failed.

7.3 Biometric Data Encipherment and Recovery

The exchange and decipherment of Enciphered Biometric Data between terminal and ICC takes place in the following steps, as illustrated in Book 3 Figure 13.

1. The biometric template is captured on a biometric capture device and the Biometric Data Block (BDB) is constructed by the Biometric Processing Application.
2. The terminal issues a GET CHALLENGE command to the ICC to obtain an 8-byte unpredictable number from the ICC. When the response to the GET CHALLENGE command is anything other than an 8-byte data value with SW1 SW2 = '9000', then the terminal considers that the Offline Biometric CVM has failed.
3. Using the public key specified above, the terminal shall generate the Enciphered Biometric Data, as follows:

   a) The public key recovered, as described in section 7.1, has exponent e and modulus n.

   b) The terminal generates a random number r, which has the same number of bits as n, and shall be less than n, i.e. r < n. Then the terminal converts r into a byte string R as defined in ISO/IEC 18033-2: R = I2OSP(r, Len(n)).

   c) The random number R is then used to generate K using the key derivation function KDF1 defined in ISO/IEC 18033-2 with SHA-256 as the hash function: K= KDF1 (R, 48).

   d) Split K from step c) into two keys K = (BEK || BMK), where the 16 most significant bytes of K are the Biometric Encryption Key BEK used to encrypt the biometric data, and the 32 least significant bytes of K are the Biometric MAC Key BMK used to generate the MAC in step g).

   e) The random number R shall be enciphered using the RSA Transform algorithm as defined in ISO/IEC 18033-2: C = RSATransform (R, e, n). The result is the Enciphered Biometric Key Seed.

   f) Encrypt the data specified in Table 26 using the DEM1 mechanism as defined in ISO/IEC 18033-2, with the Biometric Encryption Key BEK generated in step d), in order to generate the Enciphered Biometric Data.

      The symmetric cipher (SC) used in the DEM1 mechanism shall be the SC1 defined in ISO/IEC 18033-2 with the block cipher being the AES-128 defined in ISO/IEC 18033-3.

   g) Use the Biometric MAC Key BMK generated in step d) to generate an 8-byte MAC on the Enciphered Biometric Data according to the DEM1 mechanism as defined in ISO/IEC 18033-2.

The MAC algorithm used in the DEM1 mechanism shall be HMAC, as defined in ISO/IEC 9797-2 using SHA-256 as the hash function. Note that the label L defined in DEM1 shall be null, and thus according to DEM1 the HMAC is computed over the Enciphered Biometric Data appended with 8 zero bytes.

Field Name	Length	Description	Format
ICC Unpredictable Number	8	Unpredictable number obtained from the ICC with the GET CHALLENGE command	b
Biometric Solution ID Length	1	Length of Biometric Solution ID	b
Biometric Solution ID	value of Biometric Solution ID Length	As defined in Book 3 Table 48	b
Biometric Type Length	1	Length of Biometric Type data element	b
Biometric Type	value of Biometric Type Length	As defined in Book 3 Table 49	b
Biometric Subtype	1	As defined in Book 3 Table 50	b
Biometric Data Block (BDB) Length	2	Length of BDB data element	b
Biometric Data Block (BDB)	value of BDB Length	A block of data with a specific format that contains information captured from a biometric capture device	b

Table 26: Data To Be Enciphered for Biometric Encipherment

Note: The length bytes in Table 26 are simple binary bytes, and are *not* coded as BER-TLV length fields.

4. The terminal shall construct the Biometric Verification Data Template, and order the data elements as indicated in Table 27.

   a) The value of Biometric Type is set as the plaintext value used in step 3f) when referencing Table 26.

   b) The value of Biometric Solution ID is set as the plaintext value used in step 3f) when referencing Table 26.

   c) The value of Enciphered Biometric Key Seed is set as the value generated in step 3e).

d) The value of Enciphered Biometric Data is set as the value generated in step 3f).

e) The value of MAC of Enciphered Biometric Data is set as the value generated in step 3g).

Tag	Length	Field Name
'81'	var.	Biometric Type
'90'	var.	Biometric Solution ID
'DF50'	N_{PE} or N_{IC}	Enciphered Biometric Key Seed
'DF51'	var.	Enciphered Biometric Data
'DF52'	8	MAC of Enciphered Biometric Data

Table 27: Biometric Verification Data Template

Note: The data objects in Table 27 are BER-TLV coded. The length of all TLV coded data objects are coded on the minimum number of bytes (that is, on 1 byte if < 128, on 2 bytes if in the range 128 to 255, and so on). See Book 3 section B2 for BER-TLV coding rules. There shall be no '00' padding bytes before, between, or after the BER-TLV coded data objects.

5. The terminal shall construct and issue VERIFY command(s) as following:

   a) If the length of the value field of the Biometric Verification Data Template is no more than 255 bytes, the terminal shall set the value field of the Biometric Verification Data Template as the data field of the single VERIFY command.

   b) If the length of the value field of the Biometric Verification Data Template is more than 255 bytes, the terminal shall divide the value field of the Biometric Verification Data Template into 255 byte data blocks. The last data block may be 1 to 255 bytes in length. Then the terminal shall set these data blocks as the data fields of the commands and chain the commands using command chaining defined in Book 3 section 6.5.13.

   c) The terminal shall issue the VERIFY command(s).

6. With the ICC Private key, the ICC decrypts the Enciphered Biometric Data in the following way:

   a) The private key owned by the ICC has exponent d and modulus n.

   b) The ICC decrypts the Enciphered Biometric Key Seed using the RSA Transform algorithm as defined in ISO/IEC 18033-2: R = RSATransform (C, d, n), where C is Enciphered Biometric Key Seed (tag 'DF50').

c) The ICC generates two keys using the key derivation function KDF1 as defined in ISO/IEC 18033-2 with SHA-256 as the hash function: K = KDF1 (R, 48). The 16 most significant bytes of K is the Biometric Encryption Key BEK and the 32 least significant bytes of K is the Biometric MAC Key BMK.

d) The ICC uses the BMK generated in step c) to generate an 8-byte MAC on the Enciphered Biometric Data according to the DEM1 mechanism as defined in ISO/IEC 18033-2.

   The MAC algorithm used in the DEM1 mechanism shall be HMAC, as defined in ISO/IEC 9797-2 using SHA-256 as the hash function. Note that the label L defined in DEM1 shall be null, and thus according to DEM1 the HMAC is computed over the Enciphered Biometric Data appended with 8 zero bytes.

   If the generated MAC is not equal to the MAC of Enciphered Biometric Data (tag 'DF52') included in the Biometric Verification Data Template, Offline Biometric CVM has failed.

e) The ICC decrypts the Enciphered Biometric Data using the DEM1 mechanism as defined in ISO/IEC 18033-2, with the Biometric Encryption Key BEK generated in step c).

   The symmetric cipher (SC) used in the DEM1 mechanism shall be the SC1 defined in ISO/IEC 18033-2 with the block cipher being the AES-128 defined in ISO/IEC 18033-3.

7. The ICC verifies whether the ICC Unpredictable Number recovered in step 6e) is equal to the ICC Unpredictable Number generated by the ICC with the GET CHALLENGE command. If they are not the same, Offline Biometric CVM has failed.
8. The ICC verifies whether the Biometric Solution ID recovered in step 6e) is equal to the Biometric Solution ID (tag '90') included in the Biometric Verification Data Template. If they are not the same, Offline Biometric CVM has failed.
9. The ICC verifies whether the Biometric Type recovered in step 6e) is equal to the Biometric Type (tag '81') included in the Biometric Verification Data Template. If they are not the same, Offline Biometric CVM has failed.

The ordering of steps 1 and 2 is representative, not mandatory. Key retrieval as described in section 7.1 and steps 1 and 2 can be conducted in any order, provided they are all completed before the terminal applies the RSA-KEM algorithm (as described in step 3).

8 Application Cryptogram and Issuer Authentication

The aim of this section is to provide methods for the generation of the Application Cryptograms (TC, ARQC, or AAC) generated by the ICC and the Authorisation Response Cryptogram (ARPC) generated by the issuer and verified by the ICC. For more details on the role of these cryptograms in a transaction, see Book 3 section 10.8.

Note that the methods provided in this specification are not mandatory. Issuers may decide to adopt other methods for these functions.

8.1 Application Cryptogram Generation

8.1.1 Data Selection

An Application Cryptogram consists of a Message Authentication Code (MAC) generated over data:

- referenced in the ICC's DOLs and transmitted from the terminal to the ICC in the GENERATE AC or other command, and
- accessed internally by the ICC.

The recommended minimum set of data elements to be included in Application Cryptogram generation is specified in Table 28.

Value	Source
Amount, Authorised (Numeric)	Terminal
Amount, Other (Numeric)	Terminal
Terminal Country Code	Terminal
Terminal Verification Results	Terminal
Transaction Currency Code	Terminal
Transaction Date	Terminal
Transaction Type	Terminal
Unpredictable Number	Terminal
Application Interchange Profile	ICC
Application Transaction Counter	ICC

Table 28: Recommended Minimum Set of Data Elements for Application Cryptogram Generation

8.1.2 Application Cryptogram Algorithm

The method for Application Cryptogram generation takes as input a unique ICC Application Cryptogram Master Key MK_{AC} and the data selected as described in section 8.1.1, and computes the 8-byte Application Cryptogram in the following two steps:

1. Use the session key derivation function specified in section A1.3 to derive an Application Cryptogram Session Key SK_{AC} from the ICC Application Cryptogram Master Key MK_{AC} and the 2-byte Application Transaction Counter (ATC) of the ICC.
2. Generate the 8-byte Application Cryptogram by applying the MAC algorithm specified in section A1.2 to the data selected and using the Application Cryptogram Session Key derived in the previous step. For AES the 8-byte Application Cryptogram is created by setting the parameter *s* to 8.

8.2 Issuer Authentication

Two methods are supported for generation of the ARPC used for issuer authentication. Each method can use either Triple DES or AES.

8.2.1 ARPC Method 1

ARPC Method 1 for the generation of an 8-byte ARPC using Triple DES consists of applying the Triple-DES algorithm as specified in section B1.1 to:

- the 8-byte ARQC as described in section 8.1[35]
- the 2-byte Authorisation Response Code (ARC)

using the Application Cryptogram Session Key SK_{AC} (see section 8.1) in the following way:

1. Pad the 2-byte ARC with six zero bytes to obtain the 8-byte number

   X := (ARC || '00'|| '00' || '00' || '00' || '00' || '00')

2. Compute Y := ARQC ⊕ X.
3. The 8-byte ARPC is then obtained by

   ARPC := DES3(SK_{AC})[Y]

ARPC Method 1 for the generation of an 8-byte ARPC using AES is identical to the ARPC Method 1 using Triple DES (above) except that in step 3 the 8-byte ARPC is computed using AES as specified in section B1.2 and is defined to be the leftmost 8 bytes of:

AES(SK_{AC})[Y || Y_0]

where

Y_0 := ('00' || '00' || '00' || '00' || '00' || '00' || '00' || '00').

[35] It is recommended that an ARPC is only returned if the ARQC verification is successful. However if an issuer still intends to return an ARPC when ARQC verification has failed, then the ARPC should not be calculated from the ARQC received from the network.

8.2.2 ARPC Method 2

ARPC Method 2 for the generation of a 4-byte ARPC using Triple DES consists of applying the MAC algorithm as specified in section A1.2.1 to:

- the 8-byte ARQC as described in section 8.1[36]
- the 4-byte binary Card Status Update (CSU) [37]
- the 0-8 byte binary Proprietary Authentication Data

using the Application Cryptogram Session Key SK_{AC} (see section 8.1) in the following way:

1. Concatenate the ARQC, the CSU, and the Proprietary Authentication Data.

   Y := ARQC || CSU || Proprietary Authentication Data

2. Generate a MAC over the data Y by applying the MAC algorithm specified in section A1.2 to the data defined above using the Application Cryptogram Session Key derived when computing the ARQC. For this application of the MAC algorithm, the MAC is computed according to ISO/IEC 9797-1 Algorithm 3, and the parameter *s* is set to 4, thereby yielding a 4-byte MAC.

   ARPC := MAC := MAC algorithm (SK_{AC})[Y]

3. The Issuer Authentication Data (tag '91') is formed by concatenating the resulting 4-byte ARPC, the 4-byte CSU, and the Proprietary Authentication Data.

   Issuer Authentication Data := ARPC || CSU ||
   Proprietary Authentication Data

ARPC Method 2 for the generation of a 4-byte ARPC using AES is identical to the ARPC Method 2 using Triple DES (above) except that in step 2 the MAC is computed using AES and the CMAC algorithm as specified in section A1.2.2 The parameter *s* is set to 4, thereby yielding a 4-byte MAC.

8.3 Key Management

The mechanisms for Application Cryptogram and Issuer Authentication require the management by the issuer of the unique ICC Application Cryptogram Master Keys. section A1.4 specifies two optional methods for the derivation of the ICC Application Cryptogram Master Keys from the Primary Account Number (PAN) and the PAN Sequence Number.

[36] It is recommended that an ARPC is only returned if the ARQC verification is successful. However if an issuer still intends to return an ARPC when ARQC verification has failed, then the ARPC should not be calculated from the ARQC received from the network.

[37] See Book 3 Annex A for a definition of this data item.

9 Secure Messaging

The objectives of secure messaging are to ensure data confidentiality, data integrity, and authentication of the sender. Data integrity and issuer authentication are achieved using a MAC. Data confidentiality is achieved using encipherment of the data field.

9.1 Secure Messaging Format

Secure messaging shall be according to one of the following two formats.

- **Format 1:** Secure messaging format according to ISO/IEC 7816-4, where the data field of the affected command uses Basic Encoding Rules-Tag Length Value (BER-TLV) encoding and encoding rules of ASN.1/ISO 8825-1 apply strictly. This is explicitly specified in the lowest significant nibble of the class byte of the command, which is set to 'C'. This also implies that the command header is always integrated in MAC calculation.
- **Format 2:** Secure messaging format where the data field of the affected command does not use BER-TLV encoding for secure messaging, but may use it for other purposes. In this case, the data objects contained in the data field and corresponding lengths of these data objects shall be known by the sender of a command using secure messaging and known by the currently selected application. In compliance with ISO/IEC 7816-4, secure messaging according to Format 2 is explicitly specified in the lowest significant nibble of the class byte of the command, which is set to '4'.

9.2 Secure Messaging for Integrity and Authentication

9.2.1 Command Data Field

9.2.1.1 Format 1

The data field of the secured command is composed of the following TLV data objects as shown in Figure 6.

If the command to be secured has command data, this command data is carried in the first data object[38] either as plaintext data or, if secure messaging for confidentiality is applied, as a cryptogram.

If the command data is carried as plaintext data then:

[38] EMV anticipates one data object preceding the MAC data object. Depending on the command data of the unsecured command there could be more than one such data object. For these constructions please refer to ISO/IEC 7816-4.

- If the unsecured command data is not BER-TLV encoded, then the data shall be encapsulated under tag '81'.
- If the unsecured command data is BER-TLV encoded and if the tag of any data element lies in the context specific class (range '80' to 'BF') reserved for SM-related data objects, then the command data shall be encapsulated in a constructed data object under tag 'B3'.
- If the unsecured command data is BER-TLV encoded and no tag lies in the context specific class (range '80' to 'BF') reserved for SM-related data objects, then ISO/IEC 7816-4 permits that the command data may be included without encapsulation. However, if encapsulated then the command data shall be encapsulated in a constructed data object under tag 'B3'.

Note: If it is not always apparent that the data is BER-TLV encoded then the data may be encapsulated under tag '81'.

If the command data is carried as a cryptogram then it shall be encapsulated in a data object for confidentiality as described in section 9.3.1.1.

The second data object is the MAC. Its tag is '8E', and its length shall be in the range of four to eight bytes. Note that if the command to be secured has no command data (e.g. Application Block) then this MAC is the first and only data object of the secured command.

Tag 1	Length 1	Value 1	Tag 2	Length 2	Value 2
T	L	Value (L bytes)	'8E'	'04'–'08'	MAC (4–8 bytes)

Figure 6: Format 1 Command Data Field for Secure Messaging for Integrity and Authentication

An example is provided in section D2.

9.2.1.2 Format 2

The data elements (including the MAC) contained in the data field and the corresponding lengths shall be known by the sender of a command using secure messaging and known by the currently selected application. The MAC is not BER-TLV coded and shall always be the last data element in the data field and its length shall be in the range of 4 to 8 bytes (see Figure 7).

Value 1	Value 2
Command data (if present)	MAC (4-8 bytes)

Figure 7: Format 2 Command Data Field for Secure Messaging for Integrity and Authentication

9.2.2 MAC Session Key Derivation

The first step of the MAC generation for secure messaging for integrity consists of deriving a unique MAC Session Key from the ICC's unique MAC Master Key and card-controlled data unique to the transaction. A method to do this is specified in section A1.3.1.

9.2.3 MAC Computation

The MAC is computed by applying the mechanism described in section A1.2 with the MAC Session Key derived as described in section 9.2.2 to the message to be protected.

If secure messaging is according to Format 1, the message to be protected shall be constructed from the header of the command APDU (CLA INS P1 P2) and the command data (if present) according to the rules specified in ISO/IEC 7816-4.

Note that for Format 1 the rules specified in ISO/IEC 7816-4 already define padding, so the padding of the first step of the MAC computation defined in section A1.2 shall be omitted. Specifically, the message MSG used in the MAC calculation is padded after the command header (CLA INS P1 P2 with CLA set to indicate secure messaging) and also after the data object carrying the command data if present. This data object is either a plaintext data object or, if secure messaging for confidentiality is applied, a data object for confidentiality (see section 9.3.1.1). The padding in each situation consists of one mandatory byte of '80' added to the right and then the smallest number of '00' bytes is added to the right so that the length of the resulting string is a number of bytes equal to a multiple of the block size (e.g. 8 bytes for DES or Triple DES and 16 bytes for AES).

If secure messaging is according to Format 2, the message to be protected shall be constructed according to the payment system proprietary specifications. It shall however always contain the header of the command APDU and the command data (if present).

In all cases, if the MAC used for secure messaging has been specified as having a length *s*, then the MAC is obtained by taking the leftmost (most significant) *s* bytes from the result of the calculation described above.

9.2.3.1 Format 1 MAC Chaining

If secure messaging is according to Format 1 and chaining of MACs from one command to the next is supported, the recommended method for chaining the MACs is as follows:

A value whose length is equal to the block size of the cipher (e.g. 8 bytes for DES or Triple DES and 16 bytes for AES) is inserted at the beginning of the message to be protected.[39] This value is:

- for the first or only script command, the Application Cryptogram generated by the card for the first GENERATE AC command

[39] In the terms of ISO/IEC 7816-4 and in the case that Algorithm 3 of ISO/IEC 9797-1 is used with Single DES this is equivalent to using an auxiliary block in the initial stage where this auxiliary block is the single DES encryption of the Application Cryptogram or MAC of the preceding command.

- for 8-byte block ciphers this is the 8-byte Application Cryptogram
- for 16-byte block ciphers this is the 8-byte Application Cryptogram padded to the right with 8 bytes of hexadecimal zeros
  Application Cryptogram ||'00'||'00'||'00'||'00'||'00'||'00'||'00'||'00'

- for subsequent script commands, the full cryptogram value generated by the MAC algorithm of the preceding script command (this is the full output prior to any truncation that occurs when shorter MACs are transmitted).

Note: Issuers should be aware that when multiple issuer scripts in a single response are supported, the failure of a command in one script may result in a gap in the MAC chain. This gap will cause MAC failures for commands in subsequent scripts.

9.3 Secure Messaging for Confidentiality

9.3.1 Command Data Field

9.3.1.1 Format 1

The format of a data object for confidentiality in the command data field of a secured command is shown in Figure 8.

Tag	Length	Value
T	L	Cryptogram (enciphered data field) or Padding Indicator Byte || Cryptogram (enciphered data field)

Figure 8: Format 1 – Data Object for Confidentiality

ISO/IEC 7816-4 specifies the tags which may be allocated to the cryptogram resulting from the encipherment of the data field of the unsecured command. An odd-numbered tag shall be used if the object is to be integrated in the computation of a MAC; an even-numbered tag shall be used otherwise.

If tag '86' or '87' is allocated to the data object for confidentiality, the value field of the data object for confidentiality contains the padding indicator byte followed by the cryptogram. The padding indicator byte shall be encoded according to ISO/IEC 7816-4. If another tag is used, the value field of the data object for confidentiality contains the cryptogram only.

An example is provided in section D2.

9.3.1.2 Format 2

Data encipherment is applied to the full plaintext command data field with the exception of a MAC (see Figure 9).

Value1	Value2
Cryptogram (enciphered data)	MAC (if present)

Figure 9: Format 2 Command Data Field for Secure Messaging for Confidentiality

9.3.2 Encipherment Session Key Derivation

The first step of the encipherment/decipherment for secure messaging for confidentiality consists of deriving a unique Encipherment Session Key from the ICC's unique Encipherment Master Key and card-controlled data unique to the transaction. A method to do this is specified in section A1.3.

9.3.3 Encipherment/Decipherment

Encipherment/decipherment of the plain/enciphered command data field takes place according to the mechanism described in section A1.1 with the Encipherment Session Key derived as described in section 9.3.2.

9.4 Key Management

The secure messaging mechanisms require the management by the issuer of the unique ICC MAC and Encipherment Master Keys. section A1.4 specifies methods for the derivation of the ICC MAC and Encipherment Master Keys from the Primary Account Number (PAN) and the PAN Sequence Number.

10 CA Public Key Management Principles and Policies

The contents of this section have been deleted.

With the maximum size of RSA keys already in use, there is no longer a need for a structured certification authority process to replace shorter keys, as was required for earlier versions of this specification. Some minor residual content has been incorporated into section 11.

11 Terminal Security and Key Management Requirements

This section describes the general terminal requirements for handling sensitive data, such as cryptographic keys. More specifically, it addresses key management requirements for Certification Authority Public Keys. Security requirements for PIN pads are addressed by individual payment systems.

11.1 Security Requirements for PIN Pads

A PIN pad is a tamper evident device and shall support entry of a 4-12 digit PIN. Security requirements for PIN pads are addressed by individual payment systems. For general PIN management and security principles, refer to ISO 9564.

11.2 Key Management Requirements

This section specifies the requirements for the management by acquirers of the Certification Authority Public Keys in the terminals. The requirements cover the following phases:

- Introduction of a Certification Authority Public Key in a terminal
- Storage of a Certification Authority Public Key in a terminal
- Usage of a Certification Authority Public Key in a terminal
- Withdrawal of a Certification Authority Public Key from a terminal

Terminal management functions shall allow for the installation and withdrawal of Certification Authority Public Keys by acquirers or their agents. The terminal management processes need to ensure the authenticity of the keys and protect the integrity of the installed keys. Processes also need to confirm the authenticity of the instructions to introduce or withdraw keys and to confirm that the action has been completed.

11.2.1 Certification Authority Public Key Introduction

For the introduction of a new Certification Authority Public Key, payment systems make the keys available to Acquirers. It is the acquirer's responsibility to ensure that the new Certification Authority Public Key and its related data (see section 11.2.2) is installed in its terminals. The Certification Authority Public Key will also be made available to Issuers to allow the verification of Issuer Public Key Certificates requested by an Issuer.

One process for the introduction of Certification Authority Public Keys into terminals is to install the keys during terminal manufacture and deployment. The keys may be made available by the payment systems to terminal vendors, or acquirers may specify the keys necessary for their particular market segments.

On being informed of a new key being available, Acquirers are expected to install it in their terminal base within six months.

During installation into the terminal the integrity of the Certification Authority Public Key and its related data is confirmed, see Book 4 section 10.2.

For RSA, the integrity of the Certification Authority Public Key may be protected using a Check Sum calculated in the same manner as is used to protect the integrity in storage as described in section 11.2.2.

For ECC, the Certification Authority Public Key will be self-signed for distribution according to Table 29. A diagram showing the construction of a self-signed Certification Authority Public Key Certificate is shown in Figure 10.

Successful verification of the self-signed Certification Authority Public Key Certificate also ensures that the terminal supports the algorithm suite associated with the Certification Authority Public Key and that will be used for verifying Issuer Public Key Certificates.

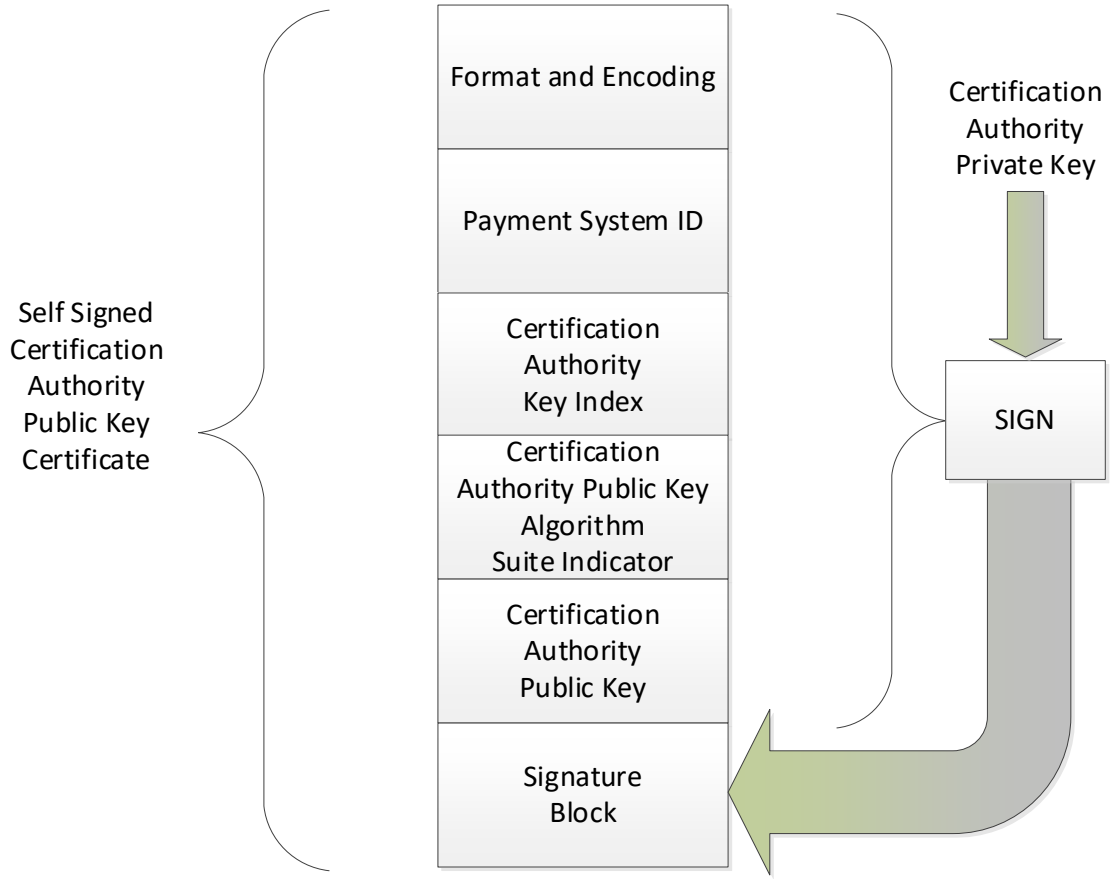


Figure 10: ECC Self-signed Certification Authority Public Key Diagram

As shown in Figure 10, the self-signed Certification Authority Public Key Certificate consists of a plain text prefix followed by a signature block. The certificate is specified in the following table where for this version of the specifications the encoding of the certificate is binary (i.e. value, value, value... with no additional identifier or length fields).

	Name	Description	Length (bytes)
1	Certificate Format	Always Hex value '20' for this version of the specifications.	1
2	Certificate Encoding	Always Hex value '00' for this version of the specifications.	1
3	RID	Identifies the Payment System with which the Certification Authority public key is associated. The RID is identified in the AID (first 5 bytes).	5
4	Certification Authority Public Key Index	When combined with the RID, uniquely identifies the Certification Authority Public Key.	1
5	Certification Authority Public Key Algorithm Suite Indicator	Indicates the algorithms to be used with the Certification Authority Public Key.	1
6	Certification Authority Public Key	Representation of Certification Authority Public Key (x-coordinate of Certification Authority public key point) on the curve identified by the Certification Authority Public Key Algorithm Suite Indicator.	N_{FIELD}
7	Certification Authority Public Key Certificate Signature	Output of digital signature ECC algorithm on concatenated first six data objects using the Certification Authority private key on the elliptic curve identified by the Certification Authority Public Key Algorithm Suite Indicator.	N_{SIG}

Table 29: ECC Self-Signed Certification Authority Public Key & Related Data (Binary Encoding)

At the end of the installation, after verifying the integrity of the public key and its related data, the terminal shall apply storage integrity protection of the public key and its related data to allow the subsequent re-verification of their integrity. This may be achieved using the Certification Authority Public Key Check Sum in Table 30 and Table 31 or a proprietary mechanism.

11.2.2 Certification Authority Public Key Storage

Terminals that support RSA-based offline data authentication, offline PIN encipherment, or biometric data encipherment shall provide support for at least six RSA Certification Authority Public Keys per RID for EMV-based payment systems' debit/credit applications based on this specification.

Terminals that support XDA or ECC ODE shall provide support for the ECC domain parameters for the two ECC curves specified in section B2.2.2 and at least ten ECC Certification Authority Public Keys for either the same or different curves (and related key information) per RID for EMV-based payment systems' payment applications based on this specification.

Each Certification Authority Public Key is uniquely identified by the 5-byte RID that identifies the payment system in question, and the 1-byte Certification Authority Public Key Index, unique per RID and assigned by that payment system to a particular Certification Authority Public Key.

For each Certification Authority Public Key, the minimum set of data elements that shall be available in the terminal is specified in Table 30 (for RSA) or Table 31 (for ECC).

For RSA, the Certification Authority Public Key Algorithm Indicator identifies the digital signature algorithm to be used with the corresponding Certification Authority Public Key. The only acceptable value at this moment is hexadecimal '01', indicating the usage of the RSA algorithm in the digital signature scheme as specified in section A2.1 and section B2.1. The Hash Algorithm Indicator specifies the hashing algorithm to produce the Hash Result in the digital signature scheme. The only acceptable value at this moment is hexadecimal '01', indicating the usage of the SHA-1 algorithm.

For ECC, the Certification Authority Public Key Algorithm Suite Indicator identifies the Algorithm Suite to be used with the corresponding Certification Authority Public Key. The only acceptable values at this moment are those specified in section B2.4.1. Like the Certification Authority Public Keys and related data, the integrity of the ECC domain parameters associated with each suite shall be protected by the terminal (as shall other sensitive terminal data and code).

The Certification Authority Public Key Index is unique and not duplicated within a RID irrespective of the algorithm associated with the Certification Authority Public Key. An RSA key and an ECC key for the same RID cannot have the same Index.

The integrity of the stored Certification Authority Public Keys and their related data should be verified periodically. This may be achieved using the Certification Authority Public Key Check Sum in Table 30 and Table 31 or a proprietary mechanism.

Field Name	Length	Description	Format
Registered Application Provider Identifier (RID)	5	Identifies the payment system with which the Certification Authority Public Key is associated	b
Certification Authority Public Key Index	1	Identifies the Certification Authority Public Key in conjunction with the RID	b
Certification Authority Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme	b
Certification Authority Public Key Algorithm Indicator	1	Identifies the digital signature algorithm to be used with the Certification Authority Public Key	b
Certification Authority Public Key Modulus	var. (max 248)	Value of the modulus part of the Certification Authority Public Key	b
Certification Authority Public Key Exponent	1 or 3	Value of the exponent part of the Certification Authority Public Key, equal to 3 or $2^{16} + 1$	b
Certification Authority Public Key Check Sum [40]	var.	A check value calculated on the concatenation of all parts of the Certification Authority Public Key (RID, Certification Authority Public Key Index, Certification Authority Public Key Modulus, Certification Authority Public Key Exponent) using SHA-1. Alternatively, one of the hash algorithms from section B2.3 may be used	b

Table 30: Minimum Set of Certification Authority Public Key Related Data Elements To Be Stored in Terminal (RSA)

[40] Only necessary if used to verify the integrity of the Certification Authority Public Key.

Field Name	Length	Description	Format
Registered Application Provider Identifier (RID)	5	Identifies the payment system with which the Certification Authority Public Key is associated	b
Certification Authority Public Key Index	1	Identifies the Certification Authority Public Key in conjunction with the RID	b
Certification Authority Public Key Algorithm Suite Indicator	1	Identifies the algorithm suite to be used with the Certification Authority Public Key. See section B2.4.1.	b
Certification Authority Public Key	$2*N_{FIELD}$	Representation of Certification Authority Public Key (xy-coordinates of Certification Authority Public Key point) on the curve identified by the Certification Authority Public Key Algorithm Suite Indicator.	b
Certification Authority Public Key Check Sum [40]	var.	A check value calculated on the concatenation of the above data elements using a hash algorithm. The same hash algorithm may be used for all keys in the terminal or the hash algorithm associated with the Certification Authority Public Key Algorithm Suite may be used	b

Table 31: Minimum Set of Certification Authority Public Key Related Data Elements To Be Stored in Terminal (ECC)

11.2.3 Certification Authority Public Key Withdrawal

When a payment system has decided to withdraw one of its Certification Authority Public Keys, an acquirer shall ensure that this Certification Authority Public Key can no longer be used in its terminals for offline static and dynamic data authentication during transactions as of a certain date.

It is recommended that acquirers do not implement expiration dates coded into the terminals but instead remove keys according to the expiration dates as indicated by the payment systems.

The following principles apply for the withdrawal by an acquirer of Certification Authority Public Keys from its terminals:

- The terminal shall be able to verify that it received the withdrawal notification error-free from the acquirer (or its agent).
- The terminal shall be able to verify that the received withdrawal notification originated from its legitimate acquirer (or its agent).

- The acquirer shall be able to confirm that a specific Certification Authority Public Key was indeed withdrawn correctly from its terminals.

Other than in the extreme case of a Certification Authority Public Key compromise, keys are expected to be withdrawn on the previously publicised expiration dates.

To assist with this, EMVCo will conduct annual review sessions for Certification Authority Public Key pair strength evaluation, using state of the art information and analysis from the fields of computer science, cryptography, and data security.

All Certification Authority Public Keys will have December 31st as planned expiration date.

Public Key expiration dates will be set well in advance to allow card portfolio lifetimes to meet business needs and Issuers should ensure that IC Cards expire no later than the expiration date of the Issuer Public Key Certificates.

Acquirers need to remove the Certification Authority Public Keys from service in their terminals within a specific grace period after expiration. This is expected to be six months starting from the planned expiration date (until June 30th of the following calendar year) but may be deferred at payment system discretion.

It may be noted that much of this material relates to the introduction of longer RSA keys, necessary as the shorter keys lose security strength. This sequence will end when the 1984 bit keys reach an expiration date. Whilst ECC keys are expected to be stable in the long term, the ability to install and withdraw keys is still necessary to allow for infrastructure changes, such as the emergence of a new domestic payment system.

11.3 Unpredictable Number Generation

This section defines an approved method for generating the Terminal Unpredictable Number (UN).

The following functions are involved:

- SHA[] is SHA-256.
- LS4B[] outputs the least significant 4 bytes of the input.

The following data elements are involved:

Abbreviation	Data Element	Description	Format
TID	Terminal ID	EMV Data element tag '9F1C'	8 bytes alpha numeric (see Book 3 Table 33)
IFDSN	IFD Serial Number	EMV Data element tag '9F1E'	8 bytes alpha numeric (see Book 3 Table 33)
TVP	Time varying parameter	Date and time with finer granularity than 0.1 second	Implementation dependent. Should be an internal value, not the external clock
RAND	Value from external RNG	Random value sourced outside the kernel (e.g. PCI-approved hardware RNG) to be used if available	8 bytes binary
P	Pre-image of UN	P is an internal register maintained in the terminal in volatile memory and never output.	32 bytes binary
Q	Persistent variant of P	Q shall be initialised to a terminal-unique random number prior to deployment. It is maintained in the terminal in non-volatile memory and never output. It is updated every time the power is cycled.	32 bytes binary

Table 32: Data Elements Used to Generate Terminal Unpredictable Number (UN)

Note that Q is held in non-volatile memory and so persists when the terminal is powered down.

UN generation is as follows

1. Power-up

   Each time the terminal is powered-up it updates Q and generates a value P as follows:

   Set Q = SHA[Q || TVP || IFDSN || TID || RAND (if available)]

   Set P = Q

2. Per transaction

   - Before a transaction, the terminal generates a UN as follows:

     Set UN = LS4B(SHA[P || RAND (if available)])

   - After a transaction (even if it fails) refresh P as follows:

     Set P = SHA[P || TVP || RAND (if available) || AC (if available)]

3. Power-down

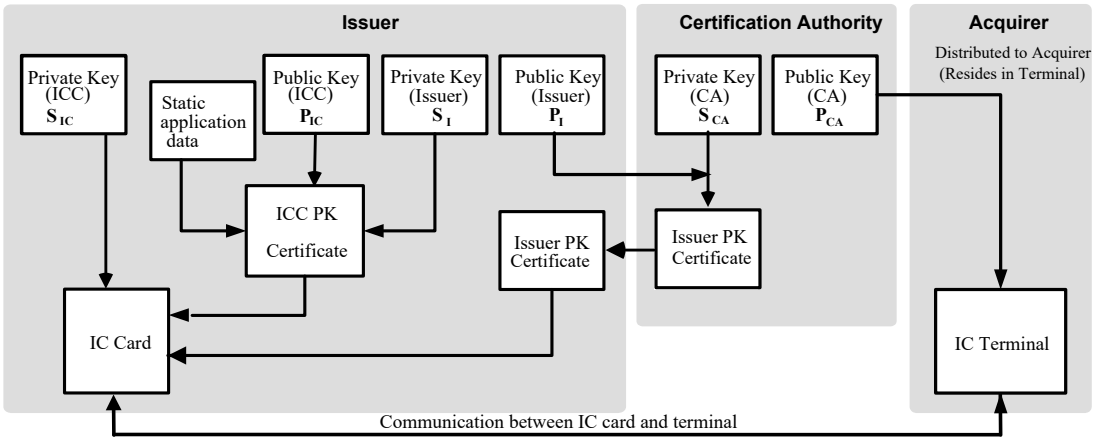
   Set Q = P

12 Offline Dynamic Data Authentication Using ECC (XDA)

This section describes Extended Data Authentication (XDA), an offline dynamic data authentication mechanism that utilises ECC public key cryptography.

When performing XDA the ICC generates a digital signature that is verified by the terminal. This digital signature authenticates the ICC and confirms the legitimacy of the Application Cryptogram (AC) and other critical data originating from the ICC and terminal. This precludes the counterfeiting of any such card and provides data integrity between the card and the terminal. XDA signatures are generated by the ICC in response to the GENERATE AC command.

XDA requires the existence of a certification authority, a highly secure cryptographic facility that 'signs' the Issuer's Public Keys. If a terminal application is configured to be XDA capable then the terminal shall contain the appropriate certification authority's public key(s) for that application. This specification permits multiple AIDs to share the same Certification Authority Public Keys. The relationship between the data and the cryptographic keys is shown in Figure 11. See also section 11.2.



Card provides to Terminal:

- Issuer PK Certificate (P_I signed by the CA S_{CA})
- ICC PK Certificate (P_{IC} and static application data signed by Issuer S_I)
- Card and terminal dynamic data and digital signature (dynamic data signed by Card S_{IC})

Terminal:

- Uses P_{CA} to verify that the Issuer's P_I was signed by CA
- Uses P_I to verify that Card P_{IC} and static application data were signed by Issuer
- Uses P_{IC} to verify the card's signature on the dynamic data

Figure 11: Diagram of Offline Dynamic Data Authentication

ICCs that support XDA contain an ICC Private Key that the ICC uses for generating the Signed Dynamic Application Data.

ICCs that support XDA shall generate the data element listed in Table 33:

Data Element	Length	Description
Signed Dynamic Application Data	var.	Generated by the ICC using the private key that corresponds to the public key authenticated in the ICC Public Key Certificate. This data element is a digital signature covering critical ICC and terminal data, as described in section 12.5.

Table 33: Data Element Generated for XDA

XDA uses the signature scheme specified in section A2.2 and section B2.2. Section 12.1 contains an overview of the keys and certificates involved in the XDA process. Sections 12.2, 12.3, and 12.4 specify the initial steps in the process, namely:

- Retrieval of the Certification Authority Public Key by the terminal
- Authentication and Recovery of the Issuer Public Key by the terminal
- Authentication and Recovery of the ICC Public Key by the terminal

If the first of these steps fails then the CA ECC key is considered to be missing and therefore XDA is considered to have failed. If either of the last two of these steps fail then ECC key recovery is considered to have failed and therefore XDA is considered to have failed. Terminal actions in case of ECC key recovery failures are addressed in Book 4 section 6.3. Note that ECC key recovery failure is not reflected in the Terminal Verification Results before the GENERATE AC command is sent and so cannot influence Terminal Action Analysis until after the GENERATE AC command is sent.

The final step of the XDA process consists of generation by the ICC and verification by the terminal of the Signed Dynamic Application Data. Section 12.5 specifies the dynamic signature generation and verification processes for XDA.

If dynamic signature verification fails then XDA is considered to have failed. XDA shall only be considered successful if CA key retrieval is successful and ECC key recovery is successful and XDA signature verification is successful.

12.1 Keys and Certificates

To support offline dynamic data authentication an ICC shall have its own public key pair consisting of a private signature key and the corresponding public verification key. The ICC Public Key shall be stored in the ICC in a public key certificate tag '9F46'. To support offline data encipherment (PIN or biometrics) a separate public key certificate is required with an independent Algorithm Suite Indicator (see section 13). Two certificates are still required even if the same key pair is used for both XDA and ODE.[41]

More precisely, a three-layer public key certification scheme is used. Each ICC Public Key is certified by its issuer, and the Certification Authority certifies the Issuer Public Key. This implies that, for the verification of an ICC signature, the terminal first needs to verify two certificates in order to authenticate and recover the ICC Public Key, which is then employed to verify the ICC's dynamic signature (or to encipher offline data).

To generate the Issuer Public Key Certificate, the Certification Authority signs the Issuer Public Key data using the Certification Authority Private Key S_{CA} with the signature scheme specified in section A2.2.

To generate the ICC Public Key Certificate, the Issuer signs the ICC Public Key data using the Issuer Private Key S_I with the signature scheme specified in section A2.2.

Aside from the Certification Authority Public Key, all the information necessary for ICC Public Key authentication is specified in Table 34 and stored in the ICC. With the exception of the RID, which can be obtained from the AID, this information may be retrieved with the READ RECORD command.

All public keys are ECC keys (a hybrid approach using a combination of RSA and ECC keys is not supported).

[41] This is unlike RSA certificates, where if a public key is used for both ODA and ODE, then only a single public key certificate is needed.

Tag	Name	Description
—	Registered Application Provider Identifier (RID)	5 byte identifier obtained from the Application Identifier (AID)
'8F'	Certification Authority Public Key Index	1 byte which indicates which of the application's Certification Authority Public Keys that reside in the terminal is to be used with this ICC.
'90'	Issuer Public Key Certificate	Includes a digital signature on the Issuer Public Key and other data that enables the terminal to obtain an authenticated copy of the Issuer Public Key and other data as described in section 12.3.
'9F46'	ICC Public Key Certificate	Includes a digital signature on the ICC Public Key and other data that enables the terminal to obtain an authenticated copy of the ICC Public Key and other data as described in section 12.4.
'9F2D'	ICC Public Key Certificate for ODE	Includes a digital signature on the ICC Public Key for ODE and other data that enables the terminal to obtain an authenticated copy of the ICC Public Key for ODE and other data as described in section 12.4. (only needed if ODE is supported)
—	Static data to be authenticated	The ICC-resident static data to be authenticated as specified in Book 3 section 10.3.

Table 34: ICC Data Required for Public Key Authentication for XDA and ECC ODE

12.1.1 Certification Revocation List

The terminal may support a Certification Revocation List (CRL) that lists the Issuer Public Key Certificates that payment systems have revoked. If during XDA a concatenation of the RID and Certification Authority Public Key Index from the card and the Certificate Serial Number recovered from the Issuer Public Key Certificate is on this list, ECC key recovery fails as described in section 12.3 step 7.

The requirements for the CRL are listed in section 5.1.2.

12.2 Retrieval of Certification Authority Public Key

To support offline dynamic data authentication (and offline data encipherment), terminals store Certification Authority Public Keys for each RID along with associated key-related information (see section 11.2.2). The terminal is able to locate any such key (and key-related information) given the RID and Certification Authority Public Key Index as provided by the ICC.

Using the RID and Certification Authority Public Key Index read from the card, the terminal shall identify and retrieve the terminal-stored Certification Authority Public Key and associated key-related information, including the Certification Authority Public Key Algorithm Suite Indicator.

If the terminal does not have the public key associated with this index and RID or the index is missing (per Book 3 section 7.5), then the terminal shall consider the CA ECC key missing and XDA (and/or ODE) processing is considered to have failed. Otherwise, the terminal shall continue processing with the steps in the following section.

12.3 Authentication and Recovery of Issuer Public Key

The Issuer Public Key is contained in the Issuer Public Key Certificate (tag '90') included in data read from the ICC. If the Issuer Public Key Certificate is missing (per Book 3 section 7.5) then the terminal shall consider ECC key recovery as failed and XDA (and/or ODE) processing is considered to have failed.

Authentication and recovery of the Issuer Public Key consists of verification of the signature in the Issuer Public Key Certificate using the Certification Authority Public Key and the algorithms indicated by Certification Authority Public Key Algorithm Suite Indicator, verification of the format and values of the data elements found in the certificate, and recovery of the y-coordinate from the x-coordinate. This process is described below.

The result of these verifications is either a valid and authentic copy of the Issuer Public Key, or failure.

Table 35 shows the Issuer Public Key Certificate and is the data signed by the Certification Authority Private Key and the resulting signature. That is, the concatenation of the data elements 1 through 9 in this table forms the input to the signature function in section A2.2.2.

	Name	Description	Length (bytes)
1	Issuer Certificate Format	Always Hex value '12' for this version of the specifications.	1
2	Issuer Certificate Encoding	Always Hex value '00' for this version of the specifications.	1
3	Issuer Identifier	Uniquely identifies the Issuer in the context of the RID (leftmost 3-10 digits from the PAN padded to the right with hex 'F's).	5
4	Issuer Public Key Algorithm Suite Indicator	Identifies the algorithm suite to be used with the Issuer Public Key when verifying issuer signatures. See section B2.4.	1
5	Issuer Certificate Expiration Date	Issuer Certificate Expiration Date (YYYYMMDD, UTC).	4
6	Issuer Certificate Serial Number	Number assigned to the issuer certificate that is unique amongst all certificates created by the payment system key that created the issuer certificate.	3
7	RID	Payment System identified in the AID (first 5 bytes).	5
8	Certification Authority Public Key Index	In conjunction with the RID, identifies which of the Kernel application's Certification Authority Public Keys (and associated algorithms) to use when verifying the issuer certificate.	1
9	Issuer Public Key	Representation of Issuer Public Key (x-coordinate of Issuer Public Key point) on the curve identified by the Issuer Public Key Algorithm Suite Indicator.	N_{FIELD}
10	Issuer Public Key Certificate Signature	A digital signature on data objects 1 through 9 of this table. Verified using the Certification Authority Public Key and algorithms identified by data object 8.	N_{SIG}

Table 35: Issuer Public Key Certificate Tag '90' (ECC)

When using EC-SDSA, the length N_{SIG} of a digital signature is equal to the length N_{HASH} of the hash plus the length of a field element. Thus, if the signature algorithm suite uses P-256 and a 256-bit hash algorithm (such as SHA-256), then $N_{HASH} = N_{FIELD} = 32$ and $N_{SIG} = N_{HASH} + N_{FIELD} = 64$. Furthermore, for item 10 in Table 35 the lengths N_{SIG}, N_{HASH}, and N_{FIELD} are determined by the Certification Authority Public Key Algorithm Suite Indicator whereas for item 9 in Table 35 the length N_{FIELD} is determined by the Issuer Public Key Algorithm Suite Indicator.

To authenticate the Issuer Public Key the terminal shall perform the following steps:

1. Check that the length in bytes of the Issuer Public Key Certificate as defined in Table 35 is at least 21 bytes. If this fails, ECC key recovery has failed.
2. Check the Issuer Certificate Format. If it is not '12', ECC key recovery has failed.
3. Check the Issuer Certificate Encoding. If it is not '00', ECC key recovery has failed.
4. Verify that the Issuer Identifier matches the leftmost 3 – 10 digits of the PAN (tag '5A') (allowing for the possible padding of the Issuer Identifier with hexadecimal 'F's). If not, ECC key recovery has failed.
5. Verify that the date specified in the Issuer Certificate Expiration Date is equal to or later than today's date. If the Issuer Certificate Expiration Date is earlier than today's date, the certificate has expired, in which case ECC key recovery has failed.
6. Verify that the RID and the Certification Authority Public Key Index in the certificate matches the RID and the Certification Authority Public Key Index used to retrieve the Certification Authority Public Key. If not, ECC key recovery has failed.
7. Verify that the concatenation of RID, Certification Authority Public Key Index, and Issuer Certificate Serial Number is valid. If not, ECC key recovery has failed.[42]
8. Verify that the Issuer Public Key Algorithm Suite Indicator is in accordance with section B2.4.1. If not, then ECC key recovery has failed.
9. Check that the length in bytes of the Issuer Public Key Certificate as defined in Table 35 is equal to $N_{FIELD} + N_{SIG} + 21$. If this fails, ECC key recovery has failed.
10. Concatenate the first 9 data elements of the Issuer Public Key Certificate
11. Using the Certification Authority Public Key and the algorithms indicated by the associated Certification Authority Public Key Algorithm Suite Indicator, verify as specified in section A2.2.3 the Issuer Public Key Certificate Signature contained in the Issuer Public Key Certificate (tag '90') with the concatenated data from step 10 being the MSG as specified in section A2.2.3. If signature verification fails then ECC key recovery has failed.
12. Recover the y-coordinate of the Issuer Public Key point using the Point4x() function applied to the x-coordinate obtained from the Issuer Public Key field of the certificate (item 9 in Table 35) as described in section B2.2.1(e).

If the above steps were all successful, then authentication and recovery of the Issuer Public Key is successful and the terminal shall continue with the steps in the following section. If any of the above steps were not successful (i.e. ECC key recovery has failed), then XDA (and/or ODE) processing is considered to have failed.

[42] This step is optional and is to allow the revocation of the Issuer Public Key Certificate against a list that may be kept by the terminal.

12.4 Authentication and Recovery of ICC Public Key

The ICC Public Key is contained in the ICC Public Key Certificate (tag '9F46') included in data read from the ICC. If ODE is supported (see section 13) then the ICC Public Key for ODE is contained in the ICC Public Key Certificate for ODE (tag '9F2D') included in data read from the ICC. If the ICC Public Key Certificate is missing (per Book 3 section 7.5) then the terminal shall consider ECC key recovery as failed and XDA (and/or ODE) processing is considered to have failed.

Authentication and recovery of the ICC Public Key consists of verification of the signature in the ICC Public Key Certificate using the Issuer Public Key and the algorithms indicated by the Issuer Public Key Algorithm Suite Indicator, verification of the format and values of the data elements found in the certificate, and recovery of the y-coordinate from the x-coordinate. This process is described below (under ICC Public Key Authentication and Recovery).

As well as the ICC Public Key, the signature in the ICC Public Key Certificate is intended to protect the authenticity of certain other static card data. So verification of the signature additionally involves the verification of the format and values of other data elements protected by the certificate and this includes the Static Data to be Authenticated as specified in Book 3 section 10.3. This process is described below (under ICC Public Key Authentication and Recovery). The same certificate format and certificate verification process is used for ICC Public Key Certificates for ODE.

The result of these verifications is either a valid and authentic copy of the ICC Public Key and the Static Data to be Authenticated, or ECC key recovery failure.

	Name	Description	Length (bytes)
1	ICC Certificate Format	Always Hex value '14' for this version of the specifications.	1
2	ICC Certificate Encoding	Always Hex value '00' for this version of the specifications.	1
3	ICC Public Key Algorithm Suite Indicator	Identifies the algorithm suite to be used with the certified ICC Public Key (a Signature Algorithm Suite in the case of tag '9F46' and an ODE Algorithm Suite in the case of tag '9F2D'. See sections B2.4.1 and B2.4.2 respectively).	1
4	ICC Certificate Expiration Date	ICC Certificate Expiration Date (YYYYMMDD, UTC).	4
5	ICC Certificate Expiration Time	ICC Certificate Expiration Time (HHMM, UTC).	2
6	ICC Certificate Serial Number	Number assigned to the ICC certificate that is unique amongst all certificates created by the issuer key that created the ICC certificate.	6
7	ICCD Hash Encoding	For this version of the specifications, always Hex value '00', indicating TLV encoding of the input is used when computing the ICCD Hash.	1
8	ICCD Hash Algorithm Indicator	Identifies the hash algorithm used to calculate the ICCD Hash. See section B2.3.	1
9	ICCD Hash	Hash of the Issuer Certified Card Data (ICCD) using the hash function identified by the ICCD Hash Algorithm Indicator.	N_{HASH}
10	ICC Public Key	Representation of ICC Public Key (x-coordinate of ICC Public Key point) on the curve identified by the ICC Public Key Algorithm Suite Indicator.	N_{FIELD}
11	ICC Public Key Certificate Signature	A digital signature on data objects 1 through 10 of this table. Verified using the Issuer Public Key obtained from the certificate in Table 35 and the algorithms identified by the Issuer Public Key Algorithm Suite Indicator.	N_{SIG}

Table 36: ICC Public Key Certificate Tag '9F46' (XDA) and Tag '9F2D' (ODE)

When using EC-SDSA the length N_{SIG} of a digital signature is equal to the length of the hash plus the length of a field element. Thus, if the signature algorithm suite uses P-256 and a 256-bit hash algorithm (such as SHA-256) then $N_{HASH} = N_{FIELD} = 32$ and $N_{SIG} = N_{HASH} + N_{FIELD} = 64$. Furthermore, for item 11 in Table 36 the lengths N_{SIG}, N_{HASH}, and N_{FIELD} are determined by the Issuer Public Key Algorithm Suite Indicator whereas for item 10 in Table 36 the length N_{FIELD} is determined by the ICC Public Key Algorithm Suite Indicator and for item 9 in Table 36 the length N_{HASH} is determined by the ICCD Hash Algorithm Indicator.

Note: If the issuer signature algorithm suite uses P-521/SHA-512 (signature algorithm suite '11') and the ICC Public Key is P-521 (signature algorithm suite '11' or encryption algorithm suite '01') then the ICCD Hash is limited to using SHA-256 (hash algorithm '02') in order to prevent exceeding length restrictions.

ICCD Hash

As part of ICC Public Key authentication, the terminal shall check that the hash of the Issuer Certified Card Data (ICCD) is equal to the ICCD Hash included in the ICC Public Key Certificate (see Table 36).

The ICCD is the Static Data to be Authenticated which is equal to the concatenation of records identified by the AFL as participating in Offline Data Authentication, the tag, length, and value of Application Interchange Profile, the tag, length, and value of Application Identifier (AID) – terminal, and the tag, length, and value of PDOL (if received from the card) as specified in Book 3 section 10.3.

It is expected that authenticated records will include Application Primary Account Number (PAN), Application Primary Account Number (PAN) Sequence Number, and Application Expiration Date.

Note: For this version of the specifications, the encoding method used for the ICCD is restricted to TLV encoding and so the ICCD Hash will be computed over the data objects formatted in TLV representation as described in Book 3 section 10.3.

ICC Public Key Authentication and Recovery

The terminal shall perform the following steps:

1. Check that the length in bytes of the ICC Public Key Certificate as defined in Table 36 is at least 17 bytes. If this fails, ECC key recovery has failed.
2. Check that the ICC Certificate Format is '14'. If this fails, ECC key recovery has failed.
3. Check the ICC Certificate Encoding. If it is not '00', ECC key recovery has failed.
4. Check that the certificate expiration time and date recovered from the certificate is later than the current time and date. If this is not the case then the certificate has expired and ECC key recovery has failed.

5. Verify that the ICC Public Key Algorithm Suite Indicator is in accordance with section B2.4 (B2.4.1 for XDA if tag '9F46' or B2.4.2 for ODE if tag '9F2D'). If not, then ECC key recovery has failed.
6. Verify that the ICCD Hash Algorithm Indicator is an identifier in accordance with section B2.3. If not, then ECC key recovery has failed.
7. Check that the length in bytes of the ICC Public Key Certificate as defined in Table 36 is equal to 17 + N_{HASH} + N_{FIELD} + N_{SIG}. If this fails, ECC key recovery has failed.
8. Check that the hash of the Issuer Certified Card Data (ICCD) using the hash algorithm identified by ICCD Hash Algorithm Indicator is equal to the ICCD Hash included in the ICC Public Key Certificate (see Table 36). If this check fails then ECC key recovery has failed.
9. Concatenate the first 10 data elements of the ICC Public Key Certificate.
10. Using the Issuer Public Key and the algorithms indicated by the associated Issuer Public Key Algorithm Suite Indicator, verify as specified in section A2.2.3 the Digital Signature contained in the ICC Public Key Certificate with the concatenated data from step 9 being the MSG as specified in section A2.2.3. If signature verification fails, ECC key recovery has failed.
11. Recover the y-coordinate of the ICC Public Key point using the Point4x() function applied to the x-coordinate obtained from the ICC Public Key field of the certificate (item 10 in Table 36) as described in section B2.2.1(e).

If the above steps were all successful, then authentication and recovery of the ICC Public Key is successful and the terminal shall continue with the steps in the following section (in case of tag '9F46') or section 13.2 (in case of tag '9F2D'). If any of the above steps were not successful (i.e. ECC key recovery has failed), then XDA processing is considered to have failed (in case of tag '9F46') or ODE processing is considered to have failed (in case of tag '9F2D').

12.5 XDA Signature Generation and Verification

Before the terminal can verify the XDA signature it needs to authenticate and recover the relevant public keys as described in sections 12.3 and 12.4. The authentication and recovery of the public keys may happen at any time before processing the response to the first GENERATE AC command.

12.5.1 Requesting, Generating, and Verifying an XDA Signature

Conditions for requesting an XDA signature are addressed in Book 3 section 9.3 and Book 4 section 6.3.2.2.

The terminal always requests an XDA signature in the GENERATE AC command if XDA is selected. Thus, an XDA signature is requested even if an AAC is requested and even if CA ECC key retrieval has failed or ECC key recovery has failed or, in the case of a second GENERATE AC command, verification of the XDA signature requested in the first GENERATE AC command has failed.[43]

The ICC generates an XDA signature according to section 12.5.2 if the terminal has requested an XDA signature in the GENERATE AC command.

The terminal verifies an XDA signature according to section 12.5.3 if the terminal has requested and received an XDA signature, the Application Cryptogram returned is a TC or ARQC and XDA has not already failed. The terminal does not attempt XDA signature verification if an AAC is returned or if XDA has already failed.

12.5.2 Dynamic Signature Generation

The generation of the XDA signature takes place in the following steps.

1. The terminal issues a GENERATE AC command with the 'XDA signature requested' bits in the GENERATE AC command set to 01 according to Book 3 sections 6.5.5.2 and 9.3.
2. The ICC performs the following steps:
   a) The ICC generates the AC.
   b) The ICC generates a digital signature as described in section A2.2.2 on the data specified in Table 37 using its ICC Private Key S_{IC} in conjunction with the corresponding algorithms. The resulting digital signature is called the Signed Dynamic Application Data.

[43] The reason for always requesting an XDA signature, even if an AAC is requested, is to avoid complicated logic for circumstances that will rarely occur.

Creation of Signed Dynamic Application Data

The ICC shall sign the dynamic application data shown in Table 37 using the ICC Private Key and the signature function in section A2.2.2. The input (MSG) to the signature function is the concatenation of the data elements in Table 37 and the output is the Digital Signature in the Signed Dynamic Application Data.

Field Name	Length	Description	Format
Signed Data Format	1	Hex value '15'	b
PDOL data (1st GEN AC only)	$L_{PDOL\ data}$	Values of the data elements specified by the PDOL and in the order they appear in the PDOL and sent by the terminal in the GET PROCESSING OPTIONS command (Not present if no PDOL was present in final SELECT)	b
CDOL1 data (1st GEN AC only)	$L_{CDOL1\ data}$	Values of the data elements specified by the CDOL1 in the order they were sent by the terminal in the first GENERATE AC command.	b
CDOL2 data (2nd GEN AC only)	$L_{CDOL2\ data}$	(Second GENERATE AC only) Values of the data elements specified by the CDOL2 in the order they were sent by the terminal in the second GENERATE AC command.	b
Response data	L_{RESP}	TLV-encoded data in the response to the GEN AC command not including the Signed Dynamic Application Data. See Table 39.	b

Table 37: Dynamic Application Data To Be Signed (XDA)

The Response Data in Table 37 shall be exactly (including their order) as returned in the response to the GENERATE AC command, but not including the Signed Dynamic Application Data.

The Signed Dynamic Application Data consists of the concatenation of the Signed Data Format and the Digital Signature. This is illustrated in Table 38.

Field Name	Length	Description	Format
Signed Data Format	1	Hex value '15'	b
Digital Signature	N_{SIG}	This is the ICC ECC digital signature (r, s).	b

Table 38: Format of XDA Signed Dynamic Application Data

The ICC response to GENERATE AC commands with XDA signature is coded according to format 2 as specified in Book 3 section 6.5.5.4 (constructed data object with tag '77') and contains at least the mandatory data objects (TLV coded in the response) specified in Table 39, and optionally the Issuer Application Data and other data objects.

Tag	Length	Value	Presence
'9F27'	1	Cryptogram Information Data	M
'9F36'	2	Application Transaction Counter	M
'9F26'	8	Application Cryptogram	M
'9F10'	var. up to 32	Issuer Application Data	O
	var.	Other Data Objects	O
'9F4B'	N_{SIG} + 1	Signed Dynamic Application Data	M

Table 39: Data Objects Included in Response to GENERATE AC with XDA Signature

12.5.3 Dynamic Signature Verification

This section continues the terminal processing from section 12.5.2 step 1.

The terminal receives and parses the GENERATE AC response.

Note: The terminal can determine the type of Application Cryptogram by inspecting the CID in the response.

The terminal shall continue XDA processing and verify the XDA signature in the following steps:

Verification of Signed Dynamic Application Data

1. Parse the Signed Dynamic Application Data according to Table 38. If this fails, XDA signature verification has failed.
2. Check the Signed Data Format. If it is not '15', XDA signature verification has failed.
3. Verify the signature (*r*, *s*) from Table 38 on the data in Table 37 as specified in section A2.2.3, using the ICC Public Key and the associated ICC Public Key Algorithm Suite. If the verification fails, XDA signature verification has failed.

If all the above steps were executed successfully, then XDA signature verification was successful. Otherwise XDA signature verification has failed.

12.5.4 Sample XDA Flow

The figures on the following pages are an example of how a terminal might perform XDA. This sample flow provides a generalised illustration of the concepts of XDA. It does not necessarily contain all required steps and does not show parallel processing (for example, overlapping certificate recovery and signature generation). If any discrepancies are found between the text and flow, the text shall be followed. Book 4 sections 6.3.2.2.1 to 6.3.2.2.4 describe the processing for XDA-related failures.

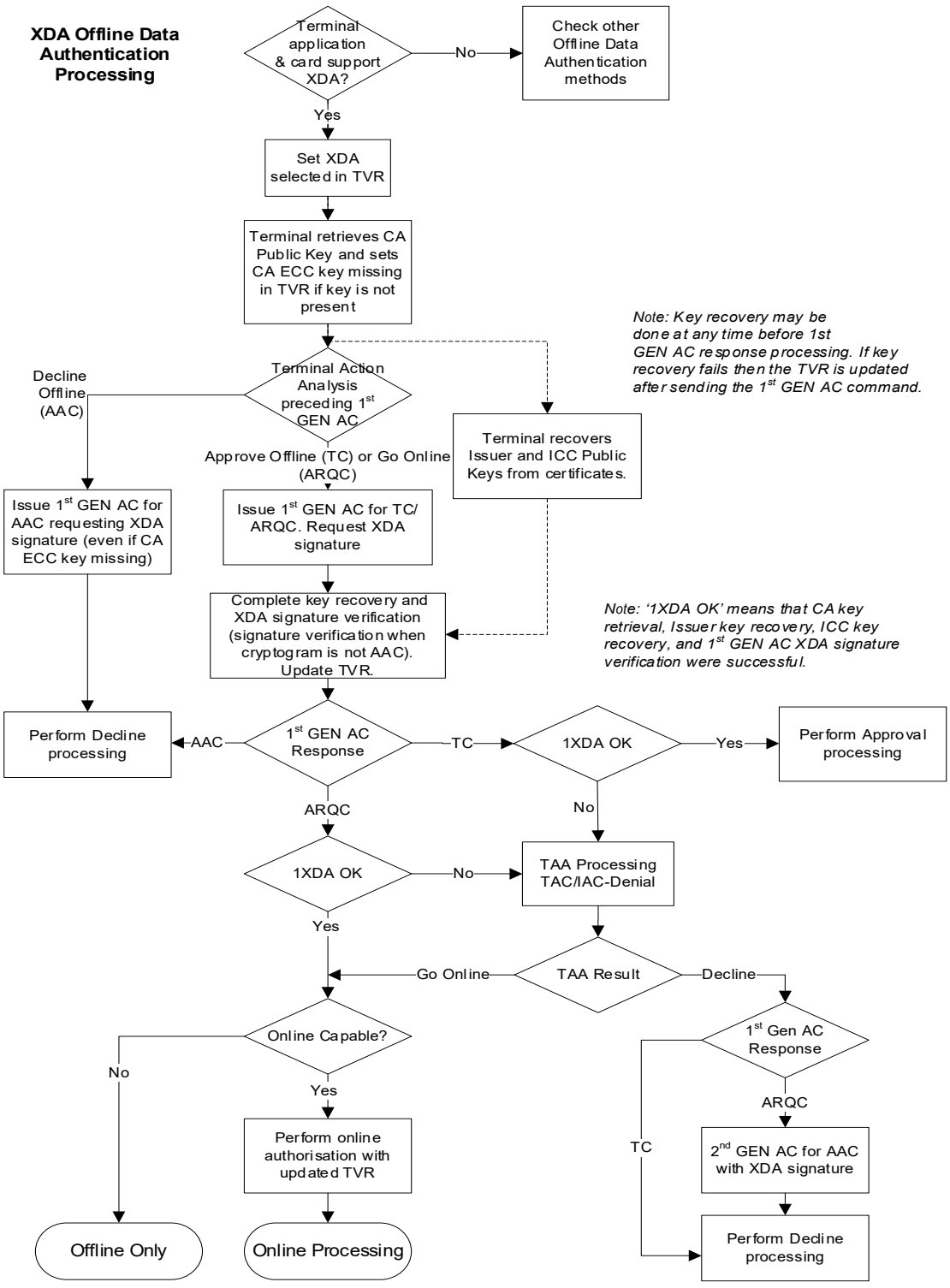


Figure 12: XDA Sample Flow Part 1 of 2

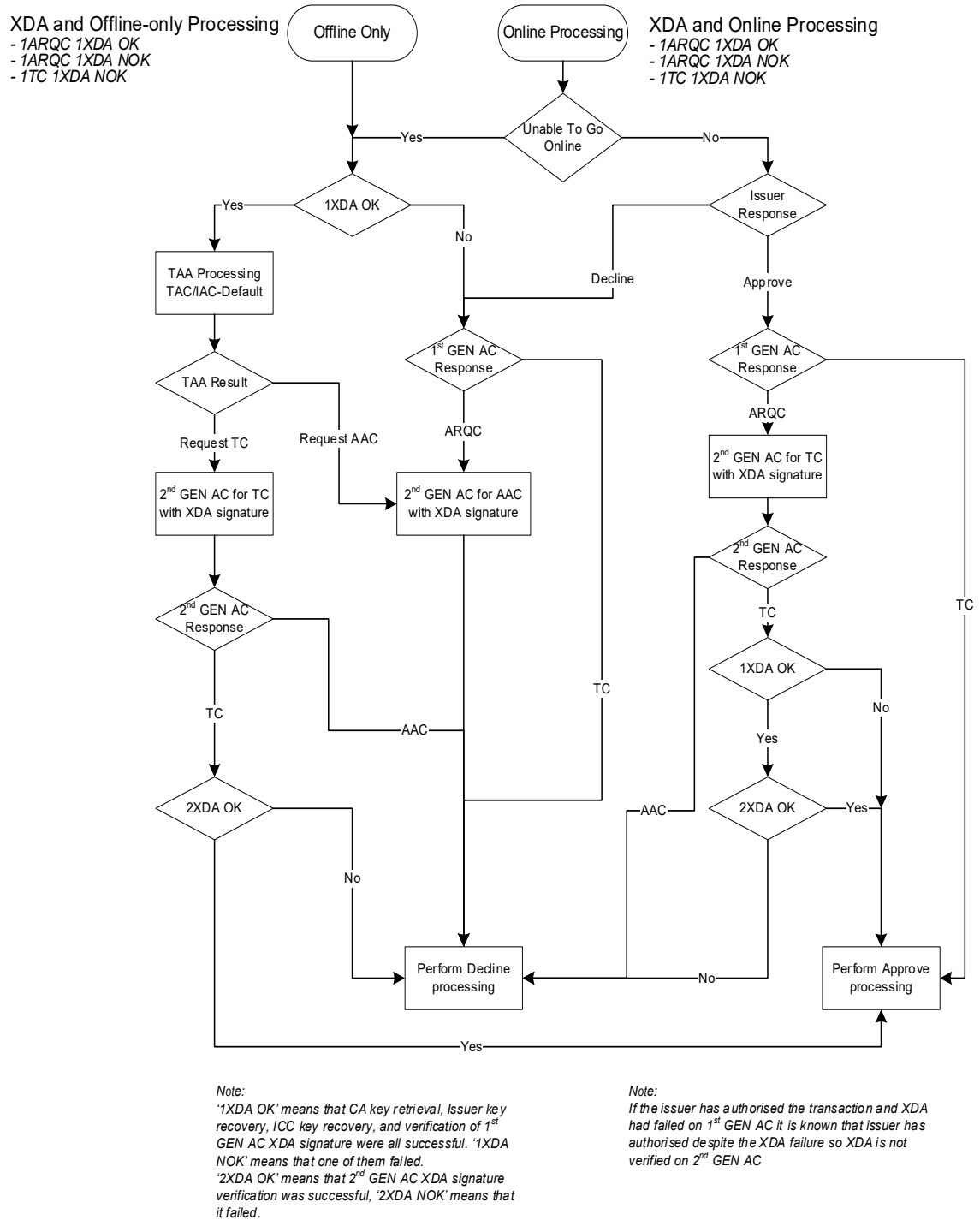


Figure 13: XDA Sample Flow Part 2 of 2

13 Offline Data Encipherment Using ECC

This section relates to the use of ECC for offline encipherment of data, in particular PINs and biometrics, to ensure the secure transfer of data from the CVM capture device to the ICC.

If supported, the ICC shall own a public/private key pair associated with offline data encipherment. The public key is then used by the CVM capture device or a secure component of the terminal (other than the CVM capture device) to derive symmetric keys using an approach based on ISO/IEC 11770-6. These are then used to encipher the input data in accordance with ISO/IEC 19772 Mechanism 5 (Encrypt-then-MAC).

If a PIN is to be enciphered then it is first formed into a PIN block as shown in Book 3 section 6.5.12.

Note: It is not possible to mix RSA and ECC during a transaction as the card can only return one Certification Authority Public Key Index and one Issuer Public Key Certificate. Thus, it is not possible to perform ODE using ECC together with ODA using RSA, nor is it possible to perform ODE using RSA with ODA using the ECC based XDA.

13.1 Keys and Certificates

If offline data encipherment using ECC is supported by the ICC as indicated by the CVM List, then the ICC has a key pair consisting of a public encipherment key and the corresponding private decipherment key. This specification allows the ICC to use a dedicated ICC public key pair for ECC ODE, or the same ICC public key pair for both XDA and ECC ODE. However, in both cases the ICC will have a dedicated ICC Public Key Certificate for ODE tag '9F2D' containing the ICC Public Key and ICC Public Key Algorithm Suite Indicator (see Table 36) which is authenticated and recovered using the same process as described in section 12.4.[44]

[44] ICC Public Key Certificates enforce key usage by including the appropriate Algorithm Suite Indicator.

13.2 PIN Encipherment

The terminal shall not encipher the PIN (step 3 below) until it has authenticated the ICC public key used for ECC ODE (sections 12.2, 12.3, and 12.4).

The exchange of an enciphered PIN between terminal and ICC shall take place in the following steps.

1. The PIN is entered in plaintext format on the PIN pad and a PIN block is constructed as defined in Book 3 section 6.5.12.
2. The terminal issues a GET CHALLENGE command to the ICC to obtain an 8-byte unpredictable number UN_{ODE} from the ICC. When the response to the GET CHALLENGE command is anything other than an 8 byte data value with SW1 SW2 = '9000', then the terminal shall consider that the Offline Enciphered PIN (ECC) CVM has failed.

   Note: A fresh GET CHALLENGE command is issued irrespective of previous CVM processing that may also have used a GET CHALLENGE command.

   Steps 1 and 2 may be performed in either order.
3. If Key Derivation has already been performed for the current transaction, then do not perform Key Derivation again and use the previously derived keys.

   If Key Derivation has not yet been performed for the current transaction, then:

   a) Generate an ephemeral key pair (d, R) as described in B2.2.4, for the curve identified by the certificate's ICC Public Key Algorithm Suite Indicator and set R_x to be the x-coordinate of R converted to a byte-string of length N_{FIELD} using I2OS() per B2.6.1.

   b) Perform Key Derivation as described in A2.3.2, with the UN_{ODE} generated in step 2 as the UN for key derivation UN_{KD}, the certificate's ICC Public Key Algorithm Suite Indicator for the Algorithm Suite Indicator, d from step 3a), and the ICC Public Key for ODE recovered from the ICC Public Key Certificate for ODE (tag '9F2D' as specified in section 13.1) as the point Q, to derive keys and initialise the counter N.
4. Apply the ECC encryption function (see A2.3.3) indicated by the certificate's ICC Public Key Algorithm Suite Indicator (see B2.4), with K_1[45] from step 3b) as K_X, K_2 from step 3b) as K_Y, and A set to null, to the data specified in Table 40 as MSG in order to obtain the ciphertext C. If encryption fails, then the terminal shall consider that the Offline Enciphered PIN (ECC) CVM has failed.
5. Concatenate R_x and C to obtain the Enciphered Data

   Enciphered Data := R_x || C

[45] The derivation function generates four AES keys, but this specification only uses K_1 and K_2.

Field Name	Length	Description	Format
Data Header	1	Hex Value '7F'	b
PIN Block	8	PIN in PIN Block	b
ICC Unpredictable Number (UN_{ODE})	8	Unpredictable number obtained from the ICC with the GET CHALLENGE command	b

Table 40: Data To Be Enciphered for PIN Encipherment (ECC)

6. The terminal issues a VERIFY command including the Enciphered Data obtained in the previous step.

13.3 PIN Decipherment and Verification

The decipherment and verification of an enciphered PIN by the ICC shall take place in the following steps.

1. If the length of the Enciphered Data received in the VERIFY command is less than N_{FIELD} of the curve plus the length of a MAC (both as determined by the Algorithm Suite used by the ICC), the ICC shall return SW1 SW2 = '6984' and Offline Enciphered PIN (ECC) CVM has failed.
2. The ICC extracts R_x from the Enciphered Data received in the VERIFY command and calculates an ECC public key Q using the Point4x() function as described in section B2.2.1(e).

   Note: The length of R_x is N_{FIELD} for the curve determined by the Algorithm Suite used by the ICC. For P-256 the length of R_x is 32 bytes. For P-521 the length of R_x is 66 bytes.

   The remainder of the Enciphered Data, after R_x has been extracted, is C.
3. The ICC derives two 128-bit AES keys K_1 and K_2 using the key derivation process described in A2.3.2, with the UN_{ODE} as the UN for key derivation UN_{KD}, the ICC Public Key Algorithm Suite Indicator, the ICC private key for ECC ODE, and Q generated above.
4. To recover the plaintext data specified in Table 40, the ICC applies the ECC decryption function (see A2.3.4), with K_1 and K_2 derived in step 3 above as K_X and K_Y respectively, C extracted in step 2 above, and A as null.
5. If this decryption function fails, the ICC shall return SW1 SW2 = '6984' to indicate that Offline Enciphered PIN (ECC) CVM has failed.
6. The ICC verifies whether the ICC Unpredictable Number recovered in step 4 is equal to the ICC Unpredictable Number (UN_{ODE}) generated by the ICC with the GET CHALLENGE command. If this is not the case, the ICC shall return SW1 SW2 = '6984' to indicate that Offline Enciphered PIN (ECC) CVM has failed.
7. The ICC verifies whether the Data Header recovered in step 4 is equal to '7F'. If this is not the case, the ICC shall return SW1 SW2 = '6984' to indicate that Offline Enciphered PIN (ECC) CVM has failed.
8. The ICC verifies whether the PIN included in the PIN Block recovered in step 4 corresponds with the PIN stored in the ICC. If this is not the case, PIN verification has failed.

The order of steps 6, 7, and 8 is mandatory.

If all the above steps were executed successfully, then enciphered PIN verification was successful.

The terminal behaviour on receipt of the VERIFY response is described in Book 3 section 10.5.1.

13.4 Biometric Data Encipherment

The encipherment and exchange of biometric data between terminal and ICC takes place in the following steps, as illustrated in Book 3 Figure 13.

1. The biometric template is captured on a biometric capture device and the Biometric Data Block (BDB) is constructed by the Biometric Processing Application.
2. The terminal issues a GET CHALLENGE command to the ICC to obtain an 8-byte unpredictable number UN_{ODE} from the ICC. When the response to the GET CHALLENGE command is anything other than an 8 byte data value with SW1 SW2 = '9000', then the terminal considers that the Offline Biometric CVM has failed.

   Note: A fresh GET CHALLENGE command is issued irrespective of previous CVM processing that may also have used a GET CHALLENGE command.

   Steps 1 and 2 may be performed in either order.
3. If Key Derivation has already been performed for the current transaction, then do not perform Key Derivation again and use the previously derived keys.

   If Key Derivation has not yet been performed for the current transaction, then:

   a) Generate an ephemeral key pair (d, R) as described in B2.2.4, for the curve identified by the certificate's ICC Public Key Algorithm Suite Indicator and set R_x to be the x-coordinate of R converted to a byte-string of length N_{FIELD} using I2OS() per B2.6.1.

   b) Perform Key Derivation as described in A2.3.2, with the UN_{ODE} generated in step 2 as the UN for key derivation UN_{KD}, the certificate's ICC Public Key Algorithm Suite Indicator for the Algorithm Suite Indicator, d from step 3a), and the ICC Public Key for ODE recovered from the ICC Public Key Certificate for ODE (tag '9F2D' as specified in section 13.1) as the point Q, to derive keys and initialise the counter N.
4. Apply the ECC encryption function (see A2.3.3) indicated by the certificate's ICC Public Key Algorithm Suite Indicator (see B2.4), with K_1[46] from step 3b) as K_X, K_2 from step 3b) as K_Y, and A set to null, to the data specified in Table 41 as MSG in order to obtain the ciphertext C. If encryption fails, then the terminal considers that the Offline Biometric CVM has failed.
5. Concatenate R_x and C to obtain the Enciphered Data

   Enciphered Data := R_x || C

[46] The derivation function generates four AES keys, but this specification only uses K_1 and K_2.

Field Name	Length	Description	Format
ICC Unpredictable Number (UN$_{ODE}$)	8	Unpredictable number obtained from the ICC with the GET CHALLENGE command	b
Biometric Solution ID Length	1	Length of Biometric Solution ID	b
Biometric Solution ID	value of Biometric Solution ID Length	As defined in Book 3 Table 48	b
Biometric Type Length	1	Length of Biometric Type data element	b
Biometric Type	value of Biometric Type Length	As defined in Book 3 Table 49	b
Biometric Subtype	1	As defined in Book 3 Table 50	b
Biometric Data Block (BDB) Length	2	Length of BDB data element	b
Biometric Data Block (BDB)	value of BDB Length	A block of data with a specific format that contains information captured from a biometric capture device	b

Table 41: Data To Be Enciphered for Biometric Encipherment

Note: The length bytes in Table 41 are simple binary bytes, and are *not* coded as BER-TLV length fields.

6. The terminal shall construct the Biometric Verification Data Template, and order the data elements as indicated in Table 42.
   a) The value of Biometric Type is set as the plaintext value used in step 4 when referencing Table 41.
   b) The value of Biometric Solution ID is set as the plaintext value used in step 4 when referencing Table 41.
   c) The value of Enciphered Biometric Data is set as the value of the Enciphered Data generated in step 5.

Tag	Length	Field Name
'81'	var.	Biometric Type
'90'	var.	Biometric Solution ID
'DF51'	var.	Enciphered Biometric Data

Table 42: Biometric Verification Data Template

Note: The data objects in Table 42 are BER-TLV coded. The length of all TLV coded data objects are coded on the minimum number of bytes (that is, on 1 byte if < 128, on 2 bytes if in the range 128 to 255, and so on). See Book 3 section B.2 for BER-TLV coding rules. There shall be no '00' padding bytes before, between, or after the BER-TLV coded data objects.

7. The terminal shall construct and issue VERIFY command(s) as following:
   a) If the length of the value field of the Biometric Verification Data Template is no more than 255 bytes, the terminal shall set the value field of the Biometric Verification Data Template as the data field of the single VERIFY command.
   b) If the length of the value field of the Biometric Verification Data Template is more than 255 bytes, the terminal shall divide the value field of the Biometric Verification Data Template into 255 byte data blocks. The last data block may be 1 to 255 bytes in length. Then the terminal shall set these data blocks as the data fields of the commands and chain the commands using command chaining defined in Book 3 section 6.5.13.
   c) The terminal shall issue the VERIFY command(s).

13.5 Biometric Data Decipherment and Recovery

The decipherment and recovery of biometric data by the ICC takes place in the following steps.

1. If the length of the Enciphered Data received in the VERIFY command(s) is less than N_{FIELD} of the curve plus the length of a MAC (both as determined by the Algorithm Suite used by the ICC), the ICC shall return SW1 SW2 '6984' and Offline Biometric CVM has failed.
2. The ICC extracts R_x from the Enciphered Data received in the Verify command, and calculates an ECC public key Q using Point4x() function as described in section B2.2.1(e).

   Note: The length of R_x is N_{FIELD} for the curve determined by the Algorithm Suite used by the ICC. For P-256 the length of R_x is 32 bytes. For P-521 the length of R_x is 66 bytes.

   The remainder of the Enciphered Data, after R_x has been extracted, is C.
3. The ICC derives two 128-bit AES keys K_1 and K_2 using the key derivation process described in A2.3.2, with the UN_{ODE} as the UN for key derivation UN_{KD}, the ICC Public Key Algorithm Suite Indicator, the ICC private key for ECC ODE, and Q generated above.
4. To recover the plaintext data specified in Table 41, the ICC applies the ECC decryption function (see A2.3.4), with K_1 and K_2 derived in step 3 above as K_X and K_Y respectively, C extracted in step 2 above, and A as null.
5. If this decryption function fails, the ICC shall return SW1 SW2 = '6984' to indicate that Offline Biometric CVM has failed.
6. The ICC verifies whether the ICC Unpredictable Number recovered in step 4 is equal to the ICC Unpredictable Number (UN_{ODE}) generated by the ICC with the GET CHALLENGE command. If they are not the same, Offline Biometric CVM has failed.
7. The ICC verifies whether the Biometric Solution ID recovered in step 4 is equal to the Biometric Solution ID (tag '90') included in the Biometric Verification Data Template. If they are not the same, Offline Biometric CVM has failed.
8. The ICC verifies whether the Biometric Type recovered in step 4 is equal to the Biometric Type (tag '81') included in the Biometric Verification Data Template. If they are not the same, Offline Biometric CVM has failed.

If all the above steps were executed successfully, then enciphered biometric recovery was successful.

Part III

Annexes

Annex A Security Mechanisms

A1 Symmetric Mechanisms

A1.1 Encipherment

Encipherment of data uses an *n*-byte block cipher ALG either in Electronic Codebook (ECB) Mode or in Cipher Block Chaining (CBC) mode according to ISO/IEC 10116.

Encipherment of a message MSG of arbitrary length with Encipherment Session Key K_S takes place in the following steps.

1. Padding and Blocking

   - If the message MSG has a length that is not a multiple of *n* bytes, add one '80' byte to the right of MSG, and then add the smallest number of '00' bytes to the right such that the length of resulting message
     MSG := (MSG || '80' || '00' || '00' || . . . || '00') is a multiple of *n* bytes.
   - If the message MSG has a length that is a multiple of *n* bytes, the following two cases can occur depending on pre-defined rules.
     - No padding takes place: MSG := MSG.
     - MSG is padded to the right with the *n*-byte block

       ('80' || '00' || '00' || … || '00' || '00' || '00' || '00')

       to obtain MSG.

   MSG is then divided into *n*-byte blocks $X_1, X_2, \ldots, X_B$.

2. Cryptogram Computation

   ECB Mode

   Encipher the blocks $X_1, X_2, \ldots, X_B$ into the *n*-byte blocks $Y_1, Y_2, \ldots, Y_B$ with the block cipher algorithm in ECB mode using the Encipherment Session Key K_S. Hence compute for i = 1, 2, . . . , B:

   $$Y_i := ALG(K_S)[X_i]$$

   CBC Mode

   Encipher the blocks $X_1, X_2, \ldots, X_B$ into the *n*-byte blocks $Y_1, Y_2, \ldots, Y_B$ with the block cipher algorithm in CBC mode using the Encipherment Session Key K_S. Hence compute for i = 1, 2, . . . , B:

   $$Y_i := ALG(K_S)[X_i \oplus Y_{i-1}]$$

   with *n*-byte initial value

Y_0 := ('00' || '00' || '00' || … || '00' || '00' || '00' || '00').

Notation:

$$Y := (Y_1 || Y_2 || \ldots || Y_B) = ENC(K_S)[MSG]$$

Decipherment is as follows.

1. Cryptogram Decipherment

   ECB Mode

   Compute for i = 1, 2, . . . , B:

   $$X_i := ALG^{-1}(K_S)[Y_i]$$

   CBC Mode

   Compute for i = 1, 2, . . . , B:

   $$X_i := ALG^{-1}(K_S)[Y_i] \oplus Y_{i-1}$$

   with *n*-byte initial value

   Y_0 := ('00' || '00' || '00' || … || '00' || '00' || '00' || '00').

2. To obtain the original message MSG, concatenate the blocks $X_1, X_2, \ldots, X_B$ and if padding has been used (see above) remove the trailing

   ('80' || '00' || '00' || . . . || '00') byte-string from the last block X_B.

   Notation:

   $$MSG = DEC(K_S)[Y]$$

A1.2 Message Authentication Code

A1.2.1 MAC Algorithms Using an 8-byte Block Cipher

The computation of an s-byte MAC ($4 \le s \le 8$) is according to ISO/IEC 9797-1 using an 8-byte (64-bit) block cipher algorithm (ALG) such as Triple DES in CBC mode. More precisely, the computation of a MAC S over a message MSG consisting of an arbitrary number of bytes with a MAC Session Key K_S takes place in the following steps.

1. Padding and Blocking

   Pad the message MSG according to ISO/IEC 7816-4 (which is equivalent to method 2 of ISO/IEC 9797-1); hence add a mandatory '80' byte to the right of MSG, and then add the smallest number of '00' bytes to the right such that the length of resulting message

   MSG := (MSG || '80' || '00' || '00' || . . . || '00') is a multiple of 8 bytes.

   MSG is then divided into 8-byte blocks $X_1, X_2, \ldots, X_B$.

2. MAC Session Key

   The MAC Session Key K_S consists of either only a leftmost key block $K_S = K_{SL}$ or the concatenation of a leftmost and a rightmost key block $K_S = (K_{SL} \,||\, K_{SR})$.

3. Cryptogram Computation

   Process the 8-byte blocks $X_1, X_2, \ldots, X_B$ with the block cipher in CBC mode using the leftmost MAC Session Key block K_{SL}:

   $$H_i := ALG(K_{SL})[X_i \oplus H_{i-1}], \text{ for } i = 1, 2, \ldots, B$$

   with initial value

   $H_0 :=$ ('00' || '00' || '00' || '00' || '00' || '00' || '00' || '00').[47]

   Compute the 8 byte block H_{B+1} in one of the following two ways.

   - According to ISO/IEC 9797-1 Algorithm 1:

     $$H_{B+1} := H_B$$

   - According to ISO/IEC 9797-1 Algorithm 3:

     $$H_{B+1} := ALG(K_{SL})[ALG^{-1}(K_{SR})[H_B]]$$

   The MAC S is then equal to the s most significant bytes of H_{B+1}.

A1.2.2 MAC Algorithms Using a 16-byte Block Cipher

The computation of an *s*-byte MAC ($4 \leq s \leq 8$) is according to ISO/IEC 9797-1 Algorithm 5 (CMAC) using a 128-bit (16-byte) block cipher algorithm (ALG) such as AES in CBC mode. More precisely, the computation of a MAC S over a message MSG consisting of an arbitrary number of bytes with a MAC Session Key K_S takes place in the following steps.

1. Padding and Blocking

   If the length of the message MSG is a multiple of 16 bytes then no padding is added.[48]

   If the length of the message MSG is not a multiple of 16 bytes then add a mandatory '80' padding byte to the right of MSG, and then add the smallest number of '00' bytes to the right such that the length of resulting message

   MSG := (MSG || '80' || '00' || '00' || . . . || '00') is a multiple of 16 bytes.

   MSG is then divided into 16-byte blocks $X_1, X_2, \ldots, X_B$.

[47] Note that pre-pending the MSG with the previous MAC (8 bytes) as a chaining block (see section 9.2.3) is equivalent to using an initial value equal to the previous MAC processed by Triple DES (Algorithm 1) or Single DES (Algorithm 3).

[48] Note that this will always be the case with Format 1 Secure Messaging as the message is padded prior to MACing.

2. MAC Session Key

   The MAC Session Key K_S consists of a *k*-bit block (see section B1).

   In accordance with ISO/IEC 9797-1 Algorithm 5 (CMAC), two 128-bit sub-keys K_1 and K_2 are also derived.

   Let Z be 16 bytes set to zero and let C be equal to Z except with its least significant bits set to 10000111b (C is a CMAC-defined constant for a 16-byte block cipher).

   $L := ALG(K_S)[Z]$

   $K_1 = L << 1$. If $msb(L) = 1$ then $K_1 := K_1 \oplus C$

   $K_2 = K_1 << 1$. If $msb(K_1) = 1$ then $K_2 := K_2 \oplus C$

   where the following notation is used:

   "<< 1" means left-shift 1 bit (i.e. the new 128-bit string comprises the 127 rightmost bits of the old string, appended with a zero bit).

   "msb" means the most significant bit (i.e. the leftmost bit).

   Note that all intermediate values must be kept secret (e.g. not used by the card as an unpredictable number).

3. Cryptogram Computation

   Mask the final block with sub-key K_1 if no padding was added

   $X_B := X_B \oplus K_1$

   and with sub-key K_2 if padding was added

   $X_B := X_B \oplus K_2$.

   Process the 16-byte blocks $X_1, X_2, \ldots, X_B$ with the block cipher in CBC mode using the MAC Session Key K_S:

   $H_i := ALG(K_S)[X_i \oplus H_{i-1}]$, for i = 1, 2, . . . , B

   with 16-byte initial value

   H_0 := ('00' || '00' || ... || '00' || '00' || '00' || '00' || '00').[49]

   The MAC S is then equal to the *s* most significant bytes of H_B.

[49] Note that pre-pending the MSG with the previous MAC (16 bytes) as a chaining block (see section 9.2.3) is equivalent to using an initial value equal to the previous MAC

A1.3 Session Key Derivation

Session keys K_S are derived from unique Master Keys K_M using diversification data R as follows

$$K_S := F(K_M)[R]$$

To prevent replay attacks, the diversification data R should have a high probability of being different for each session key derivation.

An important requirement for the diversification function F is that the number of possible outputs of the function is sufficiently large and uniformly distributed to prevent an exhaustive key search on the session key.

Section A1.3.1 specifies a method for the derivation of session keys for Application Cryptogram generation, issuer authentication, and secure messaging (see sections 8 and 9) from ICC Master Keys. This session key derivation method ensures that different transactions use different session keys.

Note that the session key derivation method provided in section A1.3.1 is not mandatory. Issuers may decide to adopt another method for this function.

A1.3.1 Common Session Key Derivation Option

The common session key derivation option generates a unique session key for each transaction performed by the application. It does this by enciphering an n-byte diversification value with the k-bit ICC Master Key (MK) to produce a k-bit ICC Session Key (SK) using an n-byte block cipher algorithm ALG in ECB mode.

The n-byte diversification value is represented as

$$R = R_0 \,||\, R_1 \,||\, R_2 \,||\, \ldots \,||\, R_{n-1}.$$

For the session key used to generate and verify the Application Cryptogram and the ARPC, the diversification value is the ATC followed by n-2 bytes of '00':

$$R := ATC \,||\, '00' \,||\, '00' \,||\, \ldots \,||\, '00' \,||\, '00' \,||\, '00'.$$

For the session keys used for secure messaging, the diversification value R is the Application Cryptogram returned in the response to the first GENERATE AC command followed by n-8 bytes of '00':

$$R := \text{Application Cryptogram} \ldots \,||\, '00' \,||\, '00' \,||\, '00'.$$

For an n-byte block cipher ALG using a k-bit key where $k = 8n$ (AES with k=128) the derivation function F is computed as follows:

$$SK := ALG\,(MK)\,[\,R\,].$$

For an n-byte block cipher ALG using a k-bit key where $16n \geq k > 8n$ (Triple DES with k=128 or AES with k=192 or 256) R is used to create two n-byte blocks as follows:

F1 = R_0 || R_1 || 'F0' || … || R_{n-1}.

F2 = R_0 || R_1 || '0F' || … || R_{n-1}.

and

SK := Leftmost k-bits of {ALG (MK) [F1] || ALG (MK) [F2] }.

The same session key is used for all commands in a single transaction.

A1.4 Master Key Derivation

This section specifies three optional methods for the derivation by the issuer of a k-bit ICC Master Key used for Application Cryptogram generation, issuer authentication, and secure messaging. The first two methods use the 8-byte block cipher Triple DES in ECB mode and the third method uses the 16-byte block cipher AES in ECB mode. The AES method supports 128-bit, 192-bit, and 256-bit keys.

Note that none of these methods are mandatory. Issuers may decide to adopt alternative methods for this function.

These methods take as input the PAN and PAN Sequence Number, plus a k-bit Issuer Master Key IMK, and produce the k-bit ICC Master Key MK in the following way:

A1.4.1 Option A

Option A is only applicable when the block cipher is Triple DES.

1. Concatenate from left to right the decimal digits of the Application PAN with the PAN Sequence Number (if the PAN Sequence Number is not present, then it is replaced by a '00' byte). If the result X is less than 16 digits long, pad it to the left with hexadecimal zeros in order to obtain an 8-byte number Y in numeric format. If X is at least 16 digits long, then Y consists of the 16 rightmost digits of X in numeric format.
2. Compute the two 8-byte numbers

   Z_L := DES3(IMK)[Y]

   and

   Z_R := DES3(IMK)[Y ⊕ ('FF'||'FF'||'FF'||'FF'||'FF'||'FF'||'FF'||'FF')]

   and define

   Z := (Z_L || Z_R)

The 128-bit ICC Master Key MK is then equal to Z, with the exception of the least significant bit of each byte of Z which is set to a value that ensures that each of the 16 bytes of MK has an odd number of nonzero bits (this to conform with the odd parity requirements for DES keys).

A1.4.2 Option B

Option B is only applicable when the block cipher is Triple DES.

If the Application PAN is equal to or less than 16 decimal digits, use Option A. If the Application PAN is greater than 16 decimal digits, do the following:

1. Concatenate from left to right the decimal digits of the Application PAN and the PAN Sequence Number (if the PAN Sequence Number is not present, it is replaced by a '00' byte). If the Application PAN has an odd number of decimal digits then concatenate a '0' padding digit to the left thereby ensuring that the result is an even number of digits.
2. Hash the result of the concatenation using the SHA-1 hashing algorithm to obtain the 20-byte hash result X.
3. Select the first 16 decimal digits (0 to 9) starting from the left side of the 20-byte (40-nibble) hash result X and use as the value Y. If this does not provide for 16 decimal digits in Y, convert the non-decimal nibbles in X to decimal digits by means of the following decimalisation table:

Input nibble of X	A	B	C	D	E	F
Decimalised nibble	0	1	2	3	4	5

Figure 14: Decimalisation for Master Key Derivation

   Add the converted digits starting from the left side of X to the end of Y until Y contains 16 digits.

   Example 1: Hash result X contains 16 or more decimal digits

   X = '12 30 AB CD 56 78 42 D4 B1 79 F2 CA 34 5D 67 89 A1 7B 64 BB'

   Y = first 16 decimal digits of X = '12 30 56 78 42 41 79 23'

   Example 2: Hash result X contains less than 16 decimal digits

   X = '1B 3C AB CD D6 E8 FA D4 B1 CD F2 CA D4 FD C7 8F A1 7B 6E BB'

   Y = decimal digits from X = '13 68 41 24 78 17 6' plus the required number of converted digits '1 20' (from 'B', 'C', and 'A'), giving:

   Y = '13 68 41 24 78 17 61 20'

4. Continue with the processing specified for Option A starting at step 2.

A1.4.3 Option C

Option C is only applicable when the n-byte block cipher is AES.

Concatenate from left to right the decimal digits of the Application PAN with the PAN Sequence Number (if the PAN Sequence Number is not present, then it is replaced by a '00' byte). Pad it to the left with hexadecimal zeros in order to obtain a 16-byte number Y in numeric format.

The k-bit ICC Master Key MK is then equal to

- In the case $k = 8n$:

  MK := AES(IMK)[Y]

- In the case $16n \geq k > 8n$:

  MK := Leftmost k–bits of {AES(IMK)[Y] || AES(IMK)[Y*]}

  where Y* = Y ⊕ ('FF'||'FF'||'FF'||'FF'||'FF'||'FF'||'FF'||'FF'
  ||'FF'||'FF'||'FF'||'FF'||'FF'||'FF'||'FF'||'FF')

  In other words Y* is a bit-wise inversion of Y.

A2 Asymmetric Mechanisms

A2.1 Digital Signature Scheme Using RSA

This section describes the special case of the RSA digital signature scheme giving message recovery using a hash function according to ISO/IEC 9796-2, which is used in this specification for offline static and dynamic data authentication.

A2.1.1 Algorithms

The digital signature scheme uses the following two types of algorithms.

- A reversible asymmetric algorithm consisting of a signing function $Sign(S_K)[\ ]$ depending on a Private Key S_K, and a recovery function $Recover(P_K)[\ ]$ depending on a Public Key P_K. Both functions map N-byte numbers onto N-byte numbers and have the property that

  $$Recover(P_K)[Sign(S_K)[X]] = X$$

  for any N-byte number X.
- A hashing algorithm Hash[ ] that maps a message of arbitrary length onto an 20-byte hash code.

A2.1.2 Signature Generation

The computation of a signature S on a message MSG consisting of an arbitrary number L of at least N – 21 bytes takes place in the following way.

1. Compute the 20-byte hash value H := Hash[MSG] of the message M.
2. Split MSG into two parts MSG = (MSG_1 || MSG_2), where MSG_1 consists of the N – 22 leftmost (most significant bytes) of MSG and MSG_2 of the remaining (least significant) L – N + 22 bytes of MSG.
3. Define the byte B := '6A'.
4. Define the byte E := 'BC'.
5. Define the N-byte block X as the concatenation of the blocks B, MSG_1, H, and E, hence:

   $$X := (B \,||\, MSG_1 \,||\, H \,||\, E)$$
6. The digital signature S is then defined as the N-byte number

   $$S := Sign(S_K)[X]$$

A2.1.3 Signature Verification

The corresponding signature verification takes place in the following way:

1. Check whether the digital signature S consists of N bytes.
2. Retrieve the N-byte number X from the digital signature S:

$$X = \text{Recover}(P_K)[S]$$

3. Partition X as X= (B || MSG_1 || H || E), where:
   - B is one byte long
   - H is 20 bytes long
   - E is one byte long
   - MSG_1 consists of the remaining N – 22 bytes
4. Check whether the byte B is equal to '6A'.
5. Check whether the byte E is equal to 'BC'.
6. Compute MSG = (MSG_1 || MSG_2) and check whether H = Hash[MSG].

If and only if these checks are correct is the message accepted as genuine.

A2.2 Digital Signature Scheme Using ECC

This section describes the ECC version of the Schnorr digital signature scheme with appendix using a hash algorithm according to ISO/IEC 14888-3. It is the optimised version where only the x-coordinate is hashed rather than the x and y coordinates.

A2.2.1 Introduction

The signature generation and verification functions specified in A2.2 are for Signature Algorithm Suites '10' and '11' (see B2.4.1).

The Elliptic Curve Schnorr Digital Signature Algorithm (EC-SDSA) consists of a signature generation function and a signature verification function as described below. These functions make use of:

- The selected ECC curves specified in section B2.2.2.
- A hash algorithm Hash[ ] that maps a message of arbitrary length onto an N_{HASH}-byte hash code, where N_{HASH} is a fixed value for the given hash algorithm. For selected hash algorithms, see section B2.3.
- The combinations of ECC curves and hash algorithms are specified in section B2.4 and are determined by the Public Key Algorithm Suite Indicator associated with the key pair that generates the signature.

A2.2.2 Signature Generation

For EC-SDSA, the computation of a signature S on a message MSG of arbitrary length using a signer's private key d on an elliptic curve E over F_p with base point G of order n (note that for the curves defined in section B2.2.2, $n < p$) shall be as follows.

Input:

- Algorithm Suite Indicator identifying the curve E, group order n, and hash function Hash[ ] (output length N_{HASH}).
- d, the private signing key of the ICC.
- MSG (length $L \geq 0$), the message to be signed.

Output:

- $S := (r, s)$, the signature (a byte-string of length $N_{HASH} + N_{FIELD}$) on the message MSG.

Process:

a) Select a statistically unique and unpredictable integer k where $0 < k < n$. (This may be a random or a strong pseudo-random generation process and may use the string-to-integer techniques used in section B2.6.)

b) Compute $kG = (x_1, y_1)$ on the curve E. Convert the x_1 coordinate to a byte string B_1 of N_{FIELD} bytes according to the integer conversion function I2OS() in section B2.6.

c) Compute $r = \text{Hash}[\ B_1\ ||\ \text{MSG}\ ]$.

d) Compute $s' = (k + r'd) \bmod n$, where r' denotes r converted to an integer according to the integer conversion function OS2I() in section B2.6 then reduced mod n.

e) Convert the integer s' into a byte string s of N_{FIELD} bytes according to the integer conversion function I2OS() in section B2.6.

f) Output $S := (r, s)$.

That is, the signature S consists of a concatenated pair comprising the N_{HASH} byte string r and an integer formatted as the N_{FIELD} byte string s. The length in bytes of the Digital Signature is $N_{HASH} + N_{FIELD}$, where N_{HASH} is the output length of the hash algorithm in bytes and N_{FIELD} is the length of p (the field size) in bytes.

Note that the above Output is very slightly different to ISO/IEC 14888-3 since according to ISO/IEC 14888-3 the second element s of the signature S is simply an integer (and no encoding as a string of N_{FIELD} bytes is specified).

A2.2.3 Signature Verification

The function described in this section is used when verifying ECC signatures associated with Issuer Public Key Certificates, ICC Public Key Certificates and card-generated XDA signatures. The function uses curve information associated with the algorithm suite identified in the Certification Authority Public Key Related Data, the Issuer Public Key Certificate and the ICC Public Key Certificate, respectively.

The verification of an EC-SDSA signature S on a message MSG using the signer's public key point P on an elliptic curve E over F_p with base point G of order n (note that with the curves defined in section B2.2.2, $n < p$) and using a hash algorithm Hash[ ] shall be as follows.

Input:

- Algorithm Suite Indicator identifying the curve E, n, and Hash[ ] (output length N_{HASH}).
- S, a byte-string that is the (asserted) signature on the message MSG.
- P (or x-coordinate thereof), public key point of signer.
- MSG (length $L \geq 0$).

Output:

- Success or failure.

Process:

Preliminary processing: If only the x-coordinate of the signer's public key is available then the public key point P is first calculated using the Point4x() function as described in section B2.2.1(e).

Verification steps:

a) If the length in bytes of S is not $N_{HASH} + N_{FIELD}$, signature verification fails.

b) Parse S into the N_{HASH} byte string r and the N_{FIELD} byte string s. Thus $S = (r, s)$.

c) Convert s to an integer s' according to the integer conversion function OS2I() in section B2.6.

d) Convert r to a non-negative integer r' less than n by using OS2I() followed by reduction modulo n.

e) Check that $0 < s' < n$ and $0 < r'$. If either is not true then signature verification fails.

f) Compute $(x_2, y_2) = s'G - r'P$.

g) Convert the x-coordinate x_2 to a byte string B_2 of N_{FIELD} bytes according to the integer conversion function I2OS() in section B2.6.

h) Compute v = Hash[B_2 || MSG].

i) If $v = r$, then signature verification is successful; otherwise signature verification fails.

N_{HASH} is the output length of the hash algorithm in bytes and N_{FIELD} is the length of p (the field size) in bytes.

A2.3 Encryption Scheme Using ECC

A2.3.1 Introduction

The encryption and decryption functions specified in A2.3 are for ODE Algorithm Suites '00' and '01' (see B2.4.2).

Key derivation is only performed when encryption and decryption are first invoked during a transaction.

Key derivation initialises a global counter N, copies of which are maintained by the terminal and by the ICC.

Note: A single counter is used even if messages are exchanged in both directions because only a single message will be processed at a time and therefore synchronisation problems will not occur. If parallel processing were to be introduced in the future then two counters would be needed, one for each direction.

Note: Although the functions in A2.3 are specified in terms of the curves P-256 and P-521 and AES, the functions can be adapted to other curves and block ciphers.

A2.3.2 Key Derivation

Input:

- UN_{KD}, an 8-byte unpredictable number used for key derivation.
- Algorithm Suite Indicator identifying the curve, key agreement, and derivation method.
- d, a private ECC key.
- Q, a public key point on the curve.

Global variable:

- N, a 2-byte counter.

Output:

- K_1, a symmetric key used for encrypting/decrypting data sent from terminal to ICC.
- K_2, a symmetric key used for authenticating data sent from terminal to ICC.
- K_3, a symmetric key used for encrypting/decrypting data sent from ICC to terminal.
- K_4, a symmetric key used for authenticating data sent from ICC to terminal.

Process:

a) Set NumSKs : = 4.

b) Using the function PointMultiply() defined in B2.2.1(d), compute Z as the x-coordinate of the point dQ.

c) Convert Z from an integer to a bit string according to the integer conversion function I2BS() in section B2.6.

   - For P-256, Z is converted to a 256-bit string.
   - For P-521, Z is converted to a 640-bit string; the rightmost 521 bits of Z are a bit string representation of the x-coordinate of the point dQ and so the leftmost 119 bits of Z are zero 'padding bits'.

d) For P-256, write Z as $Z_0 \,||\, Z_1$ where Z_i is a 128-bit string for i = 0,1. This can be denoted as $Z_0 \,||\, Z_{t-1}$ with t = 2.

e) For P-521, write Z as $Z_0 \,||\, Z_1 \,||\, Z_2 \,||\, Z_3 \,||\, Z_4$ where Z_i is a 128-bit string for i = 0,1,2,3,4. This can be denoted as $Z_0 \,||\, Z_1 \,||\, Z_2 \,||\, Z_3 \,||\, Z_{t-1}$ with t = 5.

f) Apply CMAC (see section A1.2.2) to Z to produce an AES key derivation K_{DK} key as follows, using the 128-bit zero string (0^{128}) as the CMAC key:

$C_0 := AES(0^{128})[Z_0]$

$Z_{t-1} := Z_{t-1} \oplus$ 'CDD297A9DF1458771099F4B39468565C'
where the constant is according to CMAC and calculated as $AES(0^{128})[0^{128}] << 1$

for i from 1 to t-1, let $C_i := AES(0^{128})[C_{i-1} \oplus Z_i]$

$K_{DK} := C_{t-1}$

g) Derive K_i for i = 1 to NumSKs:

$K_i := AES(K_{DK})$['0i' || DerData]

where for this specification DerData is the 15 bytes SKD_VERSION || '00' || T || '02' || '00' where

- SKD_VERSION is the 1-byte '01'
- T is the 11 byte UN_{KD} || 'A5A5A5'
- The final 2 bytes represent the length of the output (512 bits in the case of 128-bit session keys with NumSKs equal to 4)

h) Set the 2-byte variable *N* to the value '0000'. This is a global variable a copy of which is maintained by terminal and ICC. The terminal increments *N* after each successful encryption (see section A2.3.3) and the ICC increments *N* after each successful decryption (see section A2.3.4).

A2.3.3 Encryption and Authentication

This section describes both encryption of data and authentication of data. Authentication includes both the encrypted data, if present, and any additional unencrypted data. This allows the function to be used to authenticate data even if there is no data to encrypt.

Input:

- Algorithm Suite Indicator identifying the Authenticated Encryption method and Block Cipher.
- K_X, a symmetric key used for encrypting data.
- K_Y, a symmetric key used for authenticating data.
- MSG, a plaintext message.
- *A*, Additional Authentication Data.

Global variable:

- *N*, a 2-byte counter.

Output:

- *C*, the ciphertext (encrypted message) or an indication of failure.

Process:

The encryption process uses two derived keys because the encryption process is actually 'authenticated encryption' using a method called Encrypt-then-MAC (Mechanism 5 in ISO/IEC 19772).

The authenticated encryption mechanism described in this specification supports Additional Authenticated Data (AAD, data that is authenticated but not encrypted).

Encryption of a plaintext MSG, denoted P, takes place in the following steps:

a) If $N = 65535$ then encryption has failed, so abort and report failure.

b) Generate the Starting Variable SV as the 16-byte all-zero string whose leftmost 2 bytes are set to the value of N.

c) If the message payload P is null then set C^* to be null and skip to step i).

d) If the bit-length of the message payload P is not a multiple of 8, then Encryption has failed, so abort and report failure.

e) If P is not a multiple of 16 bytes, then pad[50] by appending r ($0 < r < 16$) zero-bytes to P.

f) Partition P into B 16-byte blocks:

$$P := P_1 \,||\, P_2 \,||\, \ldots \,||\, P_B$$

g) Encipher the message payload

$$C_i = P_i \oplus \text{AES}(K_X)[\, SV + ((i\text{-}1) \bmod 2^{112}) \,] \text{ for } 1 <= i <= B$$

h) If padding had been applied then truncate the final rightmost r bytes from C_B thereby creating C_B', otherwise set $C_B' = C_B$. Set:

$$C^* := C_1 \,||\, C_2 \,||\, \ldots \,||\, C_B'$$

i) If the bit-length of A is not a multiple of 8, then encryption has failed, so abort and report failure.

j) Set:

$$D := \text{I2BS}(\text{len}(A)/8\ ,\ 64) \,||\, A$$

k) Generate an 8-byte MAC using K_Y in accordance with section A1.2.2 over $D \,||\, SV \,||\, C^*$ and append this to C^* to create the ciphertext C

$$C := C^* \,||\, \text{MAC}(D \,||\, SV \,||\, C^*).$$

l) Increment N.

m) Return the ciphertext C.

[50] This padding is only specified to simplify the specification in the case that the plaintext is not a multiple of 16 bytes and is not functionally necessary as any padding bytes added to the plaintext are truncated from the ciphertext before calculation of the MAC.

No result other than an indication of failure shall be returned in case of failure.

A2.3.4 Authentication and Decryption

This section describes both authentication of data and decryption of data. Authentication includes both encrypted data, if present, and any additional unencrypted data. This allows the function to be used to authenticate data even if there is no data to decrypt.

Input:

- Algorithm Suite Indicator identifying the Authenticated Encryption method and Block Cipher.
- K_X, a symmetric key used for decrypting data.
- K_Y, a symmetric key used for authenticating data.
- C, ciphertext of a message.
- A, Additional Authentication Data.

Global variable:

- N, a 2-byte counter.

Output:

- MSG, a plaintext message or an indication of failure.

Process:

The authenticated decryption mechanism described in this specification supports Additional Authenticated Data (data that is authenticated but not encrypted).

Decryption of the ciphertext C, takes place in the following steps:

a) If $N = 65535$ then decryption has failed, so abort and report failure.

b) If the length of the ciphertext C is less than the size of a MAC (8 bytes) then decryption has failed, so abort and report failure.

c) If the bit-length of A is not a multiple of 8, then decryption has failed, so abort and report failure.

d) Set:

$$D := \text{I2BS}(\text{len}(A)/8 , 64) \;||\; A$$

e) Partition the ciphertext C into C^* and an 8-byte candidate MAC value S'.

$$C := C^* \;||\; S'$$

f) The bit-length of C^* should be a multiple of 8 and will be zero in the case that C^* is null (corresponding to a null plaintext). If the bit-length of C^* is not a multiple of 8, then decryption has failed, so abort and report failure.

g) Generate the Starting Variable SV as the 16-byte all-zero string whose leftmost 2 bytes are set to the value of N.

h) Calculate the 8-byte MAC S using K_Y in accordance with section A1.2.2 over $D \mid\mid SV \mid\mid C^*$.

i) If $S \neq S'$ then decryption has failed, so abort and report failure.

j) If C^* is null then the plaintext MSG is null so skip to step p).

k) If C^* is not a multiple of 16 bytes, then pad[51] by appending r ($0 < r < 16$) zero-bytes to C^*.

l) Partition C^* into B 16-byte blocks:

$$C^* := C_1 \mid\mid C_2 \mid\mid \ldots \mid\mid C_B$$

m) Decrypt the ciphertext as

$$P_i = C_i \oplus \text{AES}(K_X)[\, SV + ((i\text{-}1) \bmod 2^{112}) \,] \text{ for } 1 <= i <= B$$

n) If padding had been applied then truncate the final rightmost r bytes from P_B

o) The plaintext MSG is the byte-string

$$P := P_1 \mid\mid P_2 \mid\mid \ldots \mid\mid P_B$$

p) Increment N.

q) Return the plaintext message MSG.

No result other than an indication of failure shall be returned in case of failure.

[51] This padding is only specified to simplify the specification in the case that the ciphertext is not a multiple of 16 bytes and is not functionally necessary as any padding bytes added to the ciphertext are truncated from the plaintext before output.

Annex B Approved Cryptographic Algorithms

B1 Symmetric Algorithms

These specifications use block cipher algorithms with key size k and block size n. For various reasons the key size is expressed in bits and the block size in bytes. Note that the mechanisms specified in Annex A only support block cipher algorithms satisfying $16n \geq k \geq 8n$.

B1.1 Data Encryption Standard (DES) 8-byte block cipher

The double-length key triple DES encipherment algorithm (see ISO/IEC 18033-3) is the approved cryptographic algorithm to be used in the encipherment and MAC mechanisms specified in section A1 where an 8-byte block cipher is used. The algorithm is based on the (single) DES algorithm standardised in ISO 16609.

Triple DES encipherment involves enciphering an 8-byte plaintext block in an 8-byte ciphertext block with a double-length (128-bit) secret key $K = (K_L \mid\mid K_R)$ as follows:

$$Y = DES3(K)[X] = DES(K_L)[DES^{-1}(K_R)[DES(K_L)[X]]]$$

Decipherment takes place as follows:

$$X = DES^{-1}(K_L)[DES(K_R)[DES^{-1}(K_L)[Y]]]$$

Single DES is only approved for usage with the version of the MAC mechanism specified in section A1 using Algorithm 3 of ISO/IEC 9797-1 (Triple DES applied to the last block).

B1.2 Advanced Encryption Standard (AES) 16-byte block cipher

The AES encipherment algorithm (see ISO/IEC 18033-3) is the approved cryptographic algorithm to be used in the encipherment and MAC mechanisms specified in section A1 where a 16-byte block cipher is used. 128-bit, 192-bit, and 256-bit key length versions of AES are supported, but the key length should be fixed for a given implementation.

B2 Asymmetric Algorithms

B2.1 RSA Algorithm

This reversible algorithm (see Annex C reference 2) is the approved algorithm for encipherment and digital signature generation as described in section A2. The only values allowed for the public key exponent are 3 and $2^{16} + 1$.

The algorithm produces a cryptogram or digital signature whose length equals the size of the modulus used. The upper bounds for the size of the modulus are specified in Table 43.

Description	Max. Length
Certification Authority Public Key Modulus	248 bytes
Issuer Public Key Modulus	247 bytes
Issuer Public Key Modulus (SDA only)	248 bytes
ICC Public Key Modulus	247 bytes
ICC PIN Encipherment Public Key Modulus	247 bytes

Table 43: Upper Bounds for Size of Moduli

The maximum length for issuer and ICC keys is reduced because a 2-byte tag is used for the ICC certificates and ICC keys cannot be longer than issuer keys. For SDA cards the maximum length for issuer keys is 248 bytes because a 1-byte tag is used for the Signed Static Application Data.

Additional restrictions on the lengths of moduli are described in section D1.

In the choice of the lengths of the public key moduli, one should take into account the lifetime of the keys compared to the expected progress in factoring during that lifetime. The ranges (upper and lower bounds) for the key lengths mandated by each of the payment systems are specified in their corresponding proprietary specifications. Further guidance is also provided in the EMV Issuer and Application Security Guidelines.

The value of the Issuer Public Key Exponent and the ICC Public Key Exponent is determined by the issuer. The Certification Authority, Issuer, and ICC Public Key Exponents shall be equal to 3 or $2^{16} + 1$.

The Public Key Algorithm Indicator for this digital signature algorithm shall be coded as hexadecimal '01'.

The keys and signing and recovery functions for the RSA algorithm with odd-numbered public key exponent are specified below.

B2.1.1 Keys

The private key S_K of the RSA digital signature scheme with an odd-numbered public key exponent e consists of two prime numbers p and q such that p – 1 and q – 1 are co-prime to e and a private exponent d such that:

$$ed \equiv 1 \bmod (p - 1)(q - 1)$$

The corresponding public key P_K consists of the public key modulus n = pq and the public key exponent e.

B2.1.2 Signing Function

The signing function for RSA with an odd-numbered public key exponent e is defined as:

$$S = Sign(S_K)[X] := X^d \bmod n,\ 0 < X < n$$

where X is the data to be signed and S the corresponding digital signature.

B2.1.3 Recovery Function

The recovery function for RSA with an odd-numbered public key exponent e is equal to:

$$X = Recover(P_K)[S] := S^e \bmod n$$

B2.1.4 Key Generation

Payment systems and issuers shall be responsible for the security of their respective RSA public/private key generation processes. Examples of secure key generation methods can be found in Annex C reference 1.

B2.2 Elliptic Curve Cryptography (ECC)

B2.2.1 Introduction

This section describes elliptic curve cryptography (including algorithms and keys) to be used in conjunction with these specifications.

These specifications define two curves, P-256 and P-521 (see B2.2.2). P-521 is supported for contingency purposes only.

Elliptic curve cryptography is based on arithmetic on abstract mathematical objects called *elliptic curves*. An elliptic curve is built on (or *defined over*) a *field*. In the scope of this specification, this is always the finite field F_p for a prime p.

(a) The Finite Field Fp

The field F_p can be described as the set {0, 1, … , p–1} of non-negative integers smaller than p, with arithmetic *modulo* p.

Consider two field elements x and y in F_p. Their sum, difference, product, division, and inverse in F_p are defined in terms of regular integer arithmetic as follows.

Addition: The sum $x + y$ in F_p is:

$$(x + y) \bmod p$$

Subtraction: The difference $x – y$ in F_p is:

$$(x – y) \bmod p$$

Multiplication: The product $x \cdot y$ in F_p is:

$$(x \cdot y) \bmod p$$

Thus, addition, subtraction, and multiplication may be computed as usual and the result is then reduced modulo p.

Division: x / y in F_p (with $y \neq 0$) is defined in terms of an inverse:

$$(x \cdot y^{-1}) \bmod p$$

Inverse: The inverse of a non-zero y in F_p is the unique non-zero field element z (i.e. integer in {1, 2, …, p–1}) that satisfies:

$$(y \cdot z) \bmod p = 1$$

The value of z is best computed using the Extended Euclidean Algorithm for computation of the greatest common divisor (gcd). When computing the gcd of two integers a and b, it returns the gcd of a and b as well as two integers u and v that satisfy:

$$a \cdot u + b \cdot v = \gcd(a, b)$$

Because p is prime, $\gcd(p, y) = 1$, so when computing this gcd, the Extended Euclidean Algorithm returns u and v that satisfy:

$$p \cdot u + y \cdot v = 1$$

Therefore, $y \cdot v \bmod p = 1$, and so $y^{-1} = v \bmod p$.

Note that the algorithms defined in this specification do not use division or inverse.

(b) Elliptic Curves and Point Representation

An elliptic curve over F_p is a collection of points – pairs (x, y) of field elements in the underlying field that satisfy a given equation. For the selected curves, this *curve equation* has the form $y^2 = x^3 + ax + b$ which is computed in F_p. For all selected curves, $a = -3$.

To fully define an elliptic curve E over F_p, one must specify:

- The value of p.
- The curve equation (specified by the parameters $a = -3$ and b).
- A base point G (also called the *generator*) on the curve.
- The *order* n of G. For all supported curves, n is prime, and equals the number of points on the curve. (This implies that the so-called *cofactor* equals 1.)

For more details, see ISO/IEC 15946-1. The standards IEEE P1363 and SEC 1 and their references provide alternative descriptions and implementation hints that may be useful.

For transmission and processing, elliptic curve points and specifically their integer coordinates will need to be interpreted as (converted to) byte strings. See section B2.6.

This specification represents points such as public keys simply as the x-coordinate of the point. Note that with this representation a value of x can actually represent two points:

$$P = (x, y) \text{ and } -P = (x, -y).$$

(c) Point Addition

Input for adding two points:

- Algorithm Suite Indicator identifying the curve.
- Points (x_1, y_1) and (x_2, y_2) on the curve.

Output:

- Point (x_3, y_3) on the curve equal to the sum $(x_1, y_1) + (x_2, y_2)$.

The points on an elliptic curve together with one special extra point **0**, called the point at infinity, form a group, where the group operation is traditionally called addition, or point addition. That is, two points on an elliptic curve P and Q can be "added", and the result, denoted as $P + Q$, is a point on the same curve.

Point addition for selected curves in this specification is defined as follows.

- For any point P on the curve (including $P = \mathbf{0}$):

$$P + \mathbf{0} = \mathbf{0} + P = P$$

- For any point (x, y) on the curve:[52]

$$(x, y) + (x, -y) = \mathbf{0}$$

- In other words:

$$-(x, y) = (x, -y) \text{ and } (x, 0) + (x, 0) = \mathbf{0}$$

- For any two points (x_1, y_1) and (x_2, y_2) with different x-coordinates, the sum $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ which has coordinates:

$$x_3 = \lambda^2 - x_1 - x_2 \bmod p$$

$$y_3 = \lambda(x_1 - x_3) - y_1 \bmod p$$

with:

$$\lambda = (y_2 - y_1) \cdot (x_2 - x_1)^{-1} \bmod p$$

- For any point (x, y) with $y \neq 0$, the sum $(x, y) + (x, y) = (x_3, y_3)$ which has coordinates:

$$x_3 = \lambda^2 - 2x \bmod p$$

$$y_3 = \lambda(x - x_3) - y \bmod p$$

with:

$$\lambda = (3x^2 - 3) \cdot (2y)^{-1} \bmod p$$

(d) Point Multiplication – PointMultiply()

Input for multiplying a point by a positive integer:

- Algorithm Suite Indicator identifying the curve.
- Positive integer k.
- Point P on the curve.

Output:

- Point kP on the curve.

Point multiplication (also called scalar multiplication) is repeated addition of an elliptic curve point to itself: for a positive integer k, $kP = P + P + \cdots + P$ (with k copies of P). For $k = 0$, the result is the point at infinity: $0P = \mathbf{0}$.

As a result of the group structure of the set of points on the curve, there is an integer n such that $nP = \mathbf{0}$ for all P on the curve. The number n is called the order of the curve group. As a consequence, for all points P and all integers k, $kP = (k \bmod n)P$.

[52] By the curve equation, this implies that $(x, -y)$ is also on the curve.

A point multiplication kP can be computed efficiently using techniques analogous to efficient modular exponentiation. Note that especially efficient algorithms are known if the curve point is known in advance, as this allows for the use of precomputed intermediate results. These data are often called "public key multiples". In general, any fixed-base method can be used.

(e) Point Finding – Point4x()

Input for finding a point on a prime curve with given x-coordinate:

- Algorithm Suite Indicator identifying the curve.
- Positive integer $x < p$, where p is the characteristic of the curve field.

Output:

- Point *(x,y)* on the curve with least value y.

In order to find a point P on the curve that has a given x-coordinate x, it is necessary to find an integer y such that $y^2 = x^3 + ax + b \bmod p$. Note that if there is a non-zero solution y to this equation, and thus a point (x, y) on the curve, then there will be exactly two non-zero solutions y and $-y$, and thus two points P and $-P$ on the curve with this x-coordinate x. The routine Point4x() assumes that $p = 3 \bmod 4$ (which is the case for all the curves defined in this specification) and returns a point (x, y) where y can be computed as follows:

$$y' = \text{modsqrt}(x^3 + ax + b, p) = (x^3 + ax + b)^{((p+1)/4)} \bmod p$$

$$y = \min(y', p-y')$$

Under some attack conditions this function may return a point that is not on the curve. Therefore, card implementations need to take appropriate steps to ensure that the point returned is valid. Such attacks are less relevant to the Diffie-Hellman and the certificate verification functions in the Kernel.

Note: When the recovered point is to be used for Diffie-Hellman key agreement rather than signature verification either of the two possible y values may be used and therefore the $y = \min(y', p-y')$ step may be omitted and y' can be used directly.

B2.2.2 Selected Elliptic Curves

This section defines the selected curves and their representation. The curves are selected from ISO/IEC 15946-5. Other curves may be used for domestic use (see Table 48) but the details are outside the scope of this specification.

The selected curves are listed in the following Table 44.

Curve	N_{FIELD} (bytes)	Public Key Length (bytes) (= N_{FIELD})	Curve Type
P-256	32	32	Pseudo-random over F_p
P-521	66	66	Pseudo-random over F_p

Table 44: Selected Elliptic Curves

An elliptic curve Public Key is a point on the curve and in this specification a Public Key is represented by the x-coordinate of the point.

In the choice of curves one should take into account the lifetime of the keys compared to the expected progress in ECC cryptanalysis. The minimum and maximum strengths of curves mandated by each of the payment systems are specified in their corresponding proprietary specifications.

Below, each of the supported curves is specified in detail.

(a) Curve P-256

The curve P-256 is defined over F_p for a 256-bit prime p, with curve equation $y^2 = x^3 - 3x + b$ (i.e. $a = -3$). It has the following parameters. The prime p and (integer) order n are given in both decimal and hexadecimal form; field elements (the parameter b and the base point coordinates) are given in hexadecimal only.

Parameter	Value
p =	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$ = 115 79208 92103 56248 76269 74469 49407 57353 00861 43415 29031 41955 33631 30886 70978 53951 FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF
n =	115 79208 92103 56248 76269 74469 49407 57352 99969 55224 13576 03424 22259 06106 85120 44369 FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551
b =	5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6 3BCE3C3E 27D2604B
G_x =	6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C296
G_y =	4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5

Table 45: Parameters of Curve P-256

(b) Curve P-521

The curve P-521 is defined over F_p where p is the 521-bit Mersenne prime, with curve equation $y^2 = x^3 - 3x + b$ (i.e. $a = -3$). It has the following parameters. The prime p and (integer) order n are given in both decimal and hexadecimal form; field elements (the parameter b and the base point coordinates) are given in hexadecimal only.

Parameter	Value
p =	$2^{521} - 1$ = 68 64797 66013 06097 14981 90079 90813 93217 26943 53001 43305 40939 44634 59185 54318 33976 56052 12255 96406 61454 55497 72963 11391 48085 80371 21987 99971 66438 12574 02829 11150 57151 01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
n =	68 64797 66013 06097 14981 90079 90813 93217 26943 53001 43305 40939 44634 59185 54318 33976 55394 24505 77463 33217 19753 29639 96371 36332 11138 64768 61244 03803 40372 80889 27070 05449 01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFA 51868783 BF2F966B 7FCC0148 F709A5D0 3BB5C9B8 899C47AE BB6FB71E 91386409
b =	51 953EB961 8E1C9A1F 929A21A0 B68540EE A2DA725B 99B315F3 B8B48991 8EF109E1 56193951 EC7E937B 1652C0BD 3BB1BF07 3573DF88 3D2C34F1 EF451FD4 6B503F00
G_x =	C6 858E06B7 0404E9CD 9E3ECB66 2395B442 9C648139 053FB521 F828AF60 6B4D3DBA A14B5E77 EFE75928 FE1DC127 A2FFA8DE 3348B3C1 856A429B F97E7E31 C2E5BD66
G_y =	0118 39296A78 9A3BC004 5C8A5FB4 2C7D1BD9 98F54449 579B4468 17AFBD17 273E662C 97EE7299 5EF42640 C550B901 3FAD0761 353C7086 A272C240 88BE9476 9FD16650

Table 46: Parameters of Curve P-521

B2.2.3 Required Support for Elliptic Curves

EMV kernels supporting XDA or ECC ODE shall support ECC P-256 and P-521. Support of other curves is outside the scope of EMV although Algorithm Suite Indicators may be allocated (see section B2.4). P-521 is supported for contingency purposes only.

B2.2.4 Key Generation

Input for key generation:

- Algorithm Suite Indicator identifying the curve.

Output:

- An integer d and a point $P = dG$ on the curve.

This section describes the generation of ECC key pairs with special attention to long term Payment System, Issuer and ICC key pairs. The method of elliptic curve key pair generation follows ISO/IEC 15946-1.

The private key d for a given elliptic curve E with base point G of order n consists of a positive integer d, satisfying $1 < d < n\text{-}1$.

The corresponding public key point is $P = dG$ on the curve E.

Given an elliptic curve, all parameters including the base point are fixed. Therefore, key generation consists solely of random or pseudorandom generation of d and subsequent computation of the public key point $P = dG$ and thereby its x-coordinate representation.

This specification represents public keys using only their x-coordinate. For any non-zero point (x,y) on the curve, $(x,-y)$ will also be on the curve. The function in this specification that is sensitive to the value of the y-coordinate is signature verification and specifically Issuer Public Key Certificate verification using the Certification Authority Public Key, ICC Public Key Certificate verification using the Issuer Public Key, and ICC XDA signature verification using the ICC Public Key. For this reason the Payment System key, the Issuer key and the ICC key must be generated to ensure that the verifier can determine the correct value of the y-coordinate of the Public Key from knowledge of only the x-coordinate.

For this reason this specification requires that Payment System, Issuer and ICC keys are generated to ensure that the y-coordinate as an integer modulo p is less than $(p+1)/2$. The function Point4x() defined in B2.2.1(e) will then generate the correct y-coordinate given the correct x-coordinate.

Two examples of how to generate such Payment System, Issuer and ICC keys are provided below. The first approach (1) will take longer, but the second approach (2) may require special HSM functionality.

(1) Repeatedly generate a random non-negative integer d satisfying $1 < d < n\text{-}1$ – for example, d := 2+RandomInteger(n-3) – until the y-coordinate of dG is less than $(p+1)/2$.

(2) Generate a random non-negative integer d' satisfying $1 < d' < n\text{-}1$ for example d' := 2+RandomInteger(n-3), and if the y-coordinate of $d'G$ is less than $(p+1)/2$ then $d := d'$ else set $d := n - d'$.

The public key is then calculated to be the point $P = dG$ and its representation is the x-coordinate of P. Note that this construction of the Payment System key, Issuer key and ICC key reduces the key space of the private key by one bit.

ECC encryption requires that the terminal generate an ephemeral key pair. As this key is used for Diffie-Hellman key agreement rather than signature, the y-coordinate does not require the checks stipulated above. Thus, the terminal ephemeral private key d can be generated without the range checks on y:

> Generate a random non-negative integer d satisfying $1 < d < n-1$ for example $d := 2+\text{RandomInteger}(n-3)$.

For security reasons, a strong pseudo-random or random number generator is required (see section B2.5).

B2.3 Hash Algorithms (for ECC)

Hash Algorithm Indicators are allocated by EMVCo. EMVCo has partitioned the set of values: Those in the range '00' – '7F' are specified by EMV and subject to type approval, whereas those in the range '80' – 'FF' are for proprietary use (e.g. regional/domestic systems) – implementation specifications and testing are out of the scope of EMVCo.

Values indicated as "assigned" are for future proofing. Whilst EMV testing and type approval are not currently available, implementers should be aware that support for the indicated algorithms may be introduced in the future.

Table 47 below lists hash algorithms and their corresponding algorithm indicators. This indicator is used in the ICC Public Key Certificate to identify the hash algorithm to be used when calculating the ICCD Hash.

Hash Algorithm Indicator	Hash Algorithm	N_{HASH}	Block Size
'01'	SHA-1 (not used with ECC; see section B3)	n/a	n/a
'02'	SHA-256	32	64
'03'	SHA-512	64	128
'04' (assigned)	SHA-3 256	32	136
'05' (assigned)	SHA-3 512	64	72
'80'	SM3	32	64

Table 47: Recognised Hash Algorithms

The output length N_{HASH} of the hash algorithm refers to the number of bytes in the untruncated output of the hash algorithm.

All these hash algorithms are standardised in ISO/IEC 10118-3 and can hash an arbitrarily large input[53] and implementations shall be designed to support this.

[53] Actually SHA-256 and SM3 have a limit of 2^{64} bits and SHA-512 has a limit of 2^{128} bits.

A string of bytes that is input to a hash algorithm is processed in blocks and the size of these blocks, referred to as Block Size in Table 47, is a characteristic of the hash algorithm. Table 47 only includes this information because it can be relevant to implementations that need to process large input strings.

SHA-512 is included for contingency purposes only. Due to length restrictions in the ICC Public Key Certificate and ICC Public Key Certificate for ODE, SHA-512 cannot be used for ICCD Hash if the Issuer Public Key Certificate Algorithm Suite is '11' and the ICC Public Key Algorithm Suite is '11' for signature or '01' for encryption.

B2.4 Cryptographic Algorithm Suites (for ECC)

This section lists the Public Key Algorithm Suites and associated Indicators used in this specification for ECC-based mechanisms.

Algorithm Suite Indicators are allocated by EMVCo. EMVCo has partitioned the set of values: Those in the range '00' – '7F' are specified by EMV and subject to type approval, whereas those in the range '80' – 'FF' are for proprietary use (e.g. regional/domestic systems) – implementation specifications and testing are out of the scope of EMVCo.

Values indicated as "assigned" are for future proofing. Whilst EMV testing and type approval are not currently available, implementers should be aware that support for the indicated algorithms and key lengths may be introduced in the future.

B2.4.1 Cryptographic Algorithm Suites for ECC Signatures

Table 48 below applies to the algorithms used for verifying ECC signatures hence for verifying public key certificates and XDA signatures.

Signature Algorithm Suite Indicator	Signature Mechanism	Hash Algorithm	ECC Curve	N_{FIELD} / N_{SIG}
'10'	EC-SDSA	SHA-256	P-256	32 / 64
'11'	EC-SDSA	SHA-512	P-521	66 / 130
'12' (assigned)	EC-SDSA	SHA-3 256	P-256	32 / 64
'13' (assigned)	EC-SDSA	SHA-3 512	P-521	66 / 130
'80'	SM2-DSA	SM3	SM2-P256	32 / 64

Table 48: Cryptographic Algorithm Suite Indicators for ECC Signatures

The Signature Mechanism EC-SDSA is described in section A2.2.

For details of the hash algorithms for use with the Signature Algorithm Suites see section B2.3.

P-521 is included for contingency purposes only.

B2.4.2 Cryptographic Algorithm Suites for ECC Encryption

Table 49 below applies to the algorithms used for ECC encryption.

ODE Algorithm Suite Indicator	ECC Curve	N_{FIELD}	Key Agreement and Derivation	Authenticated Encryption	Block Cipher
'00'	P-256	32	DH 16 byte keys	EtM	AES
'01'	P-521	66	DH 16 byte keys	EtM	AES
'88'	SM2-P256	32	DH 16 byte keys	EtM	SM4

Table 49: Cryptographic Algorithm Suite Indicators for Offline Data Encipherment (e.g. for PIN or Biometrics)

The encryption methods indicated above are described in section A2.3.

Note that the Chinese curve SM2-P256 and block cipher SM4 are included for domestic suites.

Section A2.3 specifies:

- Diffie-Hellman (DH) key agreement and key derivation
- Encrypt-then-MAC (EtM) authenticated encryption and uses Counter Mode encryption and CMAC with Additional Authenticated Data

AES and SM4 are standardised in ISO/IEC 18033-3.

P-521 is included for contingency purposes only.

B2.5 Random Number Generation (for ECC)

This section defines functions and requirements for generating 'random numbers' especially for ECC key generation in section B2.2.4.

The function RandomInteger() takes as input a bounding integer m and returns a random non-negative integer less than m, i.e. in the range $[0,\ldots,m\text{-}1]$. One way to achieve this is as follows:

1. $M := \text{len}(m)$, the bit-length of m
2. $d := \text{BS2I}(\text{RandomBits}(M))$
3. If $(d > m - 1)$, then go to step 2.
4. Return d.

The function RandomBits() takes an integer M as parameter and returns a string $X = <X_{M-1}, \ldots, X_0>$ of M random bits. Techniques and requirements for generating random bit strings of a given length by both deterministic and non-deterministic methods are provided in ISO/IEC 18031.

ISO/IEC 18031 classifies random bit generators as either non-deterministic (where the primary entropy source can, over time, provide unlimited entropy) or deterministic (where the primary entropy source has limited entropy such as a seed value).

Note: An algorithm for generating the Terminal Unpredictable Number (tag '9F37') is included in section 11.

B2.6 Integer Conversion Functions (for ECC)

There are mathematical functions used in this specification (e.g. signature generation) that involve the processing of integers which are represented as bits and bytes. This section therefore specifies how strings of bits and bytes are to be interpreted as integers.

Note that because the elliptic curves identified in this specification operate over finite fields with a prime number p of elements, each element of the field is naturally interpreted as a non-negative integer less than p. and points are then represented as pairs of such integers.

B2.6.1 Integers and Bits and Bytes

For the purposes of this section the conversion between integers and bytes is mediated by conversion to strings of bits. The conversion functions defined in this section are consistent with those used in the standards of ISO/IEC JTC1 SC27 (see for example ISO/IEC 14888-3 Annex B) and use the term Octet instead of Byte and thereby distinguish OS (Octet String) from BS (Bit String).

BS2I() and I2BS()

Input for BS2I():

- A bit string x of length $l > 0$.

Output:

- A non-negative integer value v.

The function BS2I(x) maps a bit string x of length l to a non-negative integer value v, as follows. If $x = <x_{l-1}, \ldots, x_0>$ where $x_0, \ldots, x_{l-1}$ are bits, then the value v is defined as

$$v = \sum_{k=0}^{l-1} x_k 2^k$$

For the empty string, v is equal to 0.

- Example: BS2I(000 0010 1010 1100 0001) = 10945

The function BS2I() is invoked from the function RandomInteger() (section B2.5) used by Key Generation (section B2.2.4).

Input for I2BS():

- Two non-negative integers v and l.

Output:

- A bit string x of length l.

The function I2BS(v,l) which takes as input two non-negative integers v and l, and outputs the unique bit string x of length l such that BS2I(x) = v, if such an x exists. If no such x exists then the function outputs an error message.

- Example 1: I2BS(10945, 19) = 000 0010 1010 1100 0001
- Example 2: I2BS(10945, 14) = 10 1010 1100 0001

The function I2BS() is invoked from Key Derivation (section A2.3.2), Encryption and Authentication (section A2.3.3), and Authentication and Decryption (section A2.3.4).

OS2BS() and BS2OS()

Input for OS2BS():

- An octet string x.

Output:

- A bit string y.

The function OS2BS(x) takes as input an octet string x, interprets it as a bit string y and outputs the bit string y.

- Example: OS2BS('00' '2A' 'C1') = 0000 0000 0010 1010 1100 0001

Input for BS2OS():

- A bit string y.

Output:

- An octet string x.

The function BS2OS(y) takes as input a bit string y, whose length is a multiple of 8, and outputs the unique octet string x such that y = OS2BS(x).

- Example: BS2OS(0000 0010 1010 1100 0001)

= <0000 0000> <0010 1010> <1100 0001>

= '00' '2A' 'C1'

OS2I() and I2OS()

Input for OS2I():

- An octet string x.

Output:

- An integer v.

The function OS2I(x) takes as input an octet string x and outputs the integer v = BS2I(OS2BS(x)).

- Example: OS2I(<0010 1010> <1100 0001>) = OS2I('2A' 'C1') = 10945

The function OS2I() is invoked from PIN Encipherment (section 13.2), Biometric Data Encipherment (section 13.4), Signature Generation (section A2.2.2), and Signature Verification (section A2.2.3).

Input for I2OS():

- two non-negative integers v and l.

Output:

- An octet string x of length l in octets.

The function I2OS(v,l) takes as input two non-negative integers v and l, and outputs the unique octet string x of length l in octets such that OS2I(x) = v, if such an x exists. Otherwise, the function outputs an error message.

- Example 1: I2OS(10945, 3) = <0000 0000> <0010 1010> <1100 0001>

  = '00' '2A' 'C1'

- Example 2: I2OS(10945, 2) = <0010 1010> <1100 0001>

  = '2A' 'C1'

The function I2OS() is invoked from Signature Generation (section A2.2.2), Signature Verification (section A2.2.3), and Encryption and Authentication (section A2.3.3).

B3 Hashing Algorithms (for RSA)

B3.1 Secure Hash Algorithm (SHA-1)

This algorithm is standardised in ISO/IEC 10118-3 and is used by the RSA-based mechanism specified in section A2.1. SHA-1 takes as input messages of arbitrary length and produces a 20-byte hash value.

The Hash Algorithm Indicator for this hashing algorithm shall be coded as hexadecimal '01'.

Annex C Informative References

1. A. Bosselaers and B. Preneel (eds.), *Integrity Primitives for Secure Information Systems*, Final Report of the RACE Integrity Primitives Evaluation (RIPE, RACE R1040), LNCS 1007, Springer-Verlag, 1995.
2. R. L. Rivest, A. Shamir, and L. Adleman, 'A method for obtaining digital signatures and public key cryptosystems', *Communications of the ACM*, vol. 21, 1978, pp. 120-126.
3. *EMV Issuer and Application Security Guidelines*.
4. *NIST Special Publication 800-22*, 'A statistical test suite for random number and pseudorandom number generation for cryptographic applications'.
5. *EMV Acquirer and Terminal Security Guidelines*.

Annex D Implementation Considerations

D1 Issuer and ICC RSA Public Key Length Considerations

Section D1 applies only to RSA-based mechanisms.

This specification follows the usual convention that allows the Issuer Public Key length to be equal to or less than the CA Public Key length and allows the ICC Public Key and ICC PIN Encipherment Public Key lengths to be equal to or less than the Issuer Public Key length. However, some further restrictions occur due to limitations of the command data structure.

Book 3 section 7 states that records are limited to 254 bytes including tag and length and as a consequence, if an ICC public key pair is required, the Issuer and ICC key lengths need to be less than the CA maximum of 248 bytes.

EMV Contact Interface Specification states that the maximum number of data bytes that may be sent with a command is 255 and the maximum number of data bytes for a response is 256. If dynamically signed data is included in a response from the ICC, then the latter restriction limits the maximum length of the ICC keys (see section D1.2).

D1.1 Issuer Public Key Restriction

For card applications supporting DDA, CDA, or Offline Enciphered PIN, the TLV encoded template containing the ICC Public Key Certificate needs to fit within the 254 byte record limit. To accommodate the tags and lengths of the certificate and the record template in the record containing this certificate, the maximum size of the ICC Public Key Certificate is restricted to 247 bytes (1976 bits), and consequently the Issuer Public Key, which is the same length as the certificate, is also restricted to 247 bytes.

D1.2 ICC Public Key Restriction

D1.2.1 CDA

The following restriction applies for card applications supporting CDA:

To ensure that the GENERATE APPLICATION CRYPTOGRAM response data length (format 2) is within the 256 byte constraint, the value portion of the Signed Dynamic Application Data needs to be limited in accordance with the other data elements contained within the template. This is achieved by limiting the size of the ICC public key, since owing to the properties of the cryptographic calculation, signature results are the same length as the key.

The lengths of the data in the GENERATE APPLICATION CRYPTOGRAM response are shown in Table 50:

		Length in Bytes			
		Tag	**Length**	**Value**	**Total Length**
Response Template		1 ('77')	2	—	3
	Cryptogram Information Data	2 ('9F27')	1	1	4
	Application Transaction Counter	2 ('9F36')	1	2	5
	Signed Dynamic Application Data	2 ('9F4B')	2	N_{IC}	N_{IC} + 4
	Issuer Application Data (optional)	2 ('9F10')	1	0 to 32	0 to 35
	Other optional data				var.

Table 50: Data Lengths in GENERATE AC Response

The tag and length of the response template, together with the tags, lengths, and values of the Cryptogram Information Data and Application Transaction Counter, and the tag and length of the Signed Dynamic Application Data are fixed in size and occupy 16 bytes. Thus, without Issuer Application Data, the maximum size of the Signed Dynamic Application Data and consequently the ICC Public Key is 240 bytes (1920 bits).

If Issuer Application Data is included, then the maximum size of the Signed Dynamic Application Data needs to be reduced accordingly. Including Issuer Application Data of tag, length, and 32 bytes of value (the maximum) results in a maximum size of 205 bytes (1640 bits) for the Signed Dynamic Application Data and consequently the ICC Public Key.

Note: If other optional data is appended in the response, then the length of this data and its associated tag and length field further restricts the length of the ICC Public Key.

D1.2.2 DDA

The following restriction applies for card applications supporting INTERNAL AUTHENTICATE Format 2:

To ensure that the INTERNAL AUTHENTICATE response data length is within the 256 byte limit, the length of the Signed Dynamic Application Data plus the length of the TLV encoded optional data (if present) shall not exceed 249 bytes. The length of the ICC Public Key is the same as the Signed Dynamic Application Data. The additional 7 bytes in the response are used for the tags and lengths of the response template and the Signed Dynamic Application Data.

D2 Format 1 Secure Messaging Illustration

Below is an illustration of Format 1 Secure Messaging as defined in section 9 using a command where the command data of the unsecured command is not considered to be BER-TLV encoded and using an 8-byte block cipher. The command data is included in the computation of the MAC as a data object in accordance with section 9.2.3. This is either the plaintext data object with tag '81' or, if secure messaging for confidentiality is applied, the data object for confidentiality with tag '87'.

D2.1 Securing the Command APDU

Before application of secure messaging, a command APDU is called an 'unsecured' command. After secure messaging is applied, a command APDU is called a 'secured' command.

Throughout this section, L_c is the length of the unsecured command data field; that is, the length of the command data before encipherment, BER-TLV encoding or inclusion of a MAC. New L_c is the length of the secured command data field; that is, the command data length after encipherment (if applied), BER-TLV encoding and inclusion of the MAC.

The unsecured command APDU has either the following structure:

'X0'	INS	P1	P2	L_c	Unsecured command data field

or, if there is no data (e.g. Application Block command), the following structure:

'X0'	INS	P1	P2

The secured command APDU has the following structure:

'XC'	INS	P1	P2	New L_c	Secured command data field

The following sections describe the construction of the secured command data field and the value of New L_c. Firstly for the case where integrity[54] (only) is required and secondly for the case where both confidentiality and integrity are required.

[54] For the purposes of this Annex the term 'integrity' represents the combination of integrity and authentication.

D2.1.1 Secure Messaging for Integrity

The secured command data field:

If secure messaging for integrity (only) is applied, the secured command data field is TLV-coded in the following way:

Tag 1	Length 1	Value 1	Tag 2	Length 2	Value 2
'81'	L	Unsecured command data field	'8E'	'04'-'08'	MAC (4-8 bytes)

In this case L has the same value as L_c.

The value of New L_c:

L_c is always coded on one byte, whereas L is coded on one or two bytes, depending on the length of the unsecured command data.

- If L is coded on one byte, the value of New L_c will range from $8+L_c$ to $12+L_c$, depending on the length of the MAC.
- If L is coded on two bytes, the value of New L_c will range from $9+L_c$ to $13+L_c$, depending on the length of the MAC.

If the command to be secured has no command data (e.g. Application Block) then the only data in the secured command is the MAC. In this case the secured command data field is TLV-coded in the following way:

Tag 1	Length 1	Value 1
'8E'	'04' – '08'	MAC (4-8 bytes)

and the value of New L_c will range from 6 to 10 depending on the length of the MAC.

D2.1.2 Secure Messaging for Confidentiality and Integrity

The secured command data field:

If secure messaging for both confidentiality and integrity is applied, the secured command data field is TLV-coded in the following way:

Tag 1	Length 1	Value 1	Tag 2	Length 2	Value 2
'87'	L	'01' || enciphered data field	'8E'	'04'-'08'	MAC (4-8 bytes)

In this case:

- The first byte in the value field of the cryptogram data object for confidentiality with tag '87' is the padding indicator byte. The value '01' indicates that the plaintext data is padded according to ISO/IEC 7816-4 before encipherment.
- Since the plaintext data is padded to be a multiple of 8 bytes (see D2.2), the resulting enciphered data field will range from $1+L_c$ to $8+L_c$. Consequently L will range from $2+L_c$ to $9+L_c$.

The value of New L_c:

L_c is always coded on one byte, whereas L is coded on one or two bytes, depending on the length of the unsecured command data.

- If L is coded on one byte, the value of New L_c will range from $10+L_c$ to $21+L_c$, depending on the length of padding appended to the unsecured command data and on the length of the MAC.
- If L is coded on two bytes, the value of New L_c will range from $11+L_c$ to $22+L_c$, depending on the length of padding appended to the unsecured command data and on the length of the MAC.

If the command to be secured has no command data (e.g. Application Block) then there is no data to be enciphered and so secure messaging for integrity (only) is applied.

Notes

1. The plaintext data is transported in the value field of a plaintext data object with tag '81'.

   The enciphered data is transported in the value field of a cryptogram data object for confidentiality with tag '87'.

2. The fact that the tag of the data object (whether plaintext or cryptogram) is odd-numbered indicates that the data object is included in the MAC computation.
3. The padding indicator byte is the mandatory first byte in the value field of a cryptogram data object for confidentiality with tag '87' (see ISO/IEC 7816-4.)

D2.2 Encipherment

If secure messaging for confidentiality is applied to the command, the unsecured command data field is enciphered in the following way:

- Padding and blocking of the data field is performed according to step 1 of section A1.1. A value of '01' of the padding indicator indicates that padding according to ISO/IEC 7816-4 always takes place even if the data field is a multiple of 8 bytes. Thus, the unsecured command data field is always padded to a multiple of 8 bytes prior to encipherment with a minimum of one byte of padding always present; if the length of the unsecured command data field is a multiple of 8 bytes, it is padded with 8 bytes ('80 00 00 00 00 00 00 00') prior to encipherment.
- The padded data is enciphered according to step 2 of section A1.1 using the Encipherment Session Key derived according to section 9.3.2.

D2.3 MAC Computation

MAC computation is performed in two steps:

- Padding of the input data (for use in this computation)
- Applying a MAC algorithm to the padded input data.

D2.3.1 Padding of the Input Data

Padding of the input data is performed according to ISO/IEC 7816-4:

- The command header of the secured command APDU

'XC'	INS	P1	P2

  is padded with '80 00 00 00'.
- If the unsecured command APDU contains a data field, a mandatory '80' byte is added to the right of
  - the plaintext data object (tag '81') or
  - the cryptogram data object for confidentiality (tag '87')

  that will be contained in the secured command data field. Then the smallest number of '00' bytes is added to the right such that the length of the resulting string is a multiple of 8 bytes.

The padded input data consists of the concatenation of the padded command header and the padded plaintext data object or the padded cryptogram data object for confidentiality (if present).

If MAC chaining is implemented then an 8-byte value is inserted to the left of the padded input data. This 8-byte value is:

- The Application Cryptogram generated by the card for the first or only script command,

- The MAC (the full 8 bytes prior to any optional truncation) of the preceding script command for all following script commands.

If MAC chaining is not implemented then the 8-byte Application Cryptogram generated by the card is inserted to the left of the padded input data.

D2.3.2 Cryptogram Computation

A MAC is computed over the padded input data according to step 3 of section A1.2.1 using the MAC Session Key derived according to section 9.2.2.

D3 Application Transaction Counter Considerations

This specification describes a two byte (16 bit) counter (the ATC) that is incremented during each transaction from a nominal starting value of '0000' to a maximum of 'FFFF'. With one increment per card session it gives an expected card life of 65,535 transactions.

The counter results in uniqueness to the cryptograms and provides tracking values for the host verification services, allowing replayed transactions and cloned cards to be identified. It may also be used in session key derivation schemes, such as the scheme described in section A1.3.

To avoid attacks based on session truncation, the counter should be incremented at the start of each transaction (for example during processing of the GET PROCESSING OPTIONS command). To prevent attacks based on duplicate data the counter should not be allowed to roll-over and the application should be blocked once the counter reaches 'FFFF'. Issuers should be aware that few, if any, cards in normal use will approach the 65,535 transaction limit (60 per day every day for a 3 year card) and that cards with a high count may have been subject to attack. If a card with a shorter lifetime is desired, consideration may be given to a lower limit, or to starting the counter at an intermediate value.

D4 CDA Modes

EMV permits flexible terminal CDA behaviour that can potentially improve transaction performance. These include the selective use of CDA for online authorisations and public key retrieval relative to Terminal Action Analysis (TAA).

CDA for online authorisations

Terminals supporting CDA have the following options:

- Request or not request CDA on ARQCs
- Request or not request CDA on 2nd GENERATE AC (TC) after an approved online authorisation

As part of the EMV type approval, a terminal kernel configuration supporting CDA must now identify which of the above options the terminal supports.

Thus, an EMV terminal configuration supporting CDA will operate in one of four modes:

Mode	Request CDA on ARQC	Request CDA on 2nd GEN AC (TC) after approved online authorisation
1	Yes	Yes
2	Yes	No
3	No	No
4	No	Yes

Table 51: CDA Modes

In the years since publication of SU44, terminals have become faster, therefore the performance impact of always doing CDA, irrespective of online authorisation, is no longer significant. For this reason Mode 1 is now recommended instead of the other modes.

Public Key retrieval

Before publication of SU44, terminals experiencing CDA failure prior to TAA decline the transaction. Following publication of SU44, terminals that comply with SU44 that experience CDA failure prior to TAA shall proceed with TAA to decide whether to decline or send the transaction online.

One possible reason for CDA failing is a problem retrieving the public keys. According to SU44, terminals that find this problem before TAA will proceed with TAA to decide whether to decline or send the transaction online. Thus, an online authorisation (without CDA) is possible, rather than the decline that was previously required.

SU44 also clarifies that keys can be retrieved before or after TAA which can lead to performance improvements for terminals operating predominantly online. This is because if the TAA results in an online authorisation and if the terminal requests an ARQC without CDA (i.e. it is operating in Mode 3 or 4), then the retrieval of the issuer and ICC public keys need not be completed, saving the RSA processing.[55]

Note that section 6 now mandates for CDA that all terminals verify before TAA that they contain the Certification Authority Public Key identified by the card. Such verification does not involve time-consuming cryptographic processing. If the correct key is not present, then the Terminal Action Codes can result in Terminal Action Analysis sending the transaction online without requesting CDA in the GENERATE AC.

Recommendations

The following recommendations apply to terminals supporting CDA.

For online capable terminals that are able to perform certificate verification quickly, it is recommended that the terminal retrieve the issuer and ICC public keys before TAA. This is to ensure that in the unlikely event that key retrieval fails then the terminal can request an ARQC without CDA rather than decline the transaction.

Terminal vendors are recommended to implement Mode 1 only.

As Mode 4 does not provide significant benefit, terminal vendors are recommended not to implement Mode 4. If CDA is needed on all 2nd GENERATE AC commands requesting a TC, then Mode 1 can be used.

TVR bit setting for Modes 3 and 4

An issuer may inspect the TVR either during an online authorisation or after the transaction has completed. If the TVR bit for 'Offline data authentication was not performed' (byte 1, bit 8) is equal to zero then this should be interpreted as meaning that 'the card and terminal supported a common ODA mechanism (one of SDA, DDA, or CDA)'. If the common ODA mechanism was CDA then this bit being equal to zero does not always imply that 'Offline data authentication was performed'.

Specifically section 6 states that for CDA the TVR bit for 'Offline data authentication was not performed' is set to zero after online authorisation even though CDA may not have been performed on the 1st GENERATE AC (i.e. the terminal implements CDA Mode 3 or 4) [56]. This means that if the transaction was approved online and the terminal is Mode 3 then the transaction may successfully complete without having completed offline data authentication yet the TVR bit for 'Offline data authentication was not performed' will be set to zero.

[55] If Offline Enciphered PIN is performed then this will force the retrieval of the issuer and ICC public keys to happen before PIN verification is performed.

[56] Issuers may also note that for terminals approved to the old test plan version 4.1d of CDA Mode 3, the TVR bit for 'Offline data authentication was not performed' would remain set to 1 after the online authorisation.

Furthermore from a card perspective, during a transaction where both card and terminal support CDA, the card may legitimately receive a 2nd GENERATE AC TVR where the bit for 'Offline data authentication was not performed' is set to zero even though the card did not generate a CDA signature on the 1st GENERATE AC (i.e. the terminal implements CDA Mode 3 or 4).

Part IV

Common Core Definitions

Common Core Definitions

This Part describes an optional extension to this Book, to be used when implementing the Common Core Definitions (CCD).

These Common Core Definitions specify a minimum common set of card application implementation options, card application behaviours, and data element definitions sufficient to accomplish an EMV transaction. Terminals certified to be compliant with the existing EMV specifications will, without change, accept cards implemented according to the Common Core Definitions, since the Common Core Definitions are supported within the existing EMV requirements.

To be compliant with the Common Core Definitions, an implementation shall implement all the additional requirements in the Common Core Definitions Parts of all affected Books.

Changed Sections

Each section heading below refers to the section in this Book to which the additional requirements apply. The text defines requirements for a common core implementation, in addition to the requirements already specified in the referenced section of EMV.

Part II – Security and Key Management Techniques

6 Offline Dynamic Data Authentication

6.5 Dynamic Data Authentication (DDA)

6.5.1 Dynamic Signature Generation

An ICC that supports DDA shall contain a DDOL. The DDOL shall contain only the Unpredictable Number generated by the terminal (tag '9F37', 4 bytes binary).

6.6 Combined DDA/Application Cryptogram Generation (CDA)

6.6.1 Dynamic Signature Generation

For a CCD-compliant application that supports CDA, the following requirements shall apply.

The ICC response to the GENERATE AC command for a TC or ARQC shall contain only the data objects specified in Table CCD 1 (which, for CCD, supplants Table 20).

Tag	Length	Value	Presence
'9F27'	1	Cryptogram Information Data	M
'9F36'	2	Application Transaction Counter	M
'9F4B'	N_{IC}	Signed Dynamic Application Data	M
'9F10'	32	Issuer Application Data	M

Table CCD 1: Data Objects in Response to GENERATE AC for TC or ARQC

3. If the ICC responds with an AAC, the ICC response shall be coded according to format 2 as specified in Book 3 section 6.5.5.4 and shall contain only the data elements specified in Table CCD 2 (which, for CCD, supplants Table 21).

Tag	Length	Value	Presence
'9F27'	1	Cryptogram Information Data	M
'9F36'	2	Application Transaction Counter	M
'9F26'	8	Application Authentication Cryptogram	M
'9F10'	32	Issuer Application Data	M

Table CCD 2: Data Objects in Response to GENERATE AC for AAC

8 Application Cryptogram and Issuer Authentication

8.1 Application Cryptogram Generation

8.1.1 Data Selection

Table CCD 3 lists the set of data elements to be included in the Application Cryptogram for a cryptogram defined by the Common Core Definitions with a Cryptogram Version of '5' (which uses Triple DES) or '6' (which uses AES). The data elements shall be included in the order shown in Table CCD 3 [which, for CCD, supplants Table 28].

Value	Source
Amount, Authorised (Numeric)	Terminal
Amount Other (Numeric)	Terminal
Terminal Country Code	Terminal
Terminal Verification Results	Terminal
Transaction Currency Code	Terminal
Transaction Date	Terminal
Transaction Type	Terminal
Unpredictable Number	Terminal
Application Interchange Profile	ICC
Application Transaction Counter	ICC
Issuer Application Data	ICC

Table CCD 3: Data Elements for Application Cryptogram Generation

8.1.2 Application Cryptogram Algorithm

The Application Cryptogram shall be 8 bytes. EMV-defined Application Cryptograms shall be generated using one of two methods:

- using the MAC algorithm specified in section A1.2.1 and ISO/IEC 9797-1 Algorithm 3 with DES, and *s*=8. This method shall be used for a Cryptogram Version of '5'.
- using the MAC algorithm specified in section A1.2.2 with AES and *s*=8. This method shall be used for a Cryptogram Version of '6'.

For an application defined by the Common Core Definitions with a Cryptogram Version of '5' or '6', the AC Session Key shall be derived using the method specified in section A1.3.1.

8.2 Issuer Authentication

The CCD-compliant application shall support Issuer Authentication according to ARPC Method 2 specified in section 8.2.2.

8.2.2 ARPC Method 2

For a cryptogram defined by the Common Core Definitions with a Cryptogram Version of '5' or '6', the Card Status Update (CSU) data element shall be coded according to section C10 in the CCD part of Book 3.

The default value for Proprietary Authentication Data is zero.

If the 'Proprietary Authentication Data Included' bit in the CSU has the value 0b, then the length of Proprietary Authentication Data included in generation and validation of the ARPC shall be 0 bytes.

Note: If the 'Proprietary Authentication Data Included' bit in the CSU has the value 0b, the Proprietary Authentication Data is not protected by Issuer Authentication. The card should not take action based on the settings of any Proprietary Authentication Data that is not protected by Issuer Authentication.

If the 'Proprietary Authentication Data Included' bit in the CSU has the value 1b, then the Proprietary Authentication Data included in the Issuer Authentication Data shall be used in generation and validation of the ARPC.

8.3 Key Management

For a cryptogram defined by the Common Core Definitions with a Cryptogram Version of '5', the ICC Master Key shall be derived using the Option B method described in section A1.4.2.

For a cryptogram defined by the Common Core Definitions with a Cryptogram Version of '6', the ICC Master Key shall be derived using the Option C method described in section A1.4.3.

9 Secure Messaging

9.1 Secure Messaging Format

All commands using Secure Messaging shall use Secure Messaging Format 1 as described in this Book.

9.2 Secure Messaging for Integrity and Authentication

9.2.1 Command Data Field

All commands using Secure Messaging for integrity and authentication:

- shall use Secure Messaging Format 1 as described in section 9.2.1.1
- shall chain the MACs from one command to the next according to the method recommended in section 9.2.3.1.

9.2.1.1 Format 1

All command data shall be included in the computation of the MAC.

Data enciphered for confidentiality shall be encapsulated with tag '87'. Data not enciphered for confidentiality shall be encapsulated with tag '81'.

The CCD-compliant application shall accept 4-byte MACs, and the issuer can only rely on support of 4-byte MACs.

9.2.2 MAC Session Key Derivation

For an application with a cryptogram defined by the Common Core Definitions with a Cryptogram Version of '5' or '6', the MAC Session Key shall be derived using the method specified in section A1.3.1.

9.2.3 MAC Computation

Secure Messaging is according to Secure Messaging Format 1.

The CCD-compliant application shall accept 4-byte MACs, and the issuer can only rely on support of 4-byte MACs.

The MAC for Cryptogram Version '5' shall be generated using the MAC algorithm specified in section A1.2.1 step 3, second bullet using DES as the block cipher.

The MAC for Cryptogram Version '6' shall be generated using the MAC algorithm specified in section A1.2.2 using AES as the block cipher.

9.3 Secure Messaging for Confidentiality

9.3.1 Command Data Field

All commands using secure messaging for confidentiality shall use Secure Messaging Format 1 as described in section 9.3.1.1.

9.3.1.1 Format 1

Data enciphered for confidentiality shall be encapsulated with tag '87'.

Data that is enciphered in the Issuer Script Command data field shall always be padded before encipherment. The Padding Indicator byte shown in Figure 8 shall be included and shall be set to the value '01' to indicate padding is present.

9.3.2 Encipherment Session Key Derivation

For an application with a cryptogram defined by the Common Core Definitions with a Cryptogram Version of '5' or '6', the Encipherment Session Key shall be derived using the method specified in section A1.3.1.

9.3.3 Encipherment/Decipherment

Encipherment/decipherment of the command data field shall use the Cipher Block Chaining (CBC) Mode described in section A1.1. For Cryptogram Version '5' the Triple DES algorithm specified in section B1.1 is used. For Cryptogram Version '6' the AES algorithm specified in section B1.1 is used. For both versions the Padding Indicator byte is set to the value '01' to indicate that padding is present.

9.4 Key Management

For an application with a cryptogram defined by the Common Core Definitions with a Cryptogram Version of '5', the ICC MAC and Encipherment Master Keys shall be derived using the Option B method described in section A1.4.2.

For an application with a cryptogram defined by the Common Core Definitions with a Cryptogram Version of '6', the ICC MAC and Encipherment Master Keys shall be derived using the Option C method described in section A1.4.3.

Index

A

AAC ... 86
Abbreviations ... 25
AC ... *See* Application Cryptogram
AFL ... 42, 56
AID ... 53
AIP ... 42, 49, 56
Algorithm
  Application Cryptogram Generation ... 87
  DES ... 141
  RSA ... 152
  SHA-1 ... 168
Application Authentication Cryptogram ... *See* AAC
Application Cryptogram ... 66, 86
  and Issuer Authentication ... 86
  Generation
    Algorithm ... 87
    Data Selection ... 86
  Key Management ... 89
  MAC Chaining ... 92
Application Cryptogram Master Key ... 87
Application PAN ... 61, 95, 138
Application PAN Sequence Number ... 95, 138
Application Transaction Counter ... *See* ATC
ARPC ... 86
ARPC Methods for Issuer Authentication
  Method 1 ... 88
  Method 2 ... 89
ARQC ... 86, 88, 89
ATC ... 69, 87, 94, 120, 137, 178
Authorisation Request Cryptogram ... *See* ARQC
Authorisation Response Code ... 88
Authorisation Response Cryptogram ... *See* ARPC

C

CA Private Key ... 37
CA Public Key ... 37
Card Public Key Algorithm Suite Indicator ... 117
Card Status Update ... *See* CSU
CCD ... *See* Common Core Definitions
CDA ... 49, 65
  Dynamic Signature Generation ... 65
  Dynamic Signature Verification ... 69
  Keys and Certificates ... 53
  Retrieval of Certification Authority Public Key ... 57, 110
  Retrieval of ICC Public Key ... 59
  Retrieval of Issuer Public Key ... 57
  Sample Flow ... 72
CDOL1 ... 66, 71, 118
CDOL2 ... 66, 71, 118
Certificate Expiration Date ... 46, 59, 61, 116
Certificate Serial Number ... 46, 59, 113
Certificates and Keys
  DDA and CDA ... 53
  PIN Encipherment ... 77, 124
  SDA ... 39
  XDA ... 108
Certification Authority ... 37
Certification Authority Private Key ... 39, 53, 108
Certification Authority Public Key ... 38, 52, 57, 97, 110
  Key Management Requirements ... 97
  Retrieval for DDA and CDA ... 57, 110
  Retrieval for SDA ... 44
Certification Authority Public Key Algorithm Indicator ... 100
Certification Authority Public Key Exponent ... 39, 53, 152
Certification Authority Public Key Index ... 44, 52, 100, 110
Certification Authority Public Key Modulus ... 39, 53
CID ... 69, 120
Commands
  GENERATE AC ... 66, 118
  GET CHALLENGE ... 80, 125
  GET PROCESSING OPTIONS ... 67
  INTERNAL AUTHENTICATE ... 61, 172
  READ RECORD ... 53
  VERIFY ... 80, 126
Common Core Definitions ... 183
  Application Cryptogram Generation ... 185
  CDA ... 184
  DDA ... 183
  Dynamic Signature Generation ... 183, 184
  Encipherment/Decipherment ... 188
  Issuer Authentication ... 186
  Key Management ... 186, 188
  MAC Computation ... 187
  Secure Messaging for Confidentiality ... 188
  Secure Messaging for Integrity and Authentication 187
  Secure Messaging Format ... 187
Cryptogram Information Data ... 70, *See* CID
Cryptographic Algorithms
  Asymmetric
    RSA Algorithm ... 152
  Hashing
    Secure Hash Algorithm (SHA-1) ... 168
  Symmetric
    Data Encryption Standard (DES) ... 151
CSU ... 89

D

Data Authentication Code ... 48
Data Element Format Conventions ... 34
Data Encryption Standard ... *See* DES
Data Selection
  Application Cryptogram Generation ... 86
DDA ... 49
  Dynamic Signature Generation ... 61

Dynamic Signature Verification 63
Keys and Certificates 53
Retrieval of Certification Authority Public Key 57, 110
Retrieval of ICC Public Key 59
Retrieval of Issuer Public Key 57
DDOL 61
Decipherment
Symmetric Security Mechanisms 134
Default DDOL 61
Definitions 17
Derivation
Master Key 138
Session Key 137
DES 151
Dynamic Signature
Generation
CDA 65
DDA 61
Verification
CDA 69
DDA 63
XDA 121

E

Encipherment
Symmetric Security Mechanisms 133
Encipherment Master Key 94, 95
Encipherment Session Key 94, 176

F

Format 1 Secure Messaging Illustration 173

G

GENERATE AC
Response 69, 120
GENERATE AC Command 66, 118
GET CHALLENGE Command 80, 125
GET PROCESSING OPTIONS Command 67

H

Hash Algorithm Indicator 46, 61, 64, 71, 168
Hashing Algorithms 168

I

ICC Application Cryptogram Master Keys 89
ICC Dynamic Data 62, 68
ICC Dynamic Number 62, 64, 68
ICC Master Key 137, 138
ICC Private Key 62, 67, 118
ICC Public Key 53, 61, 63, 79, 124
Restriction on Length 171
Retrieval for DDA and CDA 59
Retrieval for XDA 114
ICC Public Key Algorithm Indicator 61
ICC Public Key Certificate 53, 108, 116
ICC Public Key Exponent 53, 152
ICC Public Key Remainder 53, 61
ICC Session Key 137
ICC Unpredictable Number 81, 127
ICCD Hash Algorithm Indicator 117
Implementation Considerations
Application Transaction Counter 178
Format 1 Secure Messaging Illustration 173
ICC Public Key Restriction 171
Issuer Public Key Restriction 170
Informative References 169
INTERNAL AUTHENTICATE Command 61, 172
Issuer Application Data 69, 120
Issuer Authentication 88
ARPC Method 1 88
ARPC Method 2 89
Key Management 89
Issuer Authentication Data 89
Issuer Certificate Expiration Date 113
Issuer Certificate Format 113
Issuer Identifier 46, 59, 113
Issuer Master Key 138
Issuer Private Key 37, 39, 53, 108
Issuer Public Key 37, 46, 59
Restriction on Length 170
Retrieval for DDA and CDA 57
Retrieval for SDA 44
Issuer Public Key Algorithm Indicator 46
Issuer Public Key Algorithm Suite Indicator 113
Issuer Public Key Certificate 37, 39, 44, 53, 108, 111, 113
Issuer Public Key Exponent 39, 53, 152
Issuer Public Key Modulus 39, 53
Issuer Public Key Remainder 39, 46, 53, 59
IV 173

K

Key Derivation
Master Key 138
Session Key 137
Key Management 89
Application Cryptogram 89
Issuer Authentication 89
Secure Messaging 95
Key Management Requirements
Certification Authority Public Key Introduction 97
Certification Authority Public Key Storage 100
Certification Authority Public Key Withdrawal 102
Key Restriction
Implementation Considerations 170, 171

Keys and Certificates
DDA and CDA 53
PIN Encipherment 77, 124
SDA 39
XDA 108

M

MAC 134
MAC Chaining 92
MAC Master Key 92, 95
MAC Session Key 92, 134, 177
Master Key Derivation 138
Message Authentication Code ..... *See* MAC

N

Normative References 14
Notations 32

O

Offline Dynamic Data Authentication 49
Offline Enciphered PIN 76

P

PDOL 67, 71
PIN Block 76, 124
PIN Decipherment and Verification 127
PIN Encipherment 76, 125
Keys and Certificates 77, 124
PIN Encipherment and Verification 80
Proprietary Authentication Data 89
Public Key Algorithm Indicator 152
Public Key Modulus 39, 53, 77, 152
Public Key Restriction
Implementation Considerations 170, 171

R

READ RECORD Command 53
References
Informative 169
Normative 14
Revision Log 3
RID 38, 44, 52, 53, 100, 110
RSA Algorithm 152

S

Scope 12
SDA 37
Keys and Certificates 39
Retrieval of Certification Authority Public Key 44
Retrieval of Issuer Public Key 44
Verification of Signed Static Application Data 47
Secure Hash Algorithm ..... *See* SHA-1
Secure Messaging 90
Format 90
Key Management 95
Secure Messaging for Confidentiality
Command Data Field
Format 1 94
Format 2 94
Encipherment Session Key Derivation 94
Encipherment/Decipherment 95
Secure Messaging for Integrity and Authentication
Command Data Field
Format 1 90
Format 2 91
MAC Chaining 92
MAC Computation 92
MAC Session Key Derivation 92
Secure Messaging Illustration 173
MAC Computation 176
Securing the Case 3 Command APDU 173
Security Mechanisms
Asymmetric
Digital Signature Scheme Giving Message Recovery 141
Symmetric
Encipherment 133
Master Key Derivation 138
Message Authentication Code 134
Session Key Derivation 137
Symmetric Decipherment 134
Session Key Derivation 137
SHA-1 168
Signed Dynamic Application Data 52, 62, 63, 69, 107, 118, 119, 120, 121
Signed Static Application Data 37, 39
Verification for SDA 47
Static Data Authentication ..... *See* SDA
Static Data Authentication Tag List 42, 47, 56
Storage
Certification Authority Public Key 100

T

TC 86
Terminal Security Requirements 97
Terminal Verification Results ..... *See* TVR
Terminology 35
Transaction Certificate ..... *See* TC
Transaction Data Hash Code 67, 71
TVR 38, 52

U

Unpredictable Number 61, 66, 118

V

VERIFY Command 80, 126

W

Withdrawal
Certification Authority Public Key 102

X

XDA
Authentication of ICC Public Key 114
Authentication of Issuer Public Key 111
Dynamic Signature Verification 121
Keys and Certificates 108
Sample Flow 121