



EMV[®]

Contactless Specifications for Payment Systems

Book C-2

Kernel 2 Specification

Version 2.11
June 2023

Legal Notice

Unless the user has an applicable separate agreement with EMVCo or with the applicable payment system, any and all uses of these Specifications is subject to the terms and conditions of the EMVCo Terms of Use agreement available at www.emvco.com and the following supplemental terms and conditions.

Except as otherwise may be expressly provided in a separate agreement with EMVCo, the license granted in the EMVCo Terms of Use specifically excludes (a) the right to disclose, distribute or publicly display these Specifications or otherwise make these Specifications available to any third party, and (b) the right to make, use, sell, offer for sale, or import any software or hardware that practices, in whole or in part, these Specifications. Further, EMVCo does not grant any right to use the Kernel Specifications to develop contactless payment applications designed for use on a Card (or components of such applications). As used in these supplemental terms and conditions, the term "Card" means a proximity integrated circuit card or other device containing an integrated circuit chip designed to facilitate contactless payment transactions. Additionally, a Card may include a contact interface and/or magnetic stripe used to facilitate payment transactions. To use the Specifications to develop contactless payment applications designed for use on a Card (or components of such applications), please contact the applicable payment system. To use the Specifications to develop or manufacture products, or in any other manner not provided in the EMVCo Terms of Use, please contact EMVCo.

These 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 2.11

The following changes have been made to Book C-2 since the publication of Version 2.10.

Incorporated changes described in the following Specification Bulletin:

- Specification Bulletin No. 261 October 2021: Implementation Options for EMV Book C-2 V2.10

Other changes:

- Simplified informative information in chapters 2 and 3 to reflect functional changes of SB261.
- Removed the 'Value Qualifier', 'Value' and 'Currency Code' fields from *User Interface Request Data* and changed the length to 13 bytes.
- Removed *Offline Accumulator Balance* from data retrievable with GET DATA command and from Data Dictionary
- Added *Token Requestor ID* to Data Dictionary and *Data Record*

Contents

Revision Log – Version 2.11	1
1 Using This Manual	17
1.1 Purpose	17
1.2 Audience	17
1.3 Related Information	18
1.4 Terminology	20
1.5 Notations	22
1.5.1 State Machine	22
1.5.2 Data Object Notation	24
1.5.3 Other Notational Conventions	26
2 General Architecture	29
2.1 Introduction	29
2.2 Reader Processes	30
2.2.1 Process P	31
2.2.2 Process D	31
2.2.3 Process S	32
2.2.4 Process K	32
2.2.5 Process M	34
2.3 The Reader Database	36
3 Reader Process K — Kernel Processing	39
3.1 Implementation Options	39
3.2 The Kernel Database	40
3.3 Mag-Stripe Mode and EMV Mode	41
3.3.1 Overall Transaction Flow	41
3.3.2 Mag-stripe Mode	41
3.3.3 EMV Mode	42
3.4 Data Exchange	43
3.4.1 Introduction	43
3.4.2 Sending Data to the Terminal	43
3.4.3 Requesting Data from the Terminal	44
3.4.4 Synchronization	45
3.5 Data Storage	46
3.5.1 Introduction	46
3.5.2 Standalone Data Storage	46
3.5.3 Integrated Data Storage	47
3.6 Mobile Transactions	52

3.6.1	Introduction.....	52
3.6.2	Mobile Mag-stripe Mode Transactions	53
3.6.3	Mobile EMV Mode Transactions	54
3.7	Relay Resistance Protocol.....	55
3.7.1	Introduction.....	55
3.7.2	Protocol	55
3.8	Optimisation For Transactions Without CDA.....	56
3.8.1	Introduction.....	56
3.8.2	Activation/Deactivation	56
3.9	Kernel Configuration Options	57
4	Data Organization	59
4.1	TLV Database.....	59
4.1.1	Principles.....	59
4.1.2	Access Conditions	60
4.1.3	Services	61
4.1.4	DOL Handling	65
4.2	Working Variables	66
4.3	List Handling.....	67
4.4	Configuration Data.....	69
4.4.1	Configuration Data – TLV Database	69
4.4.2	CA Public Key Database	71
4.4.3	Certification Revocation List	72
4.5	Lists of Data Objects in OUT	73
4.5.1	Outcome Parameter Set.....	73
4.5.2	User Interface Request Data	73
4.5.3	Data Record	73
4.5.4	Discretionary Data	76
4.6	Data Object Format	78
4.6.1	Format.....	78
4.6.2	Format Checking	78
4.7	Bitmaps Used in Discretionary Data	80
4.8	Reserved for Future Use (RFU).....	81
5	C-APDU Commands	83
5.1	Introduction.....	83
5.2	Compute Cryptographic Checksum	84
5.2.1	Definition and Scope	84
5.2.2	Command Message	84
5.2.3	Data Field Returned in the Response Message.....	85

5.2.4	Status Bytes	85
5.3	Exchange Relay Resistance Data	86
5.3.1	Definition and Scope	86
5.3.2	Command Message	86
5.3.3	Data Field Returned in the Response Message.....	86
5.3.4	Status Bytes	87
5.4	Generate AC	88
5.4.1	Definition and Scope	88
5.4.2	Command Message	88
5.4.3	Data Field Returned in the Response Message.....	90
5.4.4	Status Bytes	92
5.5	Get Data	93
5.5.1	Definition and Scope	93
5.5.2	Command Message	93
5.5.3	Data Field Returned in the Response Message.....	94
5.5.4	Status Bytes	94
5.6	Get Processing Options.....	95
5.6.1	Definition and Scope	95
5.6.2	Command Message	95
5.6.3	Data Field Returned in the Response Message.....	95
5.6.4	Status Bytes	97
5.7	Put Data	98
5.7.1	Definition and Scope	98
5.7.2	Command Message	98
5.7.3	Data Field Returned in the Response Message.....	98
5.7.4	Status Bytes	99
5.8	Read Record	100
5.8.1	Definition and Scope	100
5.8.2	Command Message	100
5.8.3	Data Field Returned in the Response Message.....	100
5.8.4	Status Bytes	101
6	Kernel State Machine	103
6.1	Implementation Principles.....	103
6.2	Kernel Started	104
6.2.1	Local Variables.....	104
6.2.2	Flow Diagram	104
6.2.3	Processing.....	106
6.3	State 1 – Idle	108

6.3.1	Local Variables	108
6.3.2	Flow Diagram	108
6.3.3	Processing.....	113
6.4	State 2 – Waiting for PDOL Data	120
6.4.1	Local Variables	120
6.4.2	Flow Diagram	120
6.4.3	Processing.....	123
6.5	State 3 – Waiting for GPO Response	125
6.5.1	Local Variables	125
6.5.2	Flow Diagram	125
6.5.3	Processing.....	134
6.6	State R1 – Waiting for Exchange Relay Resistance Data Response	143
6.6.1	Local Variables	143
6.6.2	Flow Diagram	143
6.6.3	Processing.....	148
6.7	States 3, R1 – Common Processing	153
6.7.1	Local Variables	153
6.7.2	Flow Diagram	153
6.7.3	Processing.....	158
6.8	State 4 – Waiting for EMV Read Record Response	162
6.8.1	Local Variables	162
6.8.2	Flow Diagram	162
6.8.3	Processing.....	168
6.9	State 5 – Waiting for Get Data Response	175
6.9.1	Local Variables	175
6.9.2	Flow Diagram	175
6.9.3	Processing.....	179
6.10	State 6 – Waiting for EMV Mode First Write Flag	182
6.10.1	Local Variables	182
6.10.2	Flow Diagram	182
6.10.3	Processing.....	185
6.11	States 4, 5, and 6 – Common Processing	186
6.11.1	Local Variables	186
6.11.2	Flow Diagram	186
6.11.3	Processing.....	194
6.12	State 7 – Waiting for Mag-stripe Read Record Response	203
6.12.1	Local Variables	203
6.12.2	Flow Diagram	203

6.12.3	Processing.....	207
6.13	State 8 – Waiting for Mag-stripe First Write Flag.....	213
6.13.1	Local Variables.....	213
6.13.2	Flow Diagram	213
6.13.3	Processing.....	215
6.14	States 7 and 8 – Common Processing.....	216
6.14.1	Local Variables.....	216
6.14.2	Flow Diagram	216
6.14.3	Processing.....	220
6.15	State 9 – Waiting for Generate AC Response.....	224
6.15.1	Local Variables.....	224
6.15.2	Flow Diagram	224
6.15.3	Processing.....	238
6.16	State 12 – Waiting for Put Data Response Before Generate AC.....	260
6.16.1	Local Variables.....	260
6.16.2	Flow Diagram	260
6.16.3	Processing.....	263
6.17	State 13 – Waiting for CCC Response – 1.....	265
6.17.1	Local Variables.....	265
6.17.2	Flow Diagram	265
6.17.3	Processing.....	272
6.18	State 14 – Waiting for CCC Response – 2.....	281
6.18.1	Local Variables.....	281
6.18.2	Flow Diagram	281
6.18.3	Processing.....	288
6.19	State 15 – Waiting for Put Data Response After Generate AC.....	297
6.19.1	Local Variables.....	297
6.19.2	Flow Diagram	297
6.19.3	Processing.....	300
7	Procedures.....	303
7.1	Procedure – CVM Selection	303
7.1.1	Local Variables.....	303
7.1.2	Flow Diagram	303
7.1.3	Processing.....	308
7.2	Procedure – Prepare Generate AC Command	314
7.2.1	Local Variables.....	314
7.2.2	Flow Diagram	314
7.2.3	Processing.....	322

7.3	Procedure – Processing Restrictions	329
7.3.1	Local Variables	329
7.3.2	Flow Diagram	329
7.3.3	Processing.....	336
7.4	Procedure – Terminal Action Analysis	342
7.4.1	Local Variables	342
7.4.2	Flow Diagram	342
7.4.3	Processing.....	346
8	Security Algorithms.....	349
8.1	Unpredictable Number Generation	349
8.2	OWHF2	350
8.3	OWHF2AES	351
Annex A	Data Dictionary	353
A.1	Data Objects by Name.....	353
A.1.1	Account Type	353
A.1.2	Acquirer Identifier	353
A.1.3	Active AFL	353
A.1.4	Active Tag	354
A.1.5	AC Type	354
A.1.6	Additional Terminal Capabilities.....	354
A.1.7	Amount, Authorized (Numeric).....	356
A.1.8	Amount, Other (Numeric).....	356
A.1.9	Application Capabilities Information	357
A.1.10	Application Cryptogram	358
A.1.11	Application Currency Code	358
A.1.12	Application Currency Exponent.....	359
A.1.13	Application Effective Date.....	359
A.1.14	Application Expiration Date.....	359
A.1.15	Application File Locator	360
A.1.16	Application Interchange Profile	361
A.1.17	Application Label	361
A.1.18	Application Preferred Name.....	362
A.1.19	Application PAN.....	362
A.1.20	Application PAN Sequence Number	362
A.1.21	Application Priority Indicator	362
A.1.22	Application Transaction Counter	363
A.1.23	Application Usage Control	363
A.1.24	Application Version Number (Card)	364

A.1.25	Application Version Number (Reader)	364
A.1.26	CA Public Key Index (Card)	364
A.1.27	Card Data Input Capability	365
A.1.28	CDOL1	365
A.1.29	CDOL1 Related Data	365
A.1.30	Cryptogram Information Data	366
A.1.31	CVC3 (Track1)	366
A.1.32	CVC3 (Track2)	366
A.1.33	CVM Capability – CVM Required	367
A.1.34	CVM Capability – No CVM Required	367
A.1.35	CVM List	368
A.1.36	CVM Results	368
A.1.37	Data Needed	368
A.1.38	Data Record	369
A.1.39	Data To Send	369
A.1.40	DD Card (Track1)	369
A.1.41	DD Card (Track2)	370
A.1.42	Default UDOL	370
A.1.43	Device Estimated Transmission Time For Relay Resistance R-APDU	370
A.1.44	Device Relay Resistance Entropy	371
A.1.45	DF Name	371
A.1.46	Discretionary Data	371
A.1.47	DS AC Type	372
A.1.48	DS Digest H	372
A.1.49	DSDOL	373
A.1.50	DS ID	373
A.1.51	DS Input (Card)	374
A.1.52	DS Input (Term)	374
A.1.53	DS ODS Card	374
A.1.54	DS ODS Info	375
A.1.55	DS ODS Info For Reader	375
A.1.56	DS ODS Term	376
A.1.57	DS Requested Operator ID	376
A.1.58	DS Slot Availability	377
A.1.59	DS Slot Management Control	377
A.1.60	DS Summary 1	378
A.1.61	DS Summary 2	378

A.1.62 DS Summary 3	378
A.1.63 DS Summary Status	379
A.1.64 DS Unpredictable Number	379
A.1.65 DSVN Term	380
A.1.66 Error Indication	380
A.1.67 Failed MS Cntr.....	382
A.1.68 File Control Information Issuer Discretionary Data.....	382
A.1.69 File Control Information Proprietary Template	383
A.1.70 File Control Information Template.....	383
A.1.71 Hold Time Value	384
A.1.72 ICC Dynamic Number.....	385
A.1.73 ICC Public Key Certificate	385
A.1.74 ICC Public Key Exponent	385
A.1.75 ICC Public Key Remainder	385
A.1.76 IDS Status	386
A.1.77 Interface Device Serial Number	386
A.1.78 Issuer Action Code – Default	386
A.1.79 Issuer Action Code – Denial	387
A.1.80 Issuer Action Code – Online	387
A.1.81 Issuer Application Data.....	387
A.1.82 Issuer Code Table Index.....	388
A.1.83 Issuer Country Code.....	388
A.1.84 Issuer Public Key Certificate	388
A.1.85 Issuer Public Key Exponent.....	388
A.1.86 Issuer Public Key Remainder.....	389
A.1.87 Kernel Configuration.....	389
A.1.88 Kernel ID	389
A.1.89 Language Preference	390
A.1.90 Log Entry	390
A.1.91 Mag-stripe Application Version Number (Reader).....	390
A.1.92 Mag-stripe CVM Capability – CVM Required	391
A.1.93 Mag-stripe CVM Capability – No CVM Required.....	392
A.1.94 Maximum Relay Resistance Grace Period.....	392
A.1.95 Max Time For Processing Relay Resistance APDU.....	393
A.1.96 Measured Relay Resistance Processing Time.....	393
A.1.97 Merchant Category Code.....	393
A.1.98 Merchant Custom Data.....	394
A.1.99 Merchant Identifier	394

A.1.100 Merchant Name and Location.....	394
A.1.101 Message Hold Time.....	394
A.1.102 Minimum Relay Resistance Grace Period	395
A.1.103 Min Time For Processing Relay Resistance APDU.....	395
A.1.104 Mobile Support Indicator.....	395
A.1.105 NATC(Track1)	396
A.1.106 NATC(Track2)	396
A.1.107 Next Cmd	397
A.1.108 nUN	397
A.1.109 ODA Status	398
A.1.110 Outcome Parameter Set.....	399
A.1.111 Payment Account Reference	401
A.1.112 PCVC3(Track1).....	401
A.1.113 PCVC3(Track2).....	401
A.1.114 PDOL	402
A.1.115 PDOL Related Data.....	402
A.1.116 PDOL Values.....	402
A.1.117 Phone Message Table.....	403
A.1.118 POS Cardholder Interaction Information	404
A.1.119 Post-Gen AC Put Data Status	405
A.1.120 Pre-Gen AC Put Data Status	406
A.1.121 Proceed To First Write Flag.....	407
A.1.122 Protected Data Envelope 1	407
A.1.123 Protected Data Envelope 2.....	408
A.1.124 Protected Data Envelope 3.....	408
A.1.125 Protected Data Envelope 4.....	408
A.1.126 Protected Data Envelope 5.....	408
A.1.127 PUNATC(Track1)	409
A.1.128 PUNATC(Track2)	409
A.1.129 Reader Contactless Floor Limit.....	409
A.1.130 Reader Contactless Transaction Limit	410
A.1.131 Reader Contactless Transaction Limit (No On-device CVM)	410
A.1.132 Reader Contactless Transaction Limit (On-device CVM).....	410
A.1.133 Reader CVM Required Limit.....	411
A.1.134 Read Record Response Message Template	411
A.1.135 Reference Control Parameter.....	412
A.1.136 Relay Resistance Accuracy Threshold	412
A.1.137 Relay Resistance Transmission Time Mismatch Threshold	413

A.1.138 Response Message Template Format 1	413
A.1.139 Response Message Template Format 2	413
A.1.140 RRP Counter	414
A.1.141 Security Capability	414
A.1.142 Service Code	414
A.1.143 Signed Dynamic Application Data	415
A.1.144 Static Data Authentication Tag List	415
A.1.145 Static Data To Be Authenticated	415
A.1.146 Tags To Read	416
A.1.147 Tags To Read Yet	416
A.1.148 Tags To Write After Gen AC	417
A.1.149 Tags To Write Before Gen AC	417
A.1.150 Tags To Write Yet After Gen AC	417
A.1.151 Tags To Write Yet Before Gen AC	418
A.1.152 Terminal Action Code – Default	418
A.1.153 Terminal Action Code – Denial	418
A.1.154 Terminal Action Code – Online	418
A.1.155 Terminal Capabilities	419
A.1.156 Terminal Country Code	420
A.1.157 Terminal Expected Transmission Time For Relay Resistance C-APDU 420	
A.1.158 Terminal Expected Transmission Time For Relay Resistance R-APDU 420	
A.1.159 Terminal Identification	421
A.1.160 Terminal Relay Resistance Entropy	421
A.1.161 Terminal Risk Management Data	421
A.1.162 Terminal Type	423
A.1.163 Terminal Verification Results	423
A.1.164 Token Requestor ID	425
A.1.165 Third Party Data	426
A.1.166 Time Out Value	426
A.1.167 Track 1 Data	427
A.1.168 Track 1 Discretionary Data	427
A.1.169 Track 2 Data	428
A.1.170 Track 2 Discretionary Data	428
A.1.171 Track 2 Equivalent Data	429
A.1.172 Transaction Category Code	429
A.1.173 Transaction Currency Code	430

A.1.174 Transaction Currency Exponent	430
A.1.175 Transaction Date	430
A.1.176 Transaction Time.....	430
A.1.177 Transaction Type.....	431
A.1.178 UDOL	431
A.1.179 Unpredictable Number.....	431
A.1.180 Unpredictable Number (Numeric)	432
A.1.181 Unprotected Data Envelope 1.....	432
A.1.182 Unprotected Data Envelope 2.....	432
A.1.183 Unprotected Data Envelope 3.....	433
A.1.184 Unprotected Data Envelope 4.....	433
A.1.185 Unprotected Data Envelope 5.....	433
A.1.186 User Interface Request Data	434
A.2 Data Objects by Tag.....	436
Annex B Kernel State Machine	447
B.1 Application States.....	447
B.2 State Machine	449
Annex C Glossary	451

Figures

Figure 1.1—Symbols Used in Transaction Flow Diagrams	23
Figure 2.1—General Architecture	29
Figure 2.2—Reader Logical Architecture	30
Figure 2.3—Reader Database – Persistent Datasets.....	36
Figure 3.1—Summaries – Basic Principle	50
Figure 4.1—Numbering of Positions within the Discretionary Data.....	80
Figure 4.2—Relation between Discretionary Data and Bitmap.....	80
Figure 6.1—Kernel Started Flow Diagram.....	105
Figure 6.2—State 1 Flow Diagram	109
Figure 6.3—State 2 Flow Diagram	121
Figure 6.4—State 3 Flow Diagram	126
Figure 6.5—State R1 Flow Diagram	144
Figure 6.6—States 3 and R1 – Common Processing – Flow Diagram	154
Figure 6.7—State 4 Flow Diagram	163
Figure 6.8—State 5 Flow Diagram	176
Figure 6.9—State 6 Flow Diagram	183
Figure 6.10—States 4, 5, and 6 – Common Processing – Flow Diagram	187
Figure 6.11—State 7 Flow Diagram	204
Figure 6.12—State 8 Flow Diagram	214
Figure 6.13—States 7 and 8 – Common Processing – Flow Diagram	217
Figure 6.14—State 9 Flow Diagram	225
Figure 6.15—State 12 Flow Diagram	261
Figure 6.16—State 13 Flow Diagram	266
Figure 6.17—State 14 Flow Diagram	282
Figure 6.18—State 15 Flow Diagram	298
Figure 7.1—CVM Selection Flow Diagram.....	304
Figure 7.2—Prepare Generate AC Command Flow Diagram.....	315
Figure 7.3—Processing Restrictions Flow Diagram	330
Figure 7.4—Terminal Action Analysis Flow Diagram	343

Tables

Table 1.1—Terminology.....	20
Table 1.2—Other Notational Conventions.....	26
Table 2.1—Reader Processes.....	30
Table 2.2—Process P Signals.....	31
Table 2.3—Services from Process K.....	33
Table 2.4—Responses from Process K.....	34
Table 2.5—Reader Databases.....	37
Table 2.6—Persistent Dataset Kernel 2.....	38
Table 3.1—Kernel Implementation Options.....	39
Table 3.2—Kernel Database Categories.....	40
Table 3.3—Kernel Configuration Options.....	57
Table 4.1—Access Conditions.....	60
Table 4.2—Configuration Data in TLV Database that Require Default Value.....	69
Table 4.3—Phone Message Table – Default Value.....	70
Table 4.4—CA Public Key Related Data.....	71
Table 4.5—Certification Revocation List Related Data.....	72
Table 4.6—Data Record Detail for EMV Mode Transaction.....	73
Table 4.7—Data Record Detail for Mag-Stripe Mode Transaction.....	75
Table 4.8—Discretionary Data for an EMV Mode Transaction.....	76
Table 4.9—Discretionary Data for a Mag-Stripe Mode Transaction.....	76
Table 5.1—Coding of the Instruction Byte.....	83
Table 5.2—Generic Status Bytes.....	83
Table 5.3—Compute Cryptographic Checksum Command Message.....	84
Table 5.4—Compute Cryptographic Checksum Response Message Data Field.....	85
Table 5.5—Status Bytes for Compute Cryptographic Checksum Command.....	85
Table 5.6—Exchange Relay Resistance Data Command Message.....	86
Table 5.7—Exchange Relay Resistance Data Response Message Data Field.....	86
Table 5.8—Status Bytes for Exchange Relay Resistance Data Command.....	87
Table 5.9—Generate AC Command Message.....	88
Table 5.10—Generate AC Reference Control Parameter.....	89
Table 5.11—Generate AC Response Message Data Field (Format 1).....	90
Table 5.12—Generate AC Response Message Data Field (Format 2) – No CDA....	91
Table 5.13—Generate AC Response Message Data Field (Format 2) – CDA.....	91
Table 5.14—Status Bytes for Generate AC Command.....	92
Table 5.15—Get Data Command Message.....	93
Table 5.16—Supported P1 P2 Values for Get Data Command.....	93
Table 5.17—Status Bytes for Get Data Command.....	94
Table 5.18—Get Processing Options Command Message.....	95
Table 5.19—Get Processing Options Response Message Data Field (Format 1)....	96

Table 5.20—Get Processing Options Response Message Data Field (Format 2)	96
Table 5.21—Status Bytes for Get Processing Options Command.....	97
Table 5.22—Put Data Command Message.....	98
Table 5.23—Supported P1 P2 values for Put Data Command	98
Table 5.24—Status Bytes for Put Data Command	99
Table 5.25—Read Record Command Message.....	100
Table 5.26—P2 of Read Record Command.....	100
Table 5.27—Read Record Response Message Data Field	100
Table 5.28—Status Bytes for Read Record Command	101
Table 6.1—Response Message Data Field.....	180
Table 6.2—Mandatory EMV Mode Data Objects.....	197
Table 6.3—Mandatory Card CDA Data Objects.....	200
Table 6.4—Mandatory Mag-stripe Mode Data Objects.....	210
Table 6.5—ICC Dynamic Data (IDS).....	244
Table 6.6—ICC Dynamic Data (IDS + RRP)	245
Table 6.7—ICC Dynamic Data (No IDS)	246
Table 6.8—ICC Dynamic Data (RRP).....	247

1 Using This Manual

1.1 Purpose

This document, *EMV Contactless Specifications for Payment Systems, Book C-2 – Kernel 2 Specification*, should be read in conjunction with:

- *EMV Contactless Specifications for Payment Systems, Book A – Architecture and General Requirements*, hereafter referred to as [EMV Book A], and
- *EMV Contactless Specifications for Payment Systems, Book B – Entry Point Specification*, hereafter referred to as [EMV Book B].

This document defines the behaviour of the Kernel used in combination with cards supporting a Mastercard brand or cards having a Kernel Identifier indicating Kernel 2, as defined in [EMV Book B].

The Kernel requirements cover both EMV mode and mag-stripe mode contactless transactions.

1.2 Audience

This specification is intended for use by manufacturers of contactless readers and terminals. It may also be of interest to manufacturers of contactless cards and to financial institution staff responsible for implementing financial applications in contactless cards.

1.3 Related Information

The following references are used in this document. The latest version applies unless a publication date is explicitly stated.

Reference	Document Title
[EMV Book 1]	<i>Integrated Circuit Card Specifications for Payment Systems – Book 1, Application Independent ICC to Terminal Interface Requirements</i> , Version 4.4, October 2022
[EMV Book 2]	<i>Integrated Circuit Card Specifications for Payment Systems – Book 2, Security and Key Management</i> , Version 4.4, October 2022
[EMV Book 3]	<i>Integrated Circuit Card Specifications for Payment Systems – Book 3, Application Specification</i> , Version 4.4, October 2022
[EMV Book 4]	<i>Integrated Circuit Card Specifications for Payment Systems – Book 4, Cardholder, Attendant, and Acquirer Interface Requirements</i> , Version 4.4, October 2022
[EMV Book A]	<i>EMV Contactless Specifications for Payment Systems, Book A – Architecture and General Requirements</i> , Version 2.11
[EMV Book B]	<i>EMV Contactless Specifications for Payment Systems, Book B – Entry Point Specification</i> , Version 2.11
[EMV CL L1]	<i>EMV Level 1 Specifications for Payment Systems, EMV Contactless Interface Specification</i> , Version 3.2
[EMV Token]	EMV Payment Tokenisation Specification, Technical Framework, Version 2.3, October 2021
[ISO 639-1]	Codes for the representation of names of languages – Part 1: Alpha-2 Code
[ISO 3166-1]	Codes for the representation of names of countries and their subdivisions – Part 1: Country codes
[ISO 4217]	Codes for the representation of currencies and funds
[ISO/IEC 7813]	Information technology — Identification cards — Financial transaction cards

Reference	Document Title
[ISO/IEC 7816-4]	Identification cards — Integrated circuit(s) cards with contacts — Part 4: Organization, security and commands for interchange
[ISO/IEC 7816-5]	Registration of application providers
[ISO 8583:1987]	Financial transaction card originated messages – Interchange message specifications
[ISO 8583:1993]	Financial transaction card originated messages – Interchange message specifications
[ISO/IEC 8825-1]	Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
[ISO/IEC 8859]	Information technology – 8-bit single-byte coded graphic character sets
[ISO 14443-4]	Identification cards — Contactless integrated circuit(s) cards — Proximity cards — Part 4: Transmission protocol
[ISO 18031:2005]	Information technology – Security techniques – Random bit generation
[NIST SP800-22A]	A statistical test suite for random and pseudorandom number generators for cryptographic algorithms

1.4 Terminology

The following terms are used in this document, carrying specialized meanings as indicated.

Table 1.1—Terminology

Term	Description
Card	Card, as used in these specifications, is a consumer device supporting contactless transactions.
Combination	Combination is the combination of an AID and a Kernel ID.
Configuration Option	A Configuration Option allows activation or deactivation of the Kernel software behind this option. The Configuration Option may change the execution path of the software but it does not change the software itself. A Configuration Option is set in the Kernel database per AID and transaction type.
EMV Mode	Operating mode that indicates that the contactless payment transaction is validated based on EMV minimum data.
Implementation Option	An Implementation Option allows the vendor to select whether the functionality behind the option will be implemented in a particular installation.
Kernel	The Kernel contains the interface routines, security and control functions, and logic to manage a set of commands and responses to retrieve all the necessary data from the Card to complete a transaction. The Kernel processing covers the interaction with the Card between the selection of the card application (excluded) and the processing of the transaction's outcome (excluded).
Mag-stripe Mode	Mag-stripe Mode describes an operating mode of the POS System that indicates that this particular acceptance environment and acceptance rules support magnetic stripe infrastructure. It is typically used in conjunction with the term “transaction” (i.e. Mag-stripe Mode transaction) to indicate contactless payment based on Track 1 and/or Track 2 Data obtained from the Card.

Term	Description
POS System	The POS System is the collective term given to the payment infrastructure present at the merchant. It is made up of the Terminal and Reader.
Process	A Process is a logical component within a Reader that has one or more Queues to receive Signals. The processing of Signals, in combination with the carried data, may then generate other Signals to be sent. Processing can continue until all the Queues of a Process are empty, or until the Process terminates.
Queue	A Queue is a buffer that stores events to be processed. The events are stored in the order received.
Reader	The Reader is the device that supports the Kernel(s) and provides the contactless interface used by the Card. In this specification, the Reader is considered as a separate logical entity, although it can be an integral part of the POS System.
Signal	A Signal is an asynchronous event that is placed in a Queue. A Signal can convey data as parameters, and the data provided in this way is used in the processing of the Signal.
Terminal	The Terminal is the device that connects to the authorization and/or clearing network and that together with the Reader makes up the POS System. The Terminal and the Reader may exist in a single integrated device. However, in this specification, they are considered separate logical entities.

1.5 Notations

This section lists notational conventions used in this specification:

1.5.1 State Machine

This document specifies the Kernel processing as a state machine that is triggered by Signals that cause state transitions. The application states of the Kernel are written in a specific format to distinguish them from the text:

```
state
```

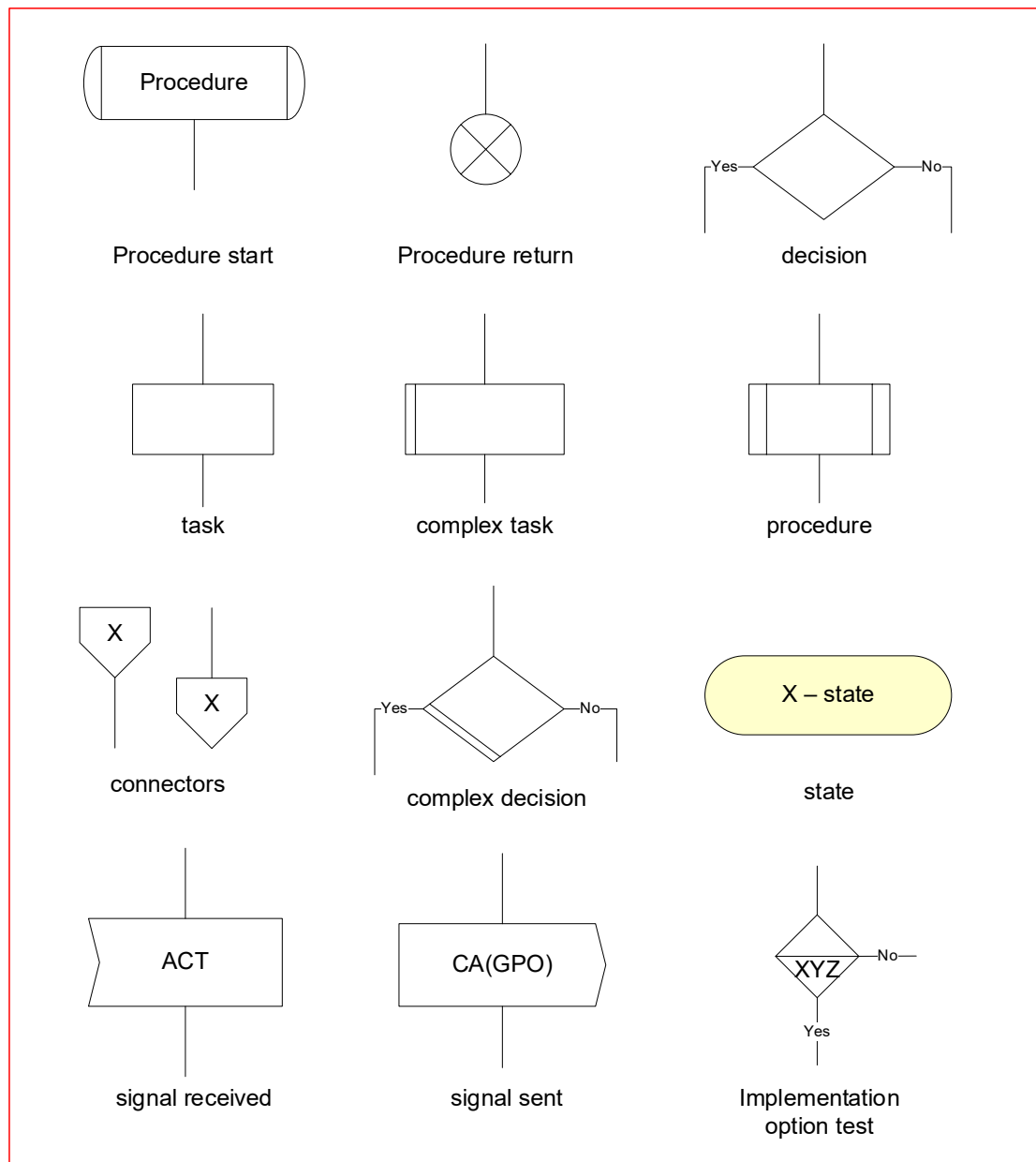
Example:

```
GOTO s4 - waiting for EMV read record response
```

The state machine of the Kernel is represented by means of a state diagram (as shown in Figure B.1).

This document uses a combination of flow diagrams and textual description in order to describe the state transitions in the state machine of the Kernel. Figure 1.1 shows the symbols used in the flow diagrams.

Figure 1.1—Symbols Used in Transaction Flow Diagrams



The combination of the flow diagrams and the corresponding textual descriptions constitute the requirements on the Kernel behaviour:

- Each diagram in this specification has a unique label.
- Each symbol in a diagram has a unique identifier that is the concatenation of the diagram label with the symbol number.
- The textual description corresponding to the symbol in a diagram starts with the identifier of the symbol.

The flow diagrams are read from top to bottom and define the order of execution of the processing steps. The textual description specifies the behaviour of the individual steps but bears no information on the order of execution.

The requirements relate to the behaviour of the Kernel but leave flexibility in the actual implementation. The implementation must behave in a way that is indistinguishable from the behaviour specified in this document. Indistinguishable means that it creates the output as predicted by this specification for a given input. There is no requirement that the implementation realize the behaviour through a state machine as described in this document.

1.5.2 Data Object Notation

Data objects used for this specification are written in a specific font to distinguish them from the text:

Data Object Name

Example:

Application File Locator

Pre-Gen AC Put Data Status

To refer to a sub-element of a data object (i.e. a specific bit, set of bits, or byte of a multi-byte data object), the following notational convention is used:

- If the sub-element is defined in the data dictionary (Annex A), with each possible value of the sub-element having a name, then the following conventions apply:
 - The reference to the sub-element is 'Name of Sub-element' in *Data Object Name*.
 - The reference to the value is VALUE OF SUB-ELEMENT.

Examples:

- 'OD-CVM verification successful' in *POS Cardholder Interaction Information* refers to bit 5 of byte 2 in *POS Cardholder Interaction Information*.

- 'CVM' in *Outcome Parameter Set* := ONLINE PIN means the same as “bits 4 to 1 of byte 4 of *Outcome Parameter Set* are set to 0010b”.
- Alternatively, an index may be used to identify a sub-element of a data object. In this case the following notational conventions apply:
 - To refer to a specific byte of a multi-byte data object, a byte index is used within brackets (i.e. []).

For example, *Terminal Verification Results*[2] represents byte 2 of *Terminal Verification Results*. The first byte (leftmost or most significant) of a data object has index 1.
 - To refer to a specific bit of a single byte multi-bit data object, a bit index is used within brackets [].

For example, *Cryptogram Information Data*[7] represents the 7th bit of the *Cryptogram Information Data*. The first bit (rightmost or least significant) of a data object has index 1.
 - To refer to a specific bit of a multi-byte data object, a byte index and a bit index are used within brackets (i.e. [][]).

For example, *Terminal Verification Results*[2][4] represents bit 4 of byte 2 of the *Terminal Verification Results*.
 - Ranges of bytes are expressed with the x:y notational convention:

For example, *Terminal Verification Results*[1:4] represents bytes 1, 2, 3, and 4 of the *Terminal Verification Results*.
 - Ranges of bits are expressed with the y:x notational convention:

For example, *Cryptogram Information Data*[5:1] represents bits 5, 4, 3, 2, and 1 of the *Cryptogram Information Data*.

1.5.3 Other Notational Conventions

Notations for processing data and managing memory are described in Table 1.2.

Table 1.2—Other Notational Conventions

Symbol	Meaning	Example
'0' to '9' and 'A' to 'F'	Hexadecimal notation. Values expressed in hexadecimal form are enclosed in straight single quotes.	27509 decimal is expressed in hexadecimal as '6B75'.
1001b	Binary notation. Values expressed in binary form are followed by the letter b.	'08' hexadecimal is expressed in binary as 00001000b.
340	Decimal notation. Values expressed in decimal form are not enclosed in single quotes.	'0C' hexadecimal is expressed in decimal as 12.
SET	A specific bit in a data object is set to the value 1b	SET 'CDA failed' in <i>Terminal Verification Results</i>
CLEAR	A specific bit in a data object is set to the value 0b	CLEAR 'Cardholder verification was not successful' in <i>Terminal Verification Results</i>
:=	A specific value is assigned to a data object or to a sub-element of a data object	'Status' in <i>Outcome Parameter Set</i> := END APPLICATION
OR	This notation is used for both the logical and bitwise OR operation. Its meaning is therefore context-specific.	Bitwise AND and OR: IF [((<i>Terminal Action Code</i> – <i>Online</i> OR <i>Issuer Action Code</i> – <i>Online</i>) AND <i>Terminal Verification Results</i>) = '0000000000']
AND	This notation is used for both the logical and bitwise AND operation. Its meaning is therefore context-specific.	Logical AND: IF [IsEmptyList(<i>Data To Send</i>) AND IsEmptyList(<i>Tags To Read</i> Yet)]

Symbol	Meaning	Example
NOT	This notation is used for the logical negation operation.	IF [NOT ParseAndStoreCardResponse(TLV)]
	Two binary data objects are concatenated.	A := 'AB34' B := A 'FFFF' means that B is assigned the value 'AB34FFFF'
IF THEN ELSE ENDIF	This textual description is used to specify decision logic, using the following syntax: IF T THEN GOTO X ELSE GOTO Y ENDIF where T is a statement resulting in true or false and X and Y are symbol identifiers.	IF <i>Amount, Authorized (Numeric) > Reader CVM Required Limit</i> THEN GOTO S456.E25 ELSE GOTO S456.E26 ENDIF
GOTO	A GOTO statement is used to indicate the next step in the following two instances: <ul style="list-style-type: none"> • A decision diamond containing a test whose outcome determines subsequent processing • An off-page reference to another flow diagram 	
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 $A = dn + r$	$54 \bmod 16 = 6$

Symbol	Meaning	Example
$A \text{ div } n$	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 $A = dn + r$	$54 \text{ div } 16 = 3$
$X \oplus Y$	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.	$11001100b \oplus 10101010b = 01100110b$ $1110b \oplus 101010b = 001110b \oplus 101010b = 100100b$
$A := \text{ALG}(K)[X]$	Encryption of a data block X with a block cipher (ALG) using a secret key K. Typical values for ALG are AES, DES, TDES, AES^{-1} , DES^{-1} , and TDES^{-1} .	$T := \text{AES}(K)[M]$
C-APDU	C-APDUs are written in all caps to distinguish them from the text	GET PROCESSING OPTIONS

2 General Architecture

2.1 Introduction

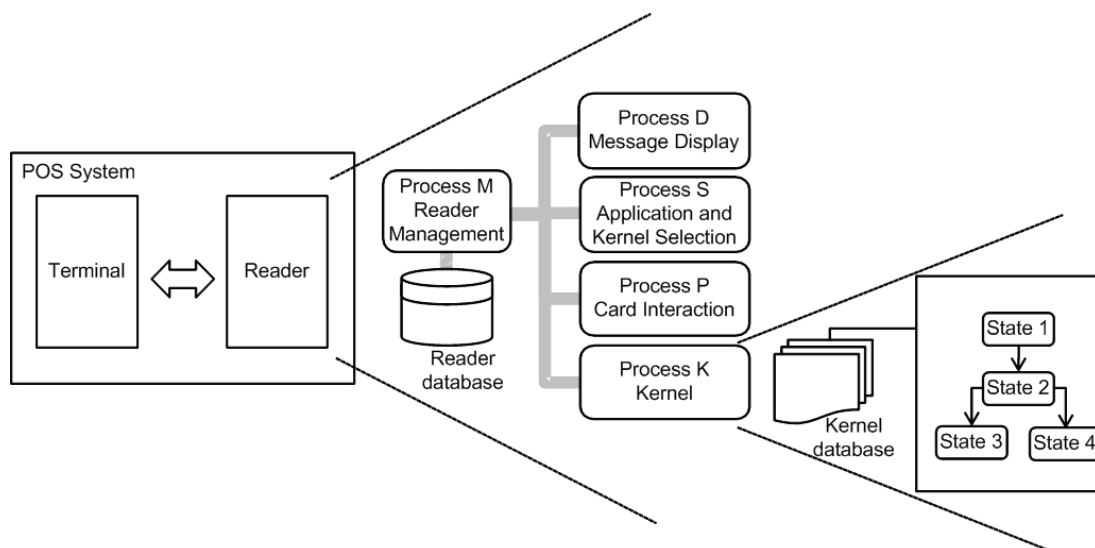
As described in [EMV Book A], the general architecture of a POS System consists of a Terminal and a Reader, where the terms Terminal and Reader refer to a separation in responsibility and functionality between two logical entities.

This document starts from this general architecture, as illustrated in the left hand side of Figure 2.1, then zooms in on the Reader. Figure 2.1 shows how the Reader functionality is allocated to different processes: Process M(ain), Process D(isplay), Process S(elect), Process P(CD), and Process K(ernel).

Communication between different processes happens through Signals stored on Queues. A Signal is an asynchronous event that is placed on a Queue waiting to be processed. A Signal can convey data as parameters, and the data provided in this way is used in the processing of the Signal.

Zooming in further on Process K, Figure 2.1 illustrates the two components of the Kernel: the Kernel software, modelled as a state machine, and the Kernel database, consisting of a number of separate datasets.

Figure 2.1—General Architecture



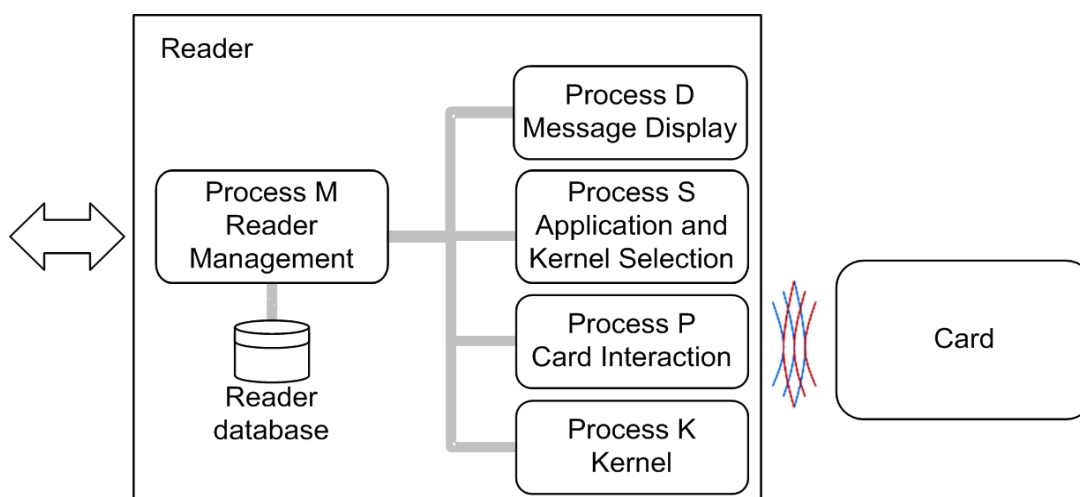
There is no requirement to create devices that use the architecture and the partitioning as laid out in this document, as equally there is no requirement in [EMV Book A] on the partitioning.

The only requirements in this document apply to the Kernel and these requirements define the externally-observable behaviour, independent of the internal organization of the Reader.

2.2 Reader Processes

As illustrated in Figure 2.2, the Reader is modelled as a set of Processes and each Process runs independently of the other processes. The role of the Reader database is explained in section 2.3.

Figure 2.2—Reader Logical Architecture



The different processes are listed in Table 2.1.

Table 2.1—Reader Processes

Process	Responsibility
Process P(CD)	Management of the contactless interface
Process D(isplay)	Management of the user interface
Process S(election)	Selection of the Card Application and Kernel
Process K(ernel)	Interaction with the Card once the application has been selected, covering the payment and data storage transaction flow specific to Kernel 2
Process M(ain)	Overall control and sequencing of the different processes. As part of this role, it is also responsible for the configuration and activation of the Kernel and the processing of its outcome.

The remainder of this section introduces the functionality and configuration of the different processes.

2.2.1 Process P

Process P implements the functionality described in [EMV CL L1] and [ISO 7816-4] and manages the access to the Card.

Process K communicates with Process P through the CA, RA and L1RSP Signals as shown in Table 2.2.

Table 2.2—Process P Signals

Signal In	Signal Out	Comment
CA	RA(R-APDU)	If there is no L1 error, the RA Signal contains the R-APDU sent back in response to a C-APDU.
	L1RSP(code)	If there is an L1 error, L1RSP is returned with code as one of the following: <ul style="list-style-type: none"> • Error – Timeout: An L1 timeout has occurred • Error – Protocol: An L1 protocol error has occurred • Error – Transmission: Any other error

Process P sends the C-APDU included in the CA Signal to the Card and responds with either:

- An RA Signal containing the R-APDU and status bytes returned by the Card, or
- An L1RSP Signal that includes an L1 event such as a timeout, transmission error, or protocol error.

2.2.2 Process D

Process D manages the User Interface Requests as defined in [EMV Book A] and displays a message and/or a status.

A MSG Signal is used as a carrier of the *User Interface Request Data*. Process D may receive MSG Signals from any other Process.

The STOP Signal clears the display immediately and flushes all pending messages.

The MSG and STOP Signals are not acknowledged.

The *User Interface Request Data* can include a message identifier, a status, a hold time, a language preference, and an amount to be displayed.

For more information on the *User Interface Request Data*, please refer to section 7.1 of [EMV Book A].

For displaying messages and/or indicating status, Process D needs the following configuration data:

- Default language
- The currency symbol to display for each currency code and the number of minor units for that currency code
- A number of message strings in the default language and potentially other languages
- A number of status identifiers (and the corresponding audio and LED Signals)

The status identifiers and message identifiers are defined in section 9.2 and section 9.4 respectively of [EMV Book A].

2.2.3 Process S

Process S manages the application and Kernel selection. It runs constantly, and it is waiting for an ACT Signal from Process M. After processing the ACT Signal, it returns the selected Combination of application and Kernel (AID and Kernel ID) and the File Control Information Template and status bytes returned by the Card in response to the final SELECT command in an OUT Signal.

Refer to [EMV Book B] for the specification of an implementation of Process S that supports a multi-kernel architecture.

2.2.4 Process K

The Reader may support multiple Kernels but only one Kernel will execute at a time. The Kernel that is activated depends on the information returned by Process S, which may in turn depend on data retrieved from the Card.

For each transaction, Process K is initialized with a Kernel-specific dataset. Within the different available datasets, the value of the data objects may vary depending on the selected AID and the transaction type. More information on the initialization of the Kernel-specific dataset is provided in section 2.3.

The database for each Kernel can be different and the data items are specific to the Kernel; a payment system or private tag can have a different meaning for different Kernels.

Once the Kernel is selected and configured, it executes as Process K. Using the services of Process P as an intermediary, Process K manages the interaction with the Card application beyond application selection. Upon completion, Process K sends its results to Process M in an OUT Signal and then terminates.

For the remainder of the document, it is assumed that Kernel 2 is selected. More detail on the configuration and initialization of Kernel 2 is provided in section 3.1.

As part of its interaction with the Card, Kernel 2:

- Checks the compatibility between the Kernel settings and the Card settings; these checks include both business (for example transaction type, domestic or international acceptance) and technical (for example versioning) aspects,
- Reads and writes the necessary payment and non-payment related data,
- Determines the need for cardholder verification and the method to be used,
- Performs risk management, resulting in the decision to approve/decline the transaction offline or seek online authorization,
- Requests messages to be displayed depending on the details of the transaction,
- Authenticates data, if and when relevant,
- Informs Process M of the transaction outcome through the OUT Signal.

From the viewpoint of the Reader and depending on the configuration options chosen, Kernel 2 provides two services:

- Through its interaction with the Card, it creates a transaction record for authorization and/or clearing.
- It can interact with the Terminal directly for Data Exchange.

Seen from the Terminal (and again depending on the configuration options chosen), Kernel 2 allows reading and writing data from and to the Card.

The different services are listed in Table 2.3, with the corresponding Signal to call the service indicated in the right column. Only Process M or the Terminal request these services from Process K.

Table 2.3—Services from Process K

Services	Corresponding Signal
Return an authorization or clearing record.	ACT(Data) As a minimum, “Data” includes the <i>File Control Information Template</i> received from the Card in the response to the SELECT command.
Return data from the Kernel database or from the Card. Write data to the Kernel database or to the Card.	DET(Data)

Process K responds to the incoming service request with an outgoing Signal as described in Table 2.4.

Table 2.4—Responses from Process K

Signal In	Signal Out	Comment
ACT	OUT	The OUT Signal includes <ul style="list-style-type: none"> • <i>Outcome Parameter Set</i> • <i>Data Record</i> – if any • <i>Discretionary Data</i> • <i>User Interface Request Data</i> – if any
DET	DEK or n/a	The DEK Signal can be used to request additional data to be provided in a subsequent DET Signal, as well as to provide data that was requested via a configuration setting or a previous DET Signal. The DEK Signal contains <ul style="list-style-type: none"> • The <i>Data Needed</i> data object, which is the list of tags of data items that the Kernel needs from the Terminal • The <i>Data To Send</i> data object, which is the list of tags with data values that the Terminal has requested
n/a	DEK	

Within a transaction, the Terminal can only send one or more DET Signals after receiving a DEK Signal indicating that Process K is active.

The DEK Signal is sent only if the Kernel has data for the Terminal or needs data from the Terminal. The DEK and DET Signals are exchanged as part of the Data Exchange mechanism.

2.2.5 Process M

Process M is responsible for coordinating the other processes to perform a transaction. The overall process is as follows:

1. Process M receives the ACT Signal from the Terminal.
2. Process M starts Process P by sending it an ACT Signal to start polling for cards as described in [EMV CL L1].
3. Process M requests Process D to display the READY message (through a MSG Signal).

4. When Process P indicates to Process M that a Card is detected, Process M activates Process S. When Process S completes successfully, it responds with an OUT Signal with the selected Combination {AID – Kernel ID}, the *File Control Information Template* of the selected Card application, and the status bytes returned by the Card.
5. Based on this information, Process M then configures Process K for the specific *Transaction Type* and AID, using a Kernel-specific dataset, and sends it an ACT Signal containing transactional data such as the *Amount*, *Authorized (Numeric)* and the *File Control Information Template*. When Process K has completed the transaction, it returns an OUT Signal to Process M, including the *Outcome Parameter Set*, *Discretionary Data*, *Data Record* (if any) and optionally a *User Interface Request Data*.
6. Process M analyzes the 'Status' in *Outcome Parameter Set* and executes the instructions encoded in the other fields of the *Outcome Parameter Set*. As required, Process M instructs Process P to perform the removal sequence. Alternatively it may instruct Process S to select the next application on the Card.
7. Process M passes a subset of the *Outcome Parameter Set*, the *Data Record*, and the *Discretionary Data* to the Terminal in the OUT Signal.
8. If the transaction is processed online, the Reader should receive a MSG Signal from the Terminal to indicate whether the transaction was approved or declined.

2.3 The Reader Database

The Reader maintains a database that is divided into different datasets hold in persistent memory. An overview of the different persistent datasets is given in Figure 2.3, with additional details in Table 2.5.

Figure 2.3—Reader Database – Persistent Datasets

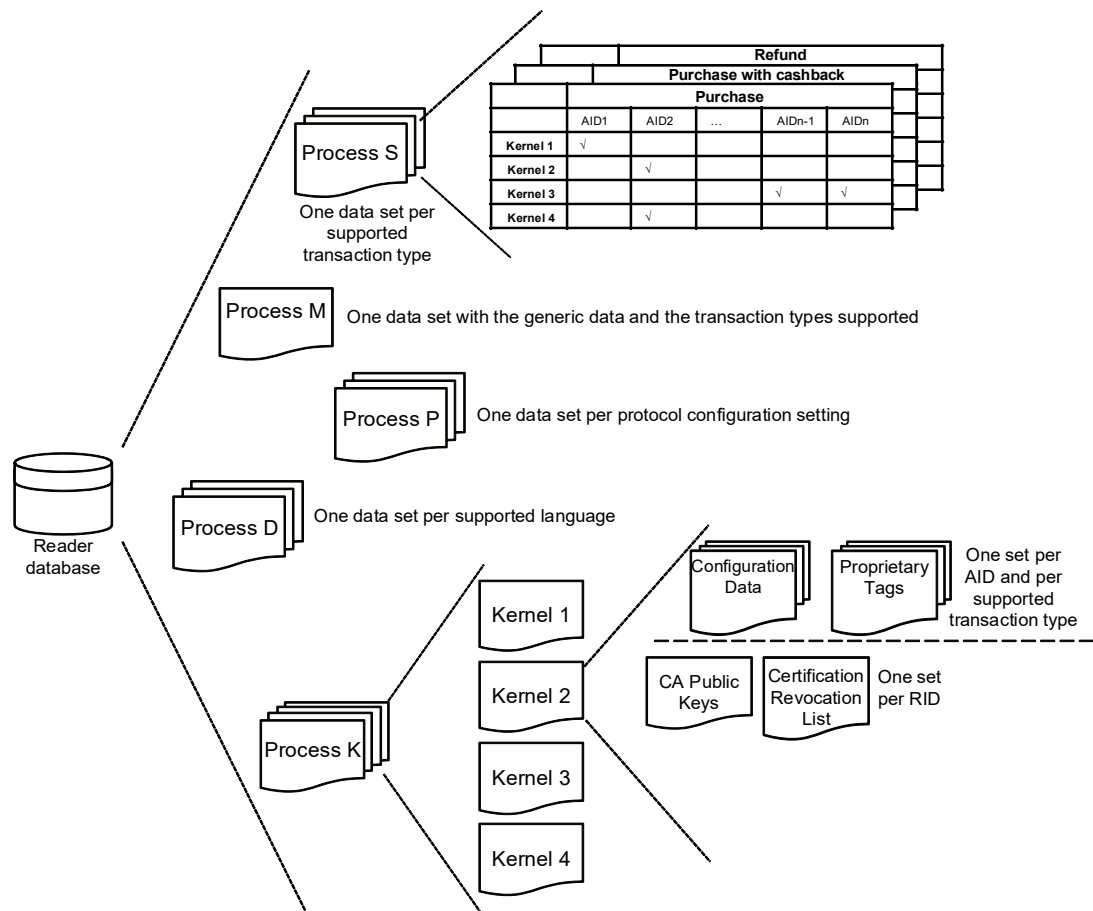


Table 2.5—Reader Databases

Process	Database
Process M	One dataset, including generic data and the different transaction types supported. Examples of generic data are <i>Interface Device Serial Number</i> , <i>Terminal Country Code</i> , <i>Transaction Currency Code</i> , and <i>Transaction Currency Exponent</i> . Examples of transaction types are purchase, purchase with cashback, and refund.
Process P	One or more datasets, one for each protocol configuration setting. Each dataset contains (part of) the configuration settings as defined in Annex A of [EMV CL L1].
Process D	Multiple datasets for Process D, one for each supported language. Each dataset contains the message strings behind the message identifiers.
Process S	Multiple datasets for Process S, one dataset per transaction type. Each dataset contains a list of Combinations {AID – Kernel ID}.
Process K	Multiple Kernel-specific datasets for Process K. Each Kernel-specific dataset includes different subsets. For Kernel 2, see Table 2.6.

If the transaction type has not been indicated by the Terminal in the ACT Signal then a configurable default transaction type is used.

Table 2.6—Persistent Dataset Kernel 2

Subset	Purpose
Configuration Data	Contains the TLV-encoded persistent data objects relevant to a specific AID (used by Process S to identify the application during application selection) and Transaction Type. The values of the TLV-encoded data objects do not vary per transaction.
Proprietary Tags	Data objects with proprietary tags (i.e. data objects with tags not listed in Table A.1). A proprietary tag may have a zero length for data returned from the Card or a length different from zero for terminal sourced data.
The list of CA public keys	<p>Information linked to the CA public keys, including the index, modulus, and exponent.</p> <p>CA public keys can be shared between AIDs that have the same RID and sharing can be done across Kernels.</p> <p>The Reader should be able to store the information for at least six keys per RID.</p>
The Certification Revocation List	A list of Issuer Public Key Certificates that payment systems have revoked for each RID supported by the Kernel. Note that as for the list of CA public keys, entries in the Certification Revocation List may be shared between Kernels where Kernels support the same RID.

3 Reader Process K — Kernel Processing

3.1 Implementation Options

Kernel 2 supports two Implementation Options listed in Table 3.1.

Table 3.1—Kernel Implementation Options

Implementation Option	Description
Mag-stripe mode	This Implementation Option gives the implementer the choice to build Kernel 2 with or without support for mag-stripe mode. This Implementation Option is further on referenced as MAG.
Data Exchange & Data Storage	This Implementation Option gives the implementer the choice to build Kernel 2 with or without support for Data Exchange, IDS and SDS. This Implementation Option is further on referenced as DE/DS.

These two Implementation Options give rise to the following four possible Kernel implementations:

- Both MAG and DE/DS implemented
- MAG implemented and DE/DS not implemented
- MAG not implemented and DE/DS implemented
- Neither MAG, nor DE/DS implemented

3.2 The Kernel Database

When the Kernel process starts, the Kernel database, as introduced in section 2.3, is already initiated with the persistent dataset of Kernel 2 for a specific AID (or RID) and Transaction Type. This includes the Configuration Data, Proprietary Tags, list of CA Public Keys and the Certification Revocation List (see Table 2.6).

At the start of the Kernel, the Kernel instantiates a TLV Database including the tags defined in Table A.1. The TLV Database is extended with the Proprietary Tags (if any). The Kernel copies the Configuration Data in the TLV Database at the start of the transaction. The TLV Database is stored in transient memory, being used as a working database which is cleared after the Kernel exits. Within the TLV Database, entries may exist with zero length.

In addition to the Kernel database, the Kernel receives transaction data items in the ACT Signal. These data items originate from the (Terminal) ACT Signal and from the OUT Signal of the application and Kernel selection process (Process S). These data items with their volatile values are added to the database as well.

During transaction processing, the Kernel may receive events from Process M, the Card, and the Terminal. This input, together with the Kernel's progression through the transaction processing, causes further updates to the Kernel database.

While performing a transaction, the Kernel ensures that updates to the TLV Database are done only by the authorized 'source' (origin) of the data item. For this purpose, data items are put in different categories and the category determines the Signal – and therefore source – that can update data objects within a category.

The different categories and corresponding Signals are illustrated in Table 3.2.

Table 3.2—Kernel Database Categories

Data Category	Signal
Terminal sourced data object – configuration data	n/a
Terminal sourced data object – transaction data	DET, ACT
Kernel defined value or internal data object	n/a Value can only be changed as part of Kernel processing
Card sourced data object	RA ¹

¹ The *File Control Information Template* is received in an ACT Signal but is treated as an RA as that is how it was delivered to Process S.

3.3 Mag-Stripe Mode and EMV Mode

3.3.1 Overall Transaction Flow

Upon receipt of an ACT Signal, the Kernel initiates the transaction on the Card through a GET PROCESSING OPTIONS command.

Based on the response from the Card, in particular the *Application Interchange Profile*, the Kernel continues with either a Mag-stripe Mode or an EMV mode transaction. In both cases, the Kernel reads data record(s) from the Card (through one or more READ RECORD commands). Then the Kernel requests the Card to generate a cryptogram, which is then included in the *Data Record*.

Once all the data from the Card, including the cryptogram, are retrieved, the Kernel indicates that the Card can be removed.

The Kernel completes the transaction by preparing the remainder of the *Data Record*, the *Outcome Parameter Set* information, and *Discretionary Data* (as defined in [EMV Book A]). For an EMV mode transaction, the *Data Record* contains EMV-like data; for a Mag-stripe Mode transaction, it contains mag-stripe-like data. The Kernel returns the above data to the main process (Process M) and this concludes the transaction for the Kernel, which then terminates execution.

The remainder of this section highlights the difference in transaction flow between Mag-stripe Mode and EMV mode transactions.

3.3.2 Mag-stripe Mode

For a Mag-stripe Mode transaction, after the GET PROCESSING OPTIONS command, the Kernel continues with the following steps:

1. It reads the data records from the Card, containing *Track 1 Data* and *Track 2 Data*, together with instruction on how to populate the discretionary data.
2. It issues the COMPUTE CRYPTOGRAPHIC CHECKSUM command, including *Unpredictable Number (Numeric)* to the Card, requesting the Card to return a CVC3 cryptogram, calculated over *Unpredictable Number (Numeric)*.
3. It populates the *Track 2 Data* with the *Unpredictable Number (Numeric)*, the *Application Transaction Counter*, and *CVC3 (Track2)*.
4. If *Track 1 Data* is present, it populates the *Track 1 Data* with the *Unpredictable Number (Numeric)*, the *Application Transaction Counter*, and *CVC3 (Track1)*.

5. It sets *nUN* equal to the (meaningful) length of the *Unpredictable Number (Numeric)* and populates *Track 2 Data* and (if present) *Track 1 Data* with this value.
6. It requests the transaction to be sent online.

3.3.3 EMV Mode

For an EMV mode transaction, after the GET PROCESSING OPTIONS command, the Kernel continues with the following steps:

1. It determines which form of Offline Data Authentication to perform.
2. It reads the data records of the Card (using READ RECORD commands).
3. It performs Terminal Risk Management and Terminal Action Analysis, and selects a cardholder verification method for the transaction.
4. It requests an *Application Cryptogram* from the Card by issuing a GENERATE AC command.
5. It performs Offline Data Authentication as appropriate.

3.4 Data Exchange

3.4.1 Introduction

Terminal and Kernel can communicate through the Data Exchange mechanism if the DE/DS Implementation Option is implemented.

The Kernel can send tagged data to and request data from the Terminal through the DEK Signal.

The Terminal can control the Kernel through the DET Signal by virtue of its ability to:

- Update the current transaction database of the Kernel
- Request tagged data from the Kernel or from the Card
- Manage the transaction flow pace by withholding necessary data (so that the Kernel asks for it) or providing these data earlier than needed to avoid delays.

3.4.2 Sending Data to the Terminal

As part of its configuration or through an ACT or DET Signal, the Kernel has a data object (*Tags To Read*) containing the tags of the data objects to be sent to the Terminal. If a tag refers to Card data, this data is retrieved through READ RECORD commands – as part of reading the records listed in the *Application File Locator* – or through a GET DATA command. The Kernel has a list of data objects that are read using GET DATA (See Table 5.16). All other data objects are read using READ RECORD commands.

Note that *Tags To Read* excludes the IDS data which is sent automatically if IDS is activated in the Kernel.

When the Kernel has completed the (currently outstanding) requests from the Terminal, it sends the data to the Terminal via a DEK Signal.

The information in the DEK Signal may trigger the Terminal to send another list of data to read (DET Signal). This list is then appended to the original list and may result in another set of GET DATA commands if the request includes tags referring to card data.

The Kernel uses a buffer, called *Tags To Read Yet*, to accumulate the different read requests included in *Tags To Read*.

Data To Send is another buffer, accumulating the multiple data that the Kernel has for the Terminal. It is populated with TLV data retrieved in response to *Tags To Read Yet* processing.

The process continues until all records have been read and there are no more data objects in the list that need to be read using a GET DATA command.

3.4.3 Requesting Data from the Terminal

If one of the following data objects is present in the Kernel database with the length of the value field set to zero, then the Kernel sends a DEK Signal to request the value of the data object:

- *Tags To Read*
- *Tags To Write Before Gen AC*
- *Tags To Write After Gen AC*
- *Proceed To First Write Flag*

In more general terms, the Kernel applies the following rules for Terminal sourced data objects (as opposed to Kernel and Card sourced data objects):

1. If the Kernel database contains a Terminal sourced data object that has length of zero and if this data object is needed during the transaction, then the Kernel requests this data object in a DEK Signal by including its tag in *Data Needed*.

The data object can be needed during the transaction for two reasons:

- The Kernel needs it for its own processing, e.g. *Amount*, *Authorized (Numeric)*.
 - The Card requests it in a DOL, e.g. *Merchant Custom Data*.
2. If the data object is not present in the Kernel database, it is not requested from the Terminal. This condition applies only if the Kernel does not need this data object for its own processing. When this data object is requested by the Card in a DOL, it is zero filled in the data of the corresponding command.
 3. If the data object is present with length different from zero, it is not requested from the Terminal. It is sent to the Card when requested in a DOL and normal padding and truncation rules apply.

By putting a Terminal sourced data object or one or more of the data objects listed above in the database with a zero length, the Terminal deliberately withholds the data so that the Kernel specifically asks for it, thereby giving the Terminal the ability to pace the transaction flow and change the value of transaction data, based on information received during the transaction flow.

As indicated above, the Kernel uses a buffer, called *Data Needed*, to accumulate tags that the Kernel needs from the Terminal. It is populated with a list of tags. In a similar manner, if IDS is being used, the Kernel uses DEK Signals to request the data that it needs to complete the transaction.

The Terminal may send multiple DET Signals, each DET Signal containing a *Tags To Write Before Gen AC* or *Tags To Write After Gen AC* data object. The Kernel manages these DET Signals through two buffers: *Tags To Write Yet Before Gen AC* and *Tags To Write Yet After Gen AC*. These buffers are used to accumulate the TLV data objects included in *Tags To Write Before Gen AC* tag and *Tags To Write After Gen AC* tag respectively.

3.4.4 Synchronization

All data are read from the Card before any data are written. To make sure the reading process is completed and that the Terminal has received all required data, the Kernel checks whether it can move to the writing stage.

This check uses the *Proceed To First Write Flag* data object, as introduced in section 3.4.3. The *Proceed To First Write Flag* may take one of the following values:

- When *Proceed To First Write Flag* is absent, the Kernel can move to the writing phase of the transaction.
- When *Proceed To First Write Flag* has length zero, the Kernel requests a value for the *Proceed To First Write Flag* from the Terminal. It waits until the Terminal provides this value before moving to the writing phase.
- When *Proceed To First Write Flag* has value zero, the Kernel waits until the Terminal provides a value different from zero before moving to the writing phase.
- When *Proceed To First Write Flag* has a value different from zero, the Kernel can move to the writing phase of the transaction.

3.5 Data Storage

3.5.1 Introduction

Data storage is an extension of the regular EMV mode transaction flow such that the Card can be used as a scratch pad or mini data store with simple write and read functionality. Data storage does not apply for Mag-stripe Mode transactions and is only supported if the DE/DS Implementation Option is implemented.

Two types of data storage are possible: Standalone Data Storage (SDS) or Integrated Data Storage (IDS). They rely on the Data Exchange mechanism as described in section 3.4.

The Kernel may support one or both data storage methods and is configured accordingly. However, the use of data storage by the Kernel in a given transaction is conditional on the Card's indication of support for data storage. The Card support for SDS and IDS is indicated in the response to the SELECT AID command. The *File Control Information Template* may contain the *Application Capabilities Information* data object which, if present, indicates the support provided for SDS and IDS.

3.5.2 Standalone Data Storage

SDS uses dedicated commands (GET DATA, PUT DATA) for explicit reading and writing of data. It introduces a range of payment system tags ('9F70' to '9F79') for the reading and writing of non-payment data, so that they can be included in *Tags To Read*, *Tags To Write Before Gen AC*, or *Tags To Write After Gen AC* (see section 3.4). The whole range is freely readable using the GET DATA command.

Writing is done using a PUT DATA command without secure messaging, for tags '9F75' to '9F79'. Writing to the tags '9F70' to '9F74' requires secure messaging and is outside the scope of this specification.

The length of the data is variable. The maximum length is implementation specific, and is between 32 and 192 bytes. If present, the *Application Capabilities Information* from the Card indicates the configuration of the SDS tags. The relevant coding is described in the data dictionary (Annex A).

Writing can be done before and after the GENERATE AC, hence two lists to distinguish between data objects to be written to the Card before and those to be written afterwards. This distinction is indicated by the list names: *Tags To Write Before Gen AC* and *Tags To Write After Gen AC*.

Each list is TLV coded, containing Tag, Length as well as Value of the data to write. The lists may be part of the Kernel configuration, or may be communicated to the Kernel during the transaction using Data Exchange, via a DET Signal.

Once the Kernel has the go-ahead to move to writing, it may send one or more PUT DATA commands to the Card, each command containing one data object from the first list (*Tags To Write Before Gen AC*) and in the order as they are in this list. Once all data from this first list are sent to the Card, the Kernel sends the GENERATE AC command.

After the GENERATE AC command, the Kernel then repeats this process for the second list (*Tags To Write After Gen AC*).

3.5.3 Integrated Data Storage

IDS builds the reading and writing functions into existing payment commands (GET PROCESSING OPTIONS and GENERATE AC). The command-response sequence exchanged between the Card and Kernel is therefore unchanged from a normal purchase transaction. It also addresses the security mechanisms of those exchanges.

This section describes the overall transaction flow and the security design.

IDS: Overall Transaction Flow

Support for IDS in the transaction flow can be summarized as follows:

1. Process S selects the application. If the Card supports IDS, this is indicated in the Card's response and the *PDOL* includes the tag of the operator identifier. The Card's response is included in the ACT Signal activating the Kernel, and is therefore part of the current transaction database of the Kernel.
2. The operator's slot is selected through the inclusion of the operator identifier in the GET PROCESSING OPTIONS command data as part of the *PDOL Related Data*.
3. If a slot is currently present for this identifier, the Card returns the contents of the slot in its response to the GET PROCESSING OPTIONS command together with slot management data. If it is not present, the Card indicates whether a new slot is available for allocation to this identifier. As well as the normal *Application Interchange Profile* and *Application File Locator* data objects, the GET PROCESSING OPTIONS response (using Format 2) returns², if available, the following:

² Although not relevant to the reading of the data, note that the GET PROCESSING OPTIONS response also includes a card challenge (*DS Unpredictable Number*). This is part of the IDS security mechanism.

- The non-payment data (*DS ODS Card*)
 - The type of data (*DS Slot Management Control*)
 - A hash of the transaction context calculated by the Card when data was written to the Card in a previous transaction (*DS Summary 1*)
 - An indication of which type of data (volatile or permanent) may be stored (*DS Slot Availability*)
4. The information on the slot data is passed to the Terminal (DEK Signal), which can then decide to update the data or allocate a new slot, as appropriate for the particular transaction. The Terminal passes this information to the Kernel (DET Signal) and the Kernel sends the new data to the Card appended to the end of the *CDOL1* data in the GENERATE AC command.

For this purpose, the Card supports a (single) *DSDOL*, applicable for the GENERATE AC command. *DSDOL* is read through the READ RECORD command, in a record present in the *Application File Locator*. The Kernel appends the (non-payment data) data objects listed in the *DSDOL* in the order as indicated in the *DSDOL* and with the lengths as indicated in *DSDOL* (except for the last element which may be shorter). Except for the last tag in *DSDOL*, all tags are handled according to the rules specified in section 5.4 of [EMV Book 3]. The last tag indicated in *DSDOL* is appended with the length defined in the TLV Database and no padding is applied if the length specified in the *DSDOL* entry is greater than the actual length of the data object in the Kernel database.

The data objects that are included in the *DSDOL* tags list are:

- The type of data (*DS ODS Info*)
 - The result of a one-way function, to set a new access control (*DS Digest H*)
 - The input to a one-way function, to get access control (*DS Input (Card)*)
 - The non-payment data envelope (*DS ODS Term*)
5. Including the additional data in the GENERATE AC command may influence the outcome of the transaction and does not automatically result in a data update or a slot allocation. Whether data will be written to the Card and the outcome of the transaction depends on four elements:
- The type of application cryptogram (i.e. TC, ARQC, or AAC) proposed by the Terminal in the *DS AC Type*
 - The type of application cryptogram resulting from the Kernel (Terminal) risk management and action analysis, indicated in *AC Type*
 - The settings in *DS ODS Info For Reader* sent by the Terminal

- The type of *Application Cryptogram* generated by the Card, as reported in *Cryptogram Information Data* – see step 6

The algorithm is described below and assumes there is an order amongst the different application cryptograms TC, ARQC, and AAC, with TC being the highest and AAC being the lowest (i.e. $TC > ARQC > AAC$). The algorithm is as follows:

The Kernel compares its *AC Type* to the Terminal's *DS AC Type*.

- If the Kernel *AC Type* is higher, then the Kernel sets its *AC Type* equal to the *DS AC Type*, and the Kernel includes the IDS data in the GENERATE AC command data. For example, if the Terminal requests an ARQC in *DS AC Type* and the Kernel's risk management decision results in a TC in *AC Type*, then the Kernel sets its *AC Type* to ARQC, which is lower.
 - If the Kernel *AC Type* is lower, then:
 - If *DS ODS Info For Reader* indicates that the IDS data can be used for *AC Type*, then the Kernel includes the IDS data in the GENERATE AC command data.
 - Otherwise:
 - If *DS ODS Info For Reader* indicates that the transaction may continue without IDS data in the GENERATE AC command data then the Kernel sends the GENERATE AC without IDS data.
 - Otherwise, the Kernel terminates the transaction and returns an OUT Signal.
6. If the IDS data are included in GENERATE AC command data, then the Card may or may not write the data. If the data is written, then the Card confirms to the Kernel that the slot has been allocated and that the new data has been updated. If there is an error with the data or if the type of *Application Cryptogram* generated by the Card is different from that requested by the Kernel, then the Card does not store the data. In any case, the Card response includes an authenticated hash of the transaction context of the initial data read (*DS Summary 2*) as well as a hash of the transaction context of the resulting data (*DS Summary 3*).
7. If the response to the GENERATE AC command indicates that the data were not written, the Kernel checks *DS ODS Info For Reader* on whether the transaction should be continued or not.

IDS: Security Design

The security design is based on the following assumptions and mechanisms.

Assumptions

The service is provided based on the data read (*DS ODS Card*) and is conditional on the data being authentic. If the data cannot be authenticated, then the service will not be provided.

The Terminal has a cryptographic method to add a MAC to the data that it stores in the data written to the Card to ensure that a third party has not tampered with the data.

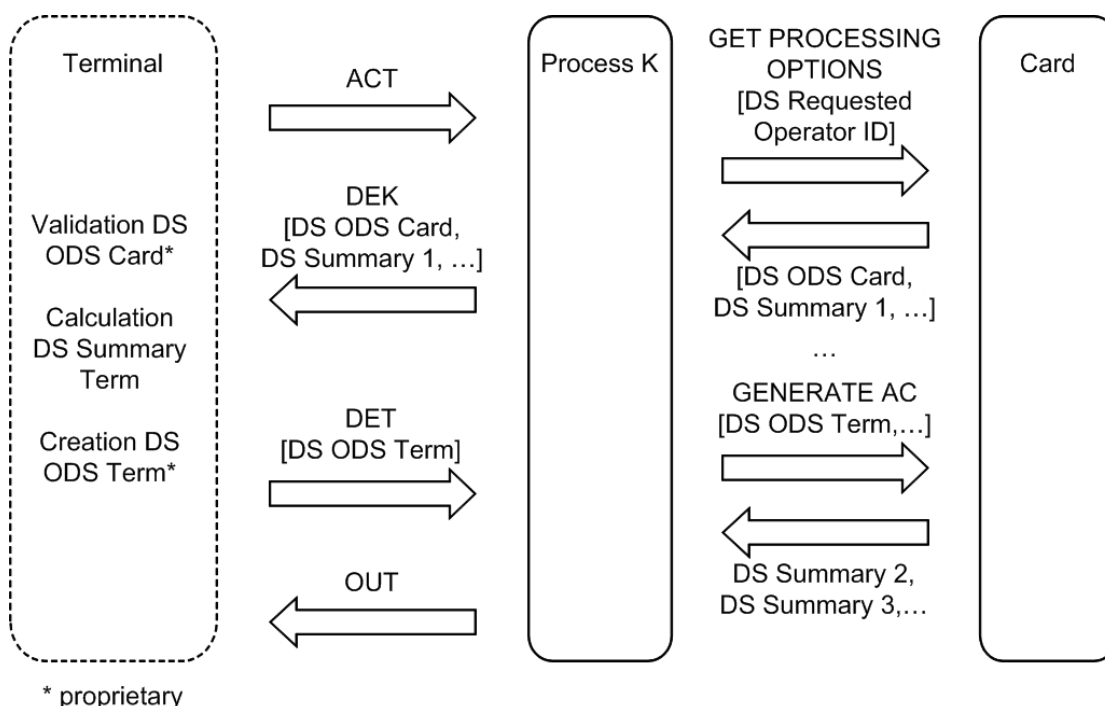
If the Terminal wants to protect the data against skimming and replay, the operator uses the security mechanisms as proposed in this specification.

Mechanisms

The security is built on a combination of the proprietary mechanisms implemented in the Terminal, hashes over the transaction data – called Summaries – and strong offline card authentication using public key cryptography.

The basic principle behind the Summaries is illustrated in Figure 3.1.

Figure 3.1—Summaries – Basic Principle



There is a Summary for the data read and for the data written.

The Summary is a data item that is:

- Computed independently by both the Card (= *DS Summary 3*) and the Terminal at each write operation
- Computed as a one-way function on the identity of the Card and transaction critical data
- Used by the Terminal as input into the (proprietary) security mechanism for protecting its data (= *DS ODS Term*)
- Returned by the Card to the Terminal next time the data is read (= *DS ODS Card*, *DS Summary 1*)
- Included in the CDA signature of the transaction to authenticate the Summaries

Because *DS Summary 1* is returned outside of the CDA signature (and therefore not authenticated), the Card returns the data object in the CDA signature as well, where it is then referred to as *DS Summary 2*.

DS ODS Card and *DS Summary 1* (as well as other data) are returned by the Card and passed to the Terminal. The Terminal validates the authenticity and integrity of *DS ODS Card*, using a proprietary mechanism in combination with *DS Summary 1*.

Assuming that *DS Summary 1* is authentic (which will be confirmed through *DS Summary 2*), the Terminal calculates a Summary over the new transaction data and updates *DS ODS Card*, which then becomes *DS ODS Term*. *DS ODS Term* is sent to the Kernel, which passes it on to the Card.

If the Card updates the slot data with *DS ODS Term*, it calculates a new Summary, taking the existing Summary as input, and stores this new Summary with the slot data. If for some reason the slot data are not updated, no new Summary is calculated and the Summary stored with the slot data does not change.

The Summary stored with the slot data is returned by the Card as *DS Summary 3*.

For the Kernel it is simple to see whether the slot update was successful or not: If the value of *DS Summary 3* is different from the value of *DS Summary 1* (and hence *DS Summary 2*), then the slot data has been updated.

Wedge attacks are detected as both the Card and Reader independently hash critical data into these Summaries. Both of the Summaries calculated by the Card (*DS Summary 2* and *DS Summary 3*) are included in the CDA signature as part of the ICC Dynamic Data. The Kernel will detect tampering with the communication between Terminal and Card when it compares *DS Summary 2* with *DS Summary 1* and *DS Summary 3* with *DS Summary 2*.

Copying and cloning is prevented through inclusion of an authenticated Card identifier (*DS ID*) and a Card challenge (*DS Unpredictable Number*) in the Summary, in combination with the operator's proprietary mechanism for generating a MAC from the data.

For write control, the security is built on a one-way function. At personalization, the Card stores the result of a one-way function over the *DS Input (Card)* data item, which must match the digest that protects the write access to the slot in the Card. Together with the new data, the Terminal provides a new digest (*DS Digest H*) to fit the newly written data (*DS ODS Term*).

3.6 Mobile Transactions

3.6.1 Introduction

Transactions involving mobile phones are different from standard card transactions as the phone can be used to authenticate the cardholder.

For this purpose, the Kernel is able to distinguish between a cardholder device that delegates the CVM processing to the Terminal and a cardholder device that can perform cardholder verification itself. For ease of reference, the latter are often referred to as phones, as this is the most common form factor that supports this functionality. Yet, the distinction between the two types of devices is independent of the form factor and is based on the *Application Interchange Profile*.

If the Kernel is configured not to support on device cardholder verification or if the cardholder device does not indicate support for on device cardholder verification, then the Kernel performs CVM processing based on the *CVM List* for an EMV mode transaction, and delegates the CVM processing to the Terminal for a Mag-stripe Mode transaction.

If on device cardholder verification is supported by both the Kernel and the cardholder device, then the Kernel delegates the CVM processing to the phone and ignores the *CVM List*, if present. The Kernel proceeds as follows:

- It sets the *Reader Contactless Transaction Limit* to the applicable limit for phones.
- If the transaction amount exceeds the *Reader CVM Required Limit*, then the Kernel informs the phone that the transaction amount exceeds the *Reader CVM Required Limit*, expecting the phone to perform CVM processing.

3.6.2 Mobile Mag-stripe Mode Transactions

For the support of mobile Mag-stripe Mode transactions, the Kernel has two mobile specific data objects:

1. The Mobile Support Indicator, indicating that the Kernel supports mobile and that a particular transaction requires CVM
2. A Reader Contactless Transaction Limit (On-device CVM) for phones (as opposed to a Reader Contactless Transaction Limit (No On-device CVM) for cards)

The Kernel also recognizes one additional Card data object, the *POS Cardholder Interaction Information*. When returned by the Card, the *POS Cardholder Interaction Information* indicates whether:

- On-device cardholder verification has been completed successfully.
- The context is conflicting, meaning the cardholder device detected a discrepancy between the data used for a first tap and the data used for a second tap, the first and second tap being both part of the same transaction.
- The application is activated, and if not, how to remedy this and activate the application.
- A button push or On-device cardholder verification (e.g. PIN entry) is required.
- The limits are exceeded or not.

The Kernel checks the *Amount, Authorized (Numeric)* against the *Reader Contactless Transaction Limit* and returns an OUT Signal if the transaction amount is greater than this limit. The OUT Signal includes a Status value of Select Next, to request that the next AID from the candidate list should be selected.

The Kernel then checks whether the transaction amount exceeds the *Reader CVM Required Limit* and, if so, updates the *Mobile Support Indicator* accordingly.

The *Mobile Support Indicator* is then included in the data of the COMPUTE CRYPTOGRAPHIC CHECKSUM command, as part of the data requested in the UDOL.

The response to the COMPUTE CRYPTOGRAPHIC CHECKSUM command includes dynamic CVC3 (*Track2*) and the *POS Cardholder Interaction Information* indicates that CVM has been performed.

For step 5 of section 3.3.2, the Kernel uses a different value for *nUN*. The Kernel offsets the (meaningful) length of the *Unpredictable Number (Numeric)* by 5 (i.e. *nUN* + 5) and uses it to populate *Track 2 Data* and (if present) *Track 1 Data*.

Offsetting *nUN* informs the issuer that CVM was required for this transaction and that CVM processing was delegated to the phone. The issuer verifies whether the CVM processing was correct by checking the correctness of the CVC3 data.

If the COMPUTE CRYPTOGRAPHIC CHECKSUM does not return the CVC3 (*Track2*) data object, the transaction is declined and the Reader uses *POS Cardholder Interaction Information* to inform the customer of the corrective action to take.

3.6.3 Mobile EMV Mode Transactions

For the support of mobile EMV mode transactions, the Kernel uses the *Reader Contactless Transaction Limit* for phones, the *Kernel Configuration* and the *POS Cardholder Interaction Information* as they were introduced in section 3.6.2.

As for a mobile Mag-stripe Mode transaction, the Kernel checks the *Application Interchange Profile* and *Kernel Configuration* data objects and sets the *Reader Contactless Transaction Limit* either equal to the value for phones or to the value used for cards.

If a device identifies itself as one that defers cardholder verification to the device, then CDA is to be used in the GENERATE AC command to avoid fraud.

The Kernel checks the *Amount, Authorized (Numeric)* against the *Reader Contactless Transaction Limit* and returns an OUT Signal if the transaction amount is greater than this limit. The OUT Signal includes a Status value of Select Next, to request that the next AID from the candidate list should be selected.

The Kernel then checks the transaction amount against the *Reader CVM Required Limit*.

If the transaction amount is equal to or below the *Reader CVM Required Limit*, then cardholder verification is not required. If the transaction amount is greater than the limit, then the Kernel sets the *CVM Results* to indicate that on-device cardholder verification was performed (by the ICC) successfully.

The *CVM Results* are included in the GENERATE AC command, as part of the data requested by *CDOL1*.

Once the response to the GENERATE AC command has been received, the Kernel performs offline card authentication.

The response to the GENERATE AC may include the *POS Cardholder Interaction Information*. The Kernel uses this in the case of a decline, to inform the customer to take corrective action.

3.7 Relay Resistance Protocol

3.7.1 Introduction

A relay attack is where a fraudulent terminal is used to mislead an unsuspecting cardholder into transacting, where the actual transaction is relayed via a fraudulent Card (or simulator) to the authentic terminal of an unsuspecting merchant. It may also be that a fraudulent reader is used without the cardholder being aware of the transaction.

3.7.2 Protocol

The relay resistance protocol works as follows:

1. A bit in Application Interchange Profile is used to tell the Reader that the Card supports the relay resistance protocol. A bit in *Kernel Configuration* is used to configure the support of the relay resistance protocol by the Reader.
2. The Reader invokes the relay resistance protocol if both the Card and Reader support it. In this case it sends a timed C-APDU (EXCHANGE RELAY RESISTANCE DATA) to the Card with a random number (*Terminal Relay Resistance Entropy*). The Card responds with a random number (*Device Relay Resistance Entropy*) and timing estimates (*Min Time For Processing Relay Resistance APDU*, *Max Time For Processing Relay Resistance APDU* and *Device Estimated Transmission Time For Relay Resistance R-APDU*).
3. If the timings determined by the Reader exceed the maximum limit computed, the Reader will try again in case there was a communication error or in case other processing on the device interrupted the EXCHANGE RELAY RESISTANCE DATA command processing. The Reader will execute up to two retries.
4. *Terminal Verification Results* are used to permit the Reader to be configured through the Terminal Action Codes to decline or send transactions online in the event that timings are outside the limits computed.
5. The relay resistance protocol relies on CDA. The timings returned by the Card in the EXCHANGE RELAY RESISTANCE DATA are also included in the *Signed Dynamic Application Data* returned in the response to the GENERATE AC command.
If a transaction is completed without CDA, then the Reader cannot trust the outcome of the relay resistance protocol. In case the transaction is not declined offline, additional data is included in the online message (in *Track 2 Equivalent Data*) so that the issuer host may perform the checks.
6. The *Terminal Relay Resistance Entropy* is the same as the *Unpredictable Number*. In the event of retries, new values for the *Unpredictable Number* are computed.

7. The Reader considers a transaction OK if the processing time determined from the measured time is within the window stated by the Card (i.e. *Max Time For Processing Relay Resistance APDU* and *Min Time For Processing Relay Resistance APDU*) In addition some tolerance is given by the reader in the form of a grace period below and above the window defined by the Card (*Minimum Relay Resistance Grace Period* and *Maximum Relay Resistance Grace Period*). The Reader has an accuracy threshold (*Relay Resistance Accuracy Threshold*) that indicates whether the measured time is greater than a reader permitted limit. Another accuracy threshold (*Relay Resistance Transmission Time Mismatch Threshold*) considers the mismatch of the Card communication time.

3.8 Optimisation For Transactions Without CDA

3.8.1 Introduction

When CDA is not used for a transaction, the transaction may be sped up if the public key certificates are not read from the Card. If CDA is not used for a transaction, the Kernel stops reading records from the Card as soon as the minimum data required for the transaction are retrieved. If the Card data are carefully stored in the records, then reading the public key certificates is avoided, speeding up the transaction.

However, in some circumstances, it may be desirable to always read all the records referenced in the *Application File Locator*. So, this optimisation can be deactivated by configuration.

3.8.2 Activation/Deactivation

When 'Read all records even when no CDA' is set in *Kernel Configuration*, the Kernel always reads all the records referenced in the *Application File Locator*.

When 'Read all records even when no CDA' is not set in *Kernel Configuration* and if the transaction is without CDA, the Kernel stops reading the records referenced in the *Application File Locator* as soon as the minimum data required for the transaction are retrieved.

3.9 Kernel Configuration Options

Not all the features have to be activated in each deployment of Kernel 2. Certain features can be deactivated through the Kernel configuration.

The different configuration options are listed in Table 3.3, as well as the method to activate a particular option. If the condition for activation is not satisfied, the option is de-activated. Not all Configuration Options are applicable for all Implementation Options. The last column in the table identifies the Implementation Options for which the Configuration Option is applicable.

Table 3.3—Kernel Configuration Options

Configuration Options	Description	Activation	Implementation Option
IDS	The Kernel supports IDS.	Through the data object <i>DS Requested Operator ID</i> and <i>DSVN Term</i> If <i>DS Requested Operator ID</i> is present (even with a length of zero) and <i>DSVN Term</i> is present with a length different from zero, then IDS is supported.	DS/DE
EMV mode only	The Kernel only supports the EMV mode transaction flow and does not support mag-stripe mode flow.	Through the setting of 'Mag-stripe mode contactless transactions not supported' in <i>Kernel Configuration</i>	MAG
Mag-stripe Mode only	The Reader only supports the mag-stripe mode transaction flow and does not support the EMV mode transaction flow.	Through the setting of 'EMV mode contactless transactions not supported' in <i>Kernel Configuration</i>	MAG

Configuration Options	Description	Activation	Implementation Option
On device cardholder verification	The Kernel supports on device cardholder verification	Through the setting of 'On device cardholder verification supported' in <i>Kernel Configuration</i>	Always
Relay resistance protocol	The Kernel supports the relay resistance protocol	Through the setting of 'Relay resistance protocol supported' in <i>Kernel Configuration</i>	Always
Deactivate optimisation when no CDA	The Kernel always reads all the records referenced in the <i>Application File Locator</i> for transactions without CDA.	Through the setting of 'Read all records even when no CDA' in <i>Kernel Configuration</i> .	Always

All the above configuration options for the Kernel are set at the level of the AID and the transaction type and are part of the TLV Database in the persistent dataset of Kernel 2.

4 Data Organization

This chapter defines the data organization of the Kernel.

4.1 TLV Database

4.1.1 Principles

The Kernel maintains a TLV Database to store all the TLV encoded data objects. This TLV Database is instantiated by the Kernel as described in Section 3.2.. It will be updated during the processing of the transaction. The list of tags included in the TLV Database is specific to the different Implementation Options and is defined in Table A.1.

The TLV Database is updated using information received from a number of sources: at start-up from the Reader, with data from the Card, with data from the Terminal, and with data that results from the Kernel's own processing.

A data object is known by the Kernel if its tag is listed in the data dictionary applicable for the Implementation Option as defined in Table A.1. Other data objects with proprietary tags not listed in Table A.1 may be present in the database at the time of instantiation.

A data object is considered to be present if its tag appears in the TLV Database (length may be zero).

A data object is empty if it is present and its length is zero. A data object is not empty if it is present and its length is greater than zero.

Data objects in the TLV Database have a name, a tag, a length, and a value; for example:

Name:	<i>Amount, Authorized (Numeric)</i>
Tag:	'9F02'
Length:	6
Value:	'000000002345'

The index to access data objects in the TLV Database is the tag. The list of tags known by the Kernel is fixed and is defined by the tags of the TLV encoded data objects in Table A.1.

The name of the TLV encoded data object is also used to represent the value field. The following example initializes the value field of the *Terminal Verification Results* to zero:

Terminal Verification Results := '0000000000'

4.1.2 Access Conditions

Data objects in the TLV Database are assigned access conditions as described in Table 4.1.

Table 4.1—Access Conditions

Access Condition	Description
ACT/DET	<p>These data objects are transaction related data objects sent to the Kernel by the Terminal with the ACT and DET Signals. They may also be present in the TLV Database when the Kernel is instantiated.</p> <p>Proprietary data objects (i.e. data objects with tags not listed in Table A.1) can be updated with the ACT and DET Signals if, and only if, their length at instantiation is different from zero.</p>
RA	<p>These data objects are transaction related data objects sent to the Kernel by the Card with the RA Signal. Proprietary data objects can be updated with the RA Signal if, and only if, their length at instantiation is equal to zero.</p> <p>An exception is data objects contained in the <i>File Control Information Template</i> which are passed to the Kernel with the ACT Signal, but which have the RA access condition assigned.</p>
K	<p>All data objects in the TLV Database can be updated by the Kernel. Every data object has the K (Kernel) access condition assigned.</p>

All data objects can be read by the Card (via a DOL) and by the Terminal (via *Tags To Read*).

4.1.3 Services

Services available to interrogate and manipulate the TLV Database are the following:

Boolean IsKnown(T)

Returns TRUE if tag T is defined in the data dictionary of the Kernel applicable for the Implementation Option. Table A.1 defines the content of the data dictionary for each Implementation Option.

Boolean IsPresent(T)

Returns TRUE if the TLV Database includes a data object with tag T.

Note that the length of the data object may be zero.

Note also that proprietary data objects that are not known can be present if they have been provided in the TLV Database at Kernel instantiation. In this case the IsKnown() service returns FALSE and the IsPresent() service returns TRUE.

Boolean IsNotPresent(T)

Returns TRUE if the TLV Database does not include a data object with tag T.

Boolean IsNotEmpty(T)

Returns TRUE if all of the following are true:

- The TLV Database includes a data object with tag T.
- The length of the data object is different from zero.

Boolean IsEmpty(T)

Returns TRUE if all the following are true:

- The TLV Database includes a data object with tag T.
- The length of the data object is zero.

T TagOf(DataObjectName)

Returns the tag of the data object with name DataObjectName.

Initialize(T)

Initializes the data object with tag T with a zero length. After initialization the data object is present in the TLV Database.

DBObject GetTLV(T)

Retrieves the TLV encoded data object with tag T from the TLV Database.
Returns NULL if the TLV Database does not include a data object with tag T.

Length GetLength(T)

Retrieves from the TLV Database the length in bytes of the data object with tag T.
Returns NULL if the TLV Database does not include a data object with tag T.

Boolean ParseAndStoreCardResponse(TLV String)

TLV Encoding Error := FALSE

Parse TLV String according the Basic Encoding Rules in [ISO/IEC 8825-1] and set TLV Encoding Error to TRUE if parsing error.

If TLV String is not a single constructed or primitive data object then set TLV Encoding Error to TRUE.

IF [TLV Encoding Error]

THEN

 Return FALSE

ELSE

 FOR every primitive TLV in TLV String

 {

 IF [NOT (IsKnown(T) AND

 class of T is Private class³ AND

 NOT update conditions of T include RA Signal)]

 THEN

³ As defined in Annex B of [EMV Book 3], the tag is Private class if bits b7 and b8 of the first byte of the tag are both set to 1b.


```
IF      [IsKnown(T)]
THEN
    IF      [(IsNotPresent(T) OR IsEmpty(T))
              AND
              update conditions of T include RA Signal
              AND
              L is within the range specified by Length field of the data
              object with tag T in the data dictionary in Annex A
              AND
              TLV is included in the correct template (if any) within
              TLV String4
            ]
    THEN
        Store LV in the TLV Database for tag T
    ELSE
        Return FALSE
    ENDIF
ELSE
    IF      [IsPresent(T)]
    THEN
        IF      [IsEmpty(T) AND update conditions of T include RA
                  Signal]
        THEN
            Store LV in the TLV Database for tag T
        ELSE
            Return FALSE
        ENDIF
    ENDIF
ENDIF
ENDIF
}
Return TRUE
ENDIF
```

⁴ Data objects must be included in the correct template as indicated in the data dictionary in Annex A. Data objects for which no template is indicated ("–") must not be returned in a template from the card.

UpdateWithDetData(Terminal Sent Data)⁵

Copies all incoming data (Terminal Sent Data) to the Kernel TLV Database if update conditions allow.

Note that individual data objects contained within lists in Terminal Sent Data are not stored in the database.

FOR every TLV in Terminal Sent Data

```
{  
  IF    [(IsKnown(T) OR IsPresent(T)) AND  
         update conditions of T include DET Signal]  
  THEN  
    Store LV in the TLV Database for tag T  
  ENDIF  
}  
IF [Terminal Sent Data includes Tags To Read]  
THEN  
  AddListToList(Tags To Read, Tags To Read Yet)  
ENDIF  
IF [Terminal Sent Data includes Tags To Write Before Gen AC]  
THEN  
  AddListToList(Tags To Write Before Gen AC, Tags To Write Yet Before Gen AC)  
ENDIF  
IF [Terminal Sent Data includes Tags To Write After Gen AC]  
THEN  
  AddListToList(Tags To Write After Gen AC, Tags To Write Yet After Gen AC)  
ENDIF
```

⁵ Only implemented for the DE/DS Implementation Option

4.1.4 DOL Handling

TLV encoded data objects moved from the Kernel to the Card are identified by a DOL sent to the Kernel by the Card.

DOLs used in this specification are processed as follows:

- *CDOL1*, *PDOL*, and *UDOL*

DOL handling must be performed according to the rules specified in section 5.4 of [EMV Book 3].

- *DSDOL*⁶

All entries except the last must be handled according to the rules specified in section 5.4 of [EMV Book 3].

The last entry in DSDOL must be handled according to the rules specified in section 5.4 of [EMV Book 3], unless the length specified in this entry is greater than the actual length of the data object in the TLV Database. In this case, no padding must be applied and the value must be appended with the length defined in the TLV Database.

Note that tags in a DOL that exist in the TLV Database with zero length are still handled according to the rules specified in section 5.4 of [EMV Book 3], but in addition in case the DE/DS Implementation Option is implemented, any such data objects get requested from the Terminal so that the Terminal is afforded the opportunity to furnish a value for these data objects.

⁶ Only when DE/DS is implemented.

4.2 Working Variables

The Kernel makes use of a number of working variables that are not stored in the TLV Database. They are managed by the Kernel in an implementation specific way.

Working variables can be:

- Local

The lifetime of local working variables is limited to the state transition process or procedure in which they are defined. These data objects do not appear in the data dictionary.

- Global

The lifetime of global working variables is the same as the lifetime of the Kernel process. Global working variables are listed in the data dictionary without a tag.

These data objects are managed by the Kernel itself.

Global working variables can only be read and written by internal processing of the Kernel.

4.3 List Handling

Data is passed between the Kernel and other entities within Signals. The data within the Signals contain a list of tags, in order to request data, or a list of data objects in response to a request.

Each list has a unique name, and acts as a container for a collection of ListItems. A ListItem is a single element in a List. A ListItem is a tag in a list of tags or a data object in a list of data objects.

The following lists of tags are supported if DE/DS is implemented:

- *Tags To Read*
- *Tags To Read Yet*
- *Data Needed*

The following lists of TLV encoded data objects are supported:

- *Tags To Write After Gen AC⁷*
- *Tags To Write Before Gen AC⁷*
- *Tags To Write Yet After Gen AC⁷*
- *Tags To Write Yet Before Gen AC⁷*
- *Data To Send⁷*
- *Data Record*
- *Discretionary Data*

The following methods are used to manipulate lists.

Initialize(List)

Initializes a List. This creates the List structure if it does not exist, and initializes its contents to be empty, i.e. the List contains no ListItems. This method can be called at any time during the operation of the Kernel in order to clear and reset a list.

AddToList(ListItem, List)

If ListItem is not included in List, then adds ListItem to the end of List.

Updates ListItem if it is already included in the List.

RemoveFromList(ListItem, List)

Removes ListItem from the List if ListItem is present in List. Ignores otherwise.

⁷ Only when DE/DS is implemented.

AddListToList(List1, List2)

Adds the ListItems in List1 that are not yet included in List2 to the end of List2.

Updates ListItems that are already included in List2.

ListItem GetAndRemoveFromList(List)

Removes and returns the first ListItem from List. Returns NULL if List is empty.

T GetNextGetDataTagFromList(List)⁸

Removes and returns the first tag from a list of tags that is categorized as being available from the Card using a GET DATA command.

If no tag is found, NULL is returned.

Boolean IsEmptyList(List)

Returns TRUE if List contains no ListItems.

Boolean IsNotEmptyList(List)

Returns TRUE if List contains ListItems.

⁸ Only when DE/DS is implemented.

4.4 Configuration Data

At the time of instantiation of the Kernel the data objects listed in this section are initialized.

4.4.1 Configuration Data – TLV Database

Configuration data objects in the TLV Database should receive a value at instantiation of the Kernel.

The data objects listed in Table 4.2 are the configuration data objects that must be present for the Kernel to work properly. If these data objects are not present at instantiation, a default value must be stored in the TLV Database.

Dependent on the chosen Implementation Options some data objects listed in Table 4.2 may not be present in the Configuration Data as specified in Table A.1.

Table 4.2—Configuration Data in TLV Database that Require Default Value

Data Object	Default Value
<i>Additional Terminal Capabilities</i>	'0000000000'
<i>Application Version Number (Reader)</i>	'0002'
<i>Card Data Input Capability</i>	'00'
<i>CVM Capability – CVM Required</i>	'00'
<i>CVM Capability – No CVM Required</i>	'00'
<i>Default UDOL</i>	'9F6A04'
<i>Hold Time Value</i>	'0D'
<i>Kernel Configuration</i>	'00'
<i>Kernel ID</i>	'02'
<i>Mag-stripe Application Version Number (Reader)</i>	'0001'
<i>Mag-stripe CVM Capability – CVM Required</i>	'F0'
<i>Mag-stripe CVM Capability – No CVM Required</i>	'F0'
<i>Message Hold Time</i>	'000013'
<i>Maximum Relay Resistance Grace Period</i>	'0032'
<i>Minimum Relay Resistance Grace Period</i>	'0014'
<i>Phone Message Table</i>	See Table 4.3
<i>Reader Contactless Floor Limit</i>	'000000000000'

Data Object	Default Value
<i>Reader Contactless Transaction Limit (No On-device CVM)</i>	'000000000000'
<i>Reader Contactless Transaction Limit (On-device CVM)</i>	'000000000000'
<i>Reader CVM Required Limit</i>	'000000000000'
<i>Relay Resistance Accuracy Threshold</i>	'012C'
<i>Relay Resistance Transmission Time Mismatch Threshold</i>	'32'
<i>Security Capability</i>	'00'
<i>Terminal Action Code – Default</i>	'840000000C'
<i>Terminal Action Code – Denial</i>	'840000000C'
<i>Terminal Action Code – Online</i>	'840000000C'
<i>Terminal Country Code</i>	'0000'
<i>Terminal Expected Transmission Time For Relay Resistance C-APDU</i>	'0012'
<i>Terminal Expected Transmission Time For Relay Resistance R-APDU</i>	'0018'
<i>Terminal Type</i>	'00'
<i>Time Out Value</i>	'01F4'
<i>Transaction Type</i>	'00'

Table 4.3 gives the default value of the *Phone Message Table* for the current definition of the *POS Cardholder Interaction Information*.

Table 4.3—Phone Message Table – Default Value

PCII Mask	PCII Value	Message Identifier	Status
'000001'	'000001'	SEE PHONE	NOT READY
'000800'	'000800'	SEE PHONE	NOT READY
'000400'	'000400'	SEE PHONE	NOT READY
'000100'	'000100'	SEE PHONE	NOT READY
'000200'	'000200'	SEE PHONE	NOT READY
'000000'	'000000'	DECLINED	NOT READY

4.4.2 CA Public Key Database

The Kernel has access to a CA Public Key Database containing the CA Public Keys applicable for the RID of the selected AID. This CA Public Key Database is made available to the Kernel and is read-only.

The CA Public Key Index uniquely identifies the CA Public Key in the CA Public Key Database.

Table 4.4 lists the set of data objects that must be available in the CA Public Key Database for each CA Public Key.

Table 4.4—CA Public Key Related Data

Field Name	Length	Description	Format
CA Public Key Index	1	Identifies the CA Public Key in conjunction with the RID	b
CA Hash Algorithm Indicator	1	Identifies the hash algorithm used to produce the Hash Result in the digital signature scheme	b
CA Public Key Algorithm Indicator	1	Identifies the digital signature algorithm to be used with the CA Public Key	b
CA Public Key Modulus	var. (max 248)	Value of the modulus part of the CA Public Key	b
CA Public Key Exponent	1 or 3	Value of the exponent part of the CA Public Key, equal to 3 or $2^{16} + 1$	b
CA Public Key Check Sum (Only necessary if used to verify the integrity of the CA Public Key)	20	A check value calculated on the concatenation of all parts of the CA Public Key (RID, CA Public Key Index, CA Public Key Modulus, CA Public Key Exponent) using SHA-1	b

4.4.3 Certification Revocation List

The Kernel has access to a CRL applicable for the RID of the selected AID. This CRL is made available to the Kernel and is read-only.

Table 4.5 lists the set of data objects that must be available in the CRL for each revoked certificate. If, during CDA, a concatenation of the *CA Public Key Index (Card)* and the Certificate Serial Number recovered from the *Issuer Public Key Certificate* is on this list, then CDA fails.

Table 4.5—Certification Revocation List Related Data

Field Name	Length	Description	Format
CA Public Key Index	1	Identifies the CA Public Key in conjunction with the RID	b
Certificate Serial Number	3	Number unique to this certificate assigned by the certification authority	b
Additional Data	var.	Optional Terminal proprietary data, such as the date the certificate was added to the revocation list	b

4.5 Lists of Data Objects in OUT

This section specifies the lists of data objects included in the OUT Signal.

4.5.1 Outcome Parameter Set

This data object is used to indicate to the Terminal the outcome of the transaction processing by the Kernel. Its content is described in section A.1.110.

4.5.2 User Interface Request Data

Depending on the outcome of the transaction, the *User Interface Request Data* may be included in the OUT Signal to communicate a message to be shown on the Terminal display. The *User Interface Request Data* may also be sent by the Kernel in a MSG Signal. Its format is described in section A.1.186.

4.5.3 Data Record

Depending on the outcome of the transaction, the Kernel may provide the Terminal with an OUT Signal including a *Data Record* that contains the necessary data objects for authorization and clearing. The *Data Record* is a list of data objects. Its content depends on the transaction profile.

The *Data Record* for an EMV mode transaction is as shown in Table 4.6. The *Data Record* for a Mag-stripe Mode transaction is as shown in Table 4.7.

Table 4.6—Data Record Detail for EMV Mode Transaction

Data Object
<i>Amount, Authorized (Numeric)</i>
<i>Amount, Other (Numeric)</i>
<i>Application Cryptogram</i>
<i>Application Expiration Date</i>
<i>Application Interchange Profile</i>
<i>Application Label</i>
<i>Application PAN</i>
<i>Application PAN Sequence Number</i>
<i>Application Preferred Name</i>
<i>Application Transaction Counter</i>

Data Object
<i>Application Usage Control</i>
<i>Application Version Number (Reader)</i>
<i>Cryptogram Information Data</i>
<i>CVM Results</i>
<i>DF Name</i>
<i>Interface Device Serial Number</i>
<i>Issuer Application Data</i>
<i>Issuer Code Table Index</i>
<i>Payment Account Reference</i>
<i>Terminal Capabilities</i>
<i>Terminal Country Code</i>
<i>Terminal Type</i>
<i>Terminal Verification Results</i>
<i>Token Requestor ID</i>
<i>Track 2 Equivalent Data</i>
<i>Transaction Category Code</i>
<i>Transaction Currency Code</i>
<i>Transaction Date</i>
<i>Transaction Type</i>
<i>Unpredictable Number</i>

Table 4.7—Data Record Detail for Mag-Stripe Mode Transaction

Data Object
<i>Application Label</i>
<i>Application Preferred Name</i>
<i>DF Name</i>
<i>Issuer Code Table Index</i>
<i>Mag-stripe Application Version Number (Reader)</i>
<i>Payment Account Reference</i>
<i>Track 1 Data</i>
<i>Track 2 Data</i>

The following methods are used to create the *Data Record*:

```
CreateEMVDataRecord ()
    Initialize(Data Record)
    FOR every Data Object in Table 4.6
    {
        IF      [IsPresent(TagOf(Data Object))]
        THEN
            AddToList(GetTLV(TagOf(Data Object)), Data Record)
        ENDIF
    }
```

```
CreateMSDataRecord ()9
    Initialize(Data Record)
    FOR every Data Object in Table 4.7
    {
        IF      [IsPresent(TagOf(Data Object))]
        THEN
            AddToList(GetTLV(TagOf(Data Object)), Data Record)
        ENDIF
    }
```

⁹ CreateMSDataRecord() is a function specific to Mag-stripe Mode and is only implemented for the MAG Implementation Option.

4.5.4 Discretionary Data

The Kernel always includes *Discretionary Data* in the OUT Signal. The *Discretionary Data* is a list of data objects. Its content depends on the transaction profile.

The *Discretionary Data* for an EMV mode transaction is as shown in Table 4.8. Dependent on the chosen Implementation Options some data objects may not be present in Table 4.8 as specified in Table A.1. The *Discretionary Data* for a mag-stripe mode transaction is as shown in Table 4.9.

Table 4.8—Discretionary Data for an EMV Mode Transaction

Data Object
<i>Application Capabilities Information</i>
<i>Application Currency Code</i>
<i>DS Summary 3</i> ¹⁰
<i>DS Summary Status</i> ¹⁰
<i>Error Indication</i>
<i>Post-Gen AC Put Data Status</i> ¹⁰
<i>Pre-Gen AC Put Data Status</i> ¹⁰
<i>Third Party Data</i>

Table 4.9—Discretionary Data for a Mag-Stripe Mode Transaction

Data Object
<i>Application Capabilities Information</i>
<i>DD Card (Track1)</i>
<i>DD Card (Track2)</i>
<i>Error Indication</i>
<i>Third Party Data</i>

¹⁰ Only when DE/DS is implemented.

The following methods are used to create the *Discretionary Data*:

CreateEMVDiscretionaryData ()

 Initialize(*Discretionary Data*)

 FOR every Data Object in Table 4.8

 {

 IF [IsPresent(TagOf(Data Object))]

 THEN

 AddToList(GetTLV(TagOf(Data Object)), *Discretionary Data*)

 ENDIF

 }

CreateMSDiscretionaryData ()¹¹

 Initialize(*Discretionary Data*)

 FOR every Data Object in Table 4.9

 {

 IF [IsPresent(TagOf(Data Object))]

 THEN

 AddToList(GetTLV(TagOf(Data Object)), *Discretionary Data*)

 ENDIF

 }

¹¹ CreateMSDiscretionaryData() is a function specific to mag-stripe mode and is only implemented for the MAG Implementation Option.

4.6 Data Object Format

4.6.1 Format

All data objects known to the Kernel (other than local working variables) are listed in the data dictionary in Annex A. All the length indications in the data dictionary are given in number of bytes. Data object formats are binary (b), numeric (n), compressed numeric (cn), alphanumeric (an), or alphanumeric special (ans).

Data objects that have the numeric (n) format are BCD encoded, right justified with leading hexadecimal zeros. Data objects that have the compressed numeric (cn) format are BCD encoded, left justified, and padded with trailing 'F's. Note that the length indicator in the numeric and compressed numeric format notations (e.g. n 4) specifies the number of digits and not the number of bytes.

Data objects that have the alphanumeric (an) or alphanumeric special (ans) format are ASCII encoded, left justified, and padded with trailing hexadecimal zeros.

Data objects that have the binary (b) format consist of either unsigned binary numbers or bit combinations that are defined in the specification.

When moving data from one entity to another (for example Card to Reader) or when concatenating data, the data must always be passed in decreasing order, regardless of how it is stored internally. The leftmost byte (byte 1) is the most significant byte.

Data objects are TLV encoded in the following cases:

- Data objects sent from the Card to the Kernel (RA Signal)
- Data objects sent to the Kernel at instantiation or with the ACT and DET Signals
- Data objects sent to the Terminal included in *Data To Send*
- Data objects included in the MSG and OUT Signals

4.6.2 Format Checking

It is the responsibility of the issuer to ensure that data in the Card is of the correct format. No format checking other than that specifically defined is mandated for the Kernel.

However, if during normal processing it is recognized that data read from the Card or provided by the Terminal is incorrectly formatted, the Kernel must perform the processing described in this section.

Other than exceptions specifically defined in this document, data object formatting that does not comply with the requirements in section 12.2.4 of [EMV Book 1] and sections 7.5 and 10.5 of [EMV Book 3] can be considered as a format error.

If a format error is detected in data received from the Card, the Kernel must update the *Error Indication* data object as follows:

'L2' in *Error Indication* := CARD DATA ERROR

If a format error is detected in data received from the Terminal, the Kernel must update the *Error Indication* data object as follows:

'L2' in *Error Indication* := TERMINAL DATA ERROR

The Kernel must then process the exception according to the state in which it occurs, as described here.

States 1, 2, 3, R1, 4, 5, 6, 7, and 8

The Kernel must prepare the *User Interface Request Data*, the *Discretionary Data* and the *Outcome Parameter Set* and send an OUT Signal (as shown here):

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

'Hold Time' in *User Interface Request Data* := *Message Hold Time*

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

Initialize(*Discretionary Data*)

AddToList(GetTLV(TagOf(*Error Indication*)), *Discretionary Data*)

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*))) Signal

The Kernel must then exit.

States 9

The Kernel must process the error as “Invalid Response – 1”, as described under connector C in Figure 6.14.

State 13

The Kernel must process the error as “Invalid Response”, as described under connector A in Figure 6.16.

State 14

The Kernel must process the error as “Invalid Response”, as described under connector A in Figure 6.17.

4.7 Bitmaps Used in Discretionary Data

Mag-stripe Mode transactions use bitmaps to indicate positions in the discretionary data field. These bitmaps are used when the Reader needs to put data into one of the discretionary data fields. The bits indicate the positions into which certain data should be placed.

Figure 4.1 indicates the numbering of the different positions in the discretionary data. In this example there are m positions within the discretionary data field, labeled p_1 to p_m .

Figure 4.1—Numbering of Positions within the Discretionary Data

Discretionary Data									
p_m	p_{m-1}	p_{m-2}	p_{m-3}	...	p_5	p_4	p_3	p_2	p_1

Each bit in the bitmap refers to a position in the discretionary data. The least significant bit of the bitmap, i.e. the rightmost bit b_1 , corresponds to position p_1 ; as indicated in Figure 4.2.

Figure 4.2—Relation between Discretionary Data and Bitmap

Discretionary Data									
p_m	p_{m-1}	p_{m-2}	p_{m-3}	...	p_5	p_4	p_3	p_2	p_1

b_r	b_{r-1}	b_{r-2}	...	b_{m+1}	b_m	b_{m-1}	b_{m-2}	b_{m-3}	...	b_5	b_4	b_3	b_2	b_1
Bitmap														

The bitmap is composed of a number of bytes, and therefore the number of bits in the bitmap is always a multiple of 8. To accommodate all the positions in a field, the number of bytes in the bitmap will normally contain more bits than the number of positions. If the number of bits in the bitmap is denoted by q , then

$$q = (r+1)*8 \quad (\text{where } r \text{ is the integer quotient of } (m-1)/8)$$

For *Track 2 Data* m_{TRACK2} is a maximum of 13 digits, resulting in a bitmap of 16 bits or 2 bytes. For *Track 1 Data* the maximum value of m_{TRACK1} is 48 resulting in a bitmap of length 6 bytes or 48 bits.

4.8 Reserved for Future Use (RFU)

A bit specified as Reserved for Future Use (RFU) must be set as specified, or to 0b if no indication is given. An entity receiving a bit specified as RFU must ignore such a bit and must not change its behaviour, unless explicitly stated otherwise.

A data field having a value coded on multiple bits or bytes must not be set to a value specified as RFU. An entity receiving a data field having a value specified as RFU behaves as defined by a requirement that specifically addresses the situation.

5 C-APDU Commands

This chapter defines the commands and responses supported by the Kernel.

5.1 Introduction

The INS byte of the C-APDU is structured according to [EMV Book 1]. The coding of INS and its relationship to CLA are shown in Table 5.1.

Table 5.1—Coding of the Instruction Byte

CLA	INS	Meaning
'80'	'2A'	COMPUTE CRYPTOGRAPHIC CHECKSUM ¹²
'80'	'EA'	EXCHANGE RELAY RESISTANCE DATA
'80'	'AE'	GENERATE AC
'80'	'CA'	GET DATA ¹³
'80'	'A8'	GET PROCESSING OPTIONS
'80'	'DA'	PUT DATA ¹³
'00'	'B2'	READ RECORD

The status bytes returned by the Card are coded as specified in section 6.3.5 of [EMV Book 3]. In addition to the status bytes specific to each command, the Card may return the status bytes shown in Table 5.2.

Table 5.2—Generic Status Bytes

SW1	SW2	Meaning
'6D'	'00'	Instruction code not supported or invalid
'6E'	'00'	Class not supported
'6F'	'00'	No precise diagnosis

¹² Only implemented for the MAG Implementation Option

¹³ Only implemented for the DE/DS Implementation Option

5.2 Compute Cryptographic Checksum

5.2.1 Definition and Scope

The COMPUTE CRYPTOGRAPHIC CHECKSUM command initiates the computation of the dynamic CVC3 on the Card.

5.2.2 Command Message

The COMPUTE CRYPTOGRAPHIC CHECKSUM command message is coded according to Table 5.3.

Table 5.3—Compute Cryptographic Checksum Command Message

Code	Value
CLA	'80'
INS	'2A'
P1	'8E'
P2	'80'
Lc	var.
Data	<i>UDOL</i> related data
Le	'00'

The data field of the command message is coded according to the *UDOL* following the rules defined in section 4.1.4. If the Card does not have a *UDOL*, the Kernel uses the *Default UDOL*.

5.2.3 Data Field Returned in the Response Message

The data field of the response message is a constructed data object with tag '77' (Response Message Template) as shown in Table 5.4. The value field may include several TLV coded data objects, but always includes the *Application Transaction Counter*. The value field may also include the *CVC3 (Track1)*, *CVC3 (Track2)*, and *POS Cardholder Interaction Information*.

Data objects in *Response Message Template Format 2* may appear in any order.

Table 5.4—Compute Cryptographic Checksum Response Message Data Field

Tag	Value	Presence
'77'	<i>Response Message Template Format 2</i>	M
	'9F36' <i>Application Transaction Counter</i>	M
	'9F60' <i>CVC3 (Track1)</i>	C
	'9F61' <i>CVC3 (Track2)</i>	C
	'DF4B' <i>POS Cardholder Interaction Information</i>	C

5.2.4 Status Bytes

The status bytes that may be sent in response to the COMPUTE CRYPTOGRAPHIC CHECKSUM command are listed in Table 5.5.

Table 5.5—Status Bytes for Compute Cryptographic Checksum Command

SW1	SW2	Meaning
'67'	'00'	Wrong length
'69'	'85'	Conditions of use not satisfied
'6A'	'86'	Incorrect parameters P1-P2
'90'	'00'	Normal processing

5.3 Exchange Relay Resistance Data

5.3.1 Definition and Scope

The EXCHANGE RELAY RESISTANCE DATA command exchanges relay resistance related data with the Card.

5.3.2 Command Message

The EXCHANGE RELAY RESISTANCE DATA command message is coded according to Table 5.6.

Table 5.6—Exchange Relay Resistance Data Command Message

Code	Value
CLA	'80'
INS	'EA'
P1	'00'
P2	'00'
Lc	'04'
Data	<i>Terminal Relay Resistance Entropy</i>
Le	'00'

5.3.3 Data Field Returned in the Response Message

The data object returned in the response message is a primitive data object with tag '80' and length '0A'. The value field consists of the concatenation without delimiters (tag and length) of the value fields of the data objects specified in Table 5.7.

Table 5.7—Exchange Relay Resistance Data Response Message Data Field

Tag	Length	Byte	Value	Presence
'80'	'0A'	1-4	<i>Device Relay Resistance Entropy</i>	M
		5-6	<i>Min Time For Processing Relay Resistance APDU</i>	M
		7-8	<i>Max Time For Processing Relay Resistance APDU</i>	M
		9-10	<i>Device Estimated Transmission Time For Relay Resistance R-APDU</i>	M

5.3.4 Status Bytes

The status bytes that may be sent in response to the EXCHANGE RELAY RESISTANCE DATA command are listed in Table 5.8.

Table 5.8—Status Bytes for Exchange Relay Resistance Data Command

SW1	SW2	Meaning
'67'	'00'	Wrong length
'69'	'85'	Conditions of use not satisfied
'6A'	'86'	Incorrect parameters P1-P2
'90'	'00'	Normal processing

5.4 Generate AC

5.4.1 Definition and Scope

The GENERATE AC command sends transaction-related data to the Card, which then computes and returns an *Application Cryptogram*. Depending on the risk management in the Card, the cryptogram returned by the Card may differ from that requested in the command message. The Card may return an AAC (transaction declined), an ARQC (online authorization request), or a TC (transaction approved).

5.4.2 Command Message

The GENERATE AC command message is coded according to Table 5.9.

Table 5.9—Generate AC Command Message

Code	Value
CLA	'80'
INS	'AE'
P1	Reference Control Parameter (see Table 5.10)
P2	'00'
Lc	var.
Data	<i>CDOL1 Related Data</i> <i>DSDOL</i> related data (conditional (if IDS write performed))
Le	'00'

Table 5.10—Generate AC Reference Control Parameter

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0							AAC
0	1							TC
1	0							ARQC
1	1							RFU
		x						RFU
		0						Other values RFU
			0					CDA not requested
			1					CDA requested
				x	x	x	x	RFU
				0	0	0	0	Other values RFU

The data field of the command message contains *CDOL1 Related Data* coded according to *CDOL1* following the rules defined in section 4.1.4.

In the case of IDS data writing, the data field of the command message is a concatenation of *CDOL1 Related Data* followed by *DSDOL* related data coded according to *DSDOL* following the rules defined in section 4.1.4.

5.4.3 Data Field Returned in the Response Message

The data field in the response message to the GENERATE AC command is coded according to either format 1 or format 2, as follows.

Format 1

In the case of format 1, the data object returned in the response message is a primitive data object *Response Message Template Format 1* with tag equal to '80'. The value field consists of the concatenation without delimiters (tag and length) of the value fields of the data objects specified in Table 5.11.

Format 1 is not used if CDA is performed.

Table 5.11—Generate AC Response Message Data Field (Format 1)

Tag	Length	Value	Presence
'80'	Var.	<i>Cryptogram Information Data</i>	M
		<i>Application Transaction Counter</i>	M
		<i>Application Cryptogram</i>	M
		<i>Issuer Application Data</i>	O

Format 2

In the case of format 2, the data object returned in the response message varies depending on whether CDA was performed or not.

CDA Not Performed

If CDA is not performed, the data object returned in the response message is a constructed data object with tag equal to '77' (*Response Message Template Format 2*), as specified in Table 5.12.

Data objects in *Response Message Template Format 2* may appear in any order.

Table 5.12—Generate AC Response Message Data Field (Format 2) – No CDA

Tag	Value		Presence
'77'	<i>Response Message Template Format 2</i>		M
	'9F27'	<i>Cryptogram Information Data</i>	M
	'9F36'	<i>Application Transaction Counter</i>	M
	'9F26'	<i>Application Cryptogram</i>	M
	'9F10'	<i>Issuer Application Data</i>	O
	'DF4B'	<i>POS Cardholder Interaction Information</i>	O

CDA Performed

If CDA is performed, the data object returned in the response message is a constructed data object with tag equal to '77' (*Response Message Template Format 2*). It contains at least the three mandatory data objects specified in Table 5.13, and optionally the *Issuer Application Data*.

Data objects in *Response Message Template Format 2* may appear in any order.

Table 5.13—Generate AC Response Message Data Field (Format 2) – CDA

Tag	Value		Presence
'77'	<i>Response Message Template Format 2</i>		M
	'9F27'	<i>Cryptogram Information Data</i>	M
	'9F36'	<i>Application Transaction Counter</i>	M
	'9F4B'	<i>Signed Dynamic Application Data</i>	M
	'9F10'	<i>Issuer Application Data</i>	O
	'DF4B'	<i>POS Cardholder Interaction Information</i>	O

5.4.4 Status Bytes

The status bytes that may be sent in response to the GENERATE AC command are listed in Table 5.14.

Table 5.14—Status Bytes for Generate AC Command

SW1	SW2	Meaning
'67'	'00'	Wrong length
'69'	'85'	Conditions of use not satisfied
'6A'	'86'	Incorrect parameters P1-P2
'90'	'00'	Normal processing

5.5 Get Data

5.5.1 Definition and Scope

The GET DATA command is used to retrieve a primitive data object from the Card not encapsulated in a record.

5.5.2 Command Message

The GET DATA command message is coded according to Table 5.15.

Table 5.15—Get Data Command Message

Code	Value
CLA	'80'
INS	'CA'
P1 P2	Tag
Lc	Not present
Data	Not present
Le	'00'

Table 5.16 lists the tag values supported for the GET DATA command.

Table 5.16—Supported P1 || P2 Values for Get Data Command

P1 P2	Data Object
'9F70'	<i>Protected Data Envelope 1</i>
'9F71'	<i>Protected Data Envelope 2</i>
'9F72'	<i>Protected Data Envelope 3</i>
'9F73'	<i>Protected Data Envelope 4</i>
'9F74'	<i>Protected Data Envelope 5</i>
'9F75'	<i>Unprotected Data Envelope 1</i>
'9F76'	<i>Unprotected Data Envelope 2</i>
'9F77'	<i>Unprotected Data Envelope 3</i>
'9F78'	<i>Unprotected Data Envelope 4</i>
'9F79'	<i>Unprotected Data Envelope 5</i>

5.5.3 Data Field Returned in the Response Message

The data field of the response message contains the primitive data object referred to in P1 || P2 of the command message (in other words, including its tag and its length).

5.5.4 Status Bytes

The status bytes that may be sent in response to the GET DATA command are listed in Table 5.17.

Table 5.17—Status Bytes for Get Data Command

SW1	SW2	Meaning
'69'	'85'	Conditions of use not satisfied
'6A'	'81'	Wrong parameter(s) P1 P2; function not supported
'6A'	'88'	Referenced data (data object) not found
'90'	'00'	Normal processing

5.6 Get Processing Options

5.6.1 Definition and Scope

The GET PROCESSING OPTIONS command initiates the transaction within the Card.

5.6.2 Command Message

The GET PROCESSING OPTIONS command message is coded according to Table 5.18.

Table 5.18—Get Processing Options Command Message

Code	Value
CLA	'80'
INS	'A8'
P1	'00'
P2	'00'
Lc	var.
Data	<i>PDOL Related Data</i>
Le	'00'

The data field of the command message consists of *PDOL Related Data*. *PDOL Related Data* is the Command Template with tag '83' and with a value field coded according to the *PDOL* provided by the Card in the response to the SELECT command. If the *PDOL* is not provided by the Card, the length field of the Command Template is set to zero. Otherwise the length field is the total length of the value fields of the data objects transmitted to the Card. The value fields are concatenated according to the rules defined in section 4.1.4.

5.6.3 Data Field Returned in the Response Message

The data field in the response message to the GET PROCESSING OPTIONS command is coded according to either format 1 or format 2, as follows.

Format 1

In the case of format 1, the data object returned in the response message is a primitive data object with tag equal to '80'. The value field consists of the concatenation without delimiters (tag and length) of the value fields of the *Application Interchange Profile* and the *Application File Locator*, as shown in Table 5.19.

Table 5.19—Get Processing Options Response Message Data Field (Format 1)

Tag	Length	Value	Presence
'80'	Var.	<i>Application Interchange Profile</i>	M
		<i>Application File Locator</i>	M

Format 2

In the case of format 2, the data object returned in the response message is a constructed data object with tag '77' (*Response Message Template Format 2*). The value field may include several TLV coded objects, but always includes the *Application Interchange Profile* and *Application File Locator*, as shown in Table 5.20. If IDS is supported by both Card and Kernel, then also the IDS related data objects shown in Table 5.20 may be included in the *Response Message Template Format 2*. Data objects in *Response Message Template Format 2* may appear in any order.

Table 5.20—Get Processing Options Response Message Data Field (Format 2)

Tag	Value		Presence
'77'	<i>Response Message Template Format 2</i>		M
	'82'	<i>Application Interchange Profile</i>	M
	'94'	<i>Application File Locator</i>	M
	'9F6F'	<i>DS Slot Management Control</i>	O
	'9F5F'	<i>DS Slot Availability</i>	O
	'9F7F'	<i>DS Unpredictable Number</i>	O
	'9F7D'	<i>DS Summary 1</i>	O
	'9F54'	<i>DS ODS Card</i>	O

5.6.4 Status Bytes

The status bytes that may be sent in response to the GET PROCESSING OPTIONS command are listed in Table 5.21.

Table 5.21—Status Bytes for Get Processing Options Command

SW1	SW2	Meaning
'67'	'00'	Wrong length
'69'	'85'	Conditions of use not satisfied
'6A'	'86'	Incorrect parameters P1-P2
'90'	'00'	Normal processing

5.7 Put Data

5.7.1 Definition and Scope

The PUT DATA command is used to store a primitive data object not encapsulated in a record in the Card.

5.7.2 Command Message

The PUT DATA command message is coded according to Table 5.22.

Table 5.22—Put Data Command Message

Code	Value
CLA	'80'
INS	'DA'
P1 P2	Tag
Lc	var.
Data	New data value
Le	Not present

Single byte tags are preceded with a leading '00' byte to fill P1 || P2. Table 5.23 lists the minimum set of tag values that must be supported for the PUT DATA command.

Table 5.23—Supported P1 || P2 values for Put Data Command

P1 P2	Data Object
'9F75'	<i>Unprotected Data Envelope 1</i>
'9F76'	<i>Unprotected Data Envelope 2</i>
'9F77'	<i>Unprotected Data Envelope 3</i>
'9F78'	<i>Unprotected Data Envelope 4</i>
'9F79'	<i>Unprotected Data Envelope 5</i>

5.7.3 Data Field Returned in the Response Message

There is no data field in the response message of the PUT DATA command.

5.7.4 Status Bytes

The status bytes that may be sent in response to the PUT DATA command are listed in Table 5.24.

Table 5.24—Status Bytes for Put Data Command

SW1	SW2	Meaning
'67'	'00'	Wrong length
'6A'	'88'	Referenced data (data object) not found
'90'	'00'	Normal processing

5.8 Read Record

5.8.1 Definition and Scope

The READ RECORD command reads a file record in a linear file. The response of the Card consists of returning the record.

5.8.2 Command Message

The READ RECORD command message is coded according to Table 5.25.

Table 5.25—Read Record Command Message

Code	Value
CLA	'00'
INS	'B2'
P1	Record number
P2	See Table 5.26
Lc	Not present
Data	Not present
Le	'00'

Table 5.26 specifies the coding of P2 of the READ RECORD command.

Table 5.26—P2 of Read Record Command

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
x	x	x	x	x				SFI
					1	0	0	P1 is a record number

5.8.3 Data Field Returned in the Response Message

The data field in the Card response contains the record requested by the command. For SFIs in the range 1-10, the record is a TLV constructed data object with tag '70' as shown in Table 5.27.

Table 5.27—Read Record Response Message Data Field

'70'	Length	Record Template
------	--------	-----------------

5.8.4 Status Bytes

The status bytes that may be sent in response to the READ RECORD command are listed in Table 5.28.

Table 5.28—Status Bytes for Read Record Command

SW1	SW2	Meaning
'69'	'85'	Conditions of use not satisfied
'6A'	'82'	Wrong parameters P1 P2; file not found
'6A'	'83'	Wrong parameters P1 P2; record not found
'6A'	'86'	Incorrect parameters P1 P2
'90'	'00'	Normal processing

6 Kernel State Machine

This chapter describes the transaction processing of the Kernel after it has been initiated by Process M.

6.1 Implementation Principles

The transaction processing is specified as a state machine that is triggered by external Signals that cause state transitions. The state machine is presented in more detail in Annex B.

These principles are used in order to present the application concepts. The same principles do not have to be followed in the actual implementation. However, the implementation must behave in a way that is indistinguishable from the behaviour specified in this chapter.

The Kernel receives incoming Signals on a Queue. The Queue is used by Process M for the ACT Signal and by Process P for the RA and L1RSP Signals.

The Kernel can be in a state for which a state transition is triggered by Signals that are not at the top of a Queue. In this case the state transition is triggered by the first Signal on the Queue that causes a state transition. Signals on the Queue that are not handled in the current state must not be lost and must stay on the Queue for processing in subsequent states.

6.2 Kernel Started

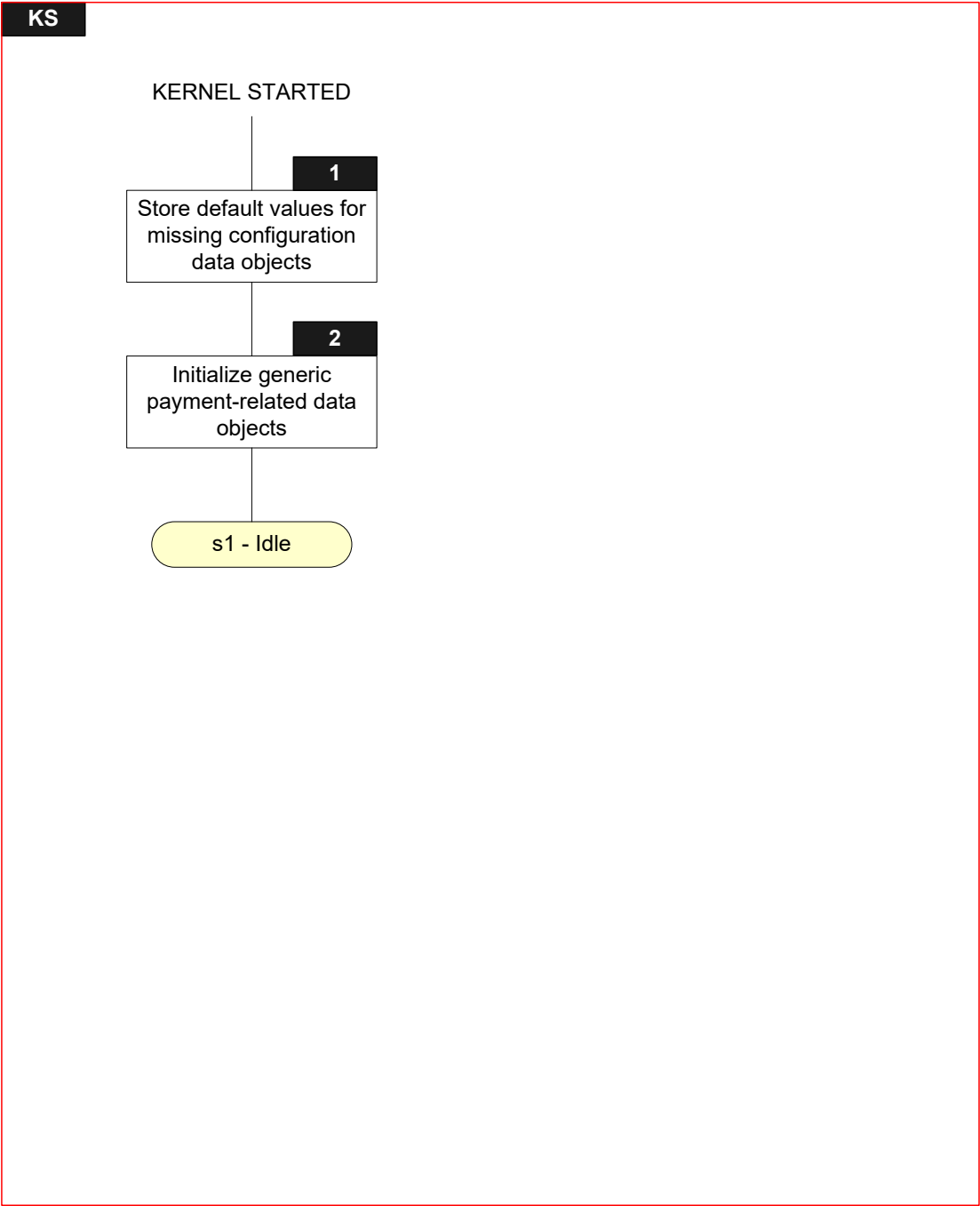
6.2.1 Local Variables

Name	Length	Format	Description
T	var.	b	Tag of TLV encoded string
L	var.	b	Length of TLV encoded string
V	var. up to 253	b	Value of TLV encoded string

6.2.2 Flow Diagram

Figure 6.1 shows the flow diagram of startup of the Kernel. Symbols in this diagram are labelled KS.X.

Figure 6.1—Kernel Started Flow Diagram



6.2.3 Processing

KS.1

Instantiate the TLV Database to be able to store the tags identified in Table A.1.

Extend the TLV Database with the Proprietary Tags, if any.

Copy the value of the configuration data objects from the Configuration Data into the TLV Database.

Use the default value as defined in Table 4.2 if a mandatory configuration data object is missing from the Configuration Data:

FOR every T for which a default value is defined in Table 4.2

```
{  
    IF      [IsNotPresent(T)]  
    THEN  
        Store LV as per Table 4.2 in the TLV Database for tag T  
    ENDIF  
}
```

KS.2

Mobile Support Indicator := '01'

Initialize *Outcome Parameter Set* as follows:

Outcome Parameter Set := '0000 ... 00'

'Status' in *Outcome Parameter Set* := N/A

'Start' in *Outcome Parameter Set* := N/A

'CVM' in *Outcome Parameter Set* := N/A

CLEAR 'UI Request on Outcome Present' in *Outcome Parameter Set*

CLEAR 'UI Request on Restart Present' in *Outcome Parameter Set*

CLEAR 'Data Record Present' in *Outcome Parameter Set*

SET 'Discretionary Data Present' in *Outcome Parameter Set*

'Receipt' in *Outcome Parameter Set* := N/A

'Alternate Interface Preference' in *Outcome Parameter Set* := N/A

'Field Off Request' in *Outcome Parameter Set* := N/A

'Removal Timeout' in *Outcome Parameter Set* := 0

'Online Response Data' in *Outcome Parameter Set* := N/A

Initialize *User Interface Request Data* as follows:

User Interface Request Data := '0000 ... 00'

'Message Identifier' in *User Interface Request Data* := N/A

'Status' in *User Interface Request Data* := N/A

'Hold Time' in *User Interface Request Data* := *Message Hold Time*

'Language Preference' in *User Interface Request Data* := '000000000000000000'

Initialize *Error Indication* as follows:

Error Indication := '0000 ... 00'

'L1' in *Error Indication* := OK

'L2' in *Error Indication* := OK

'L3' in *Error Indication* := OK

'SW12' in *Error Indication* := '0000'

'Msg On Error' in *Error Indication* := N/A

6.3 State 1 – Idle

6.3.1 Local Variables

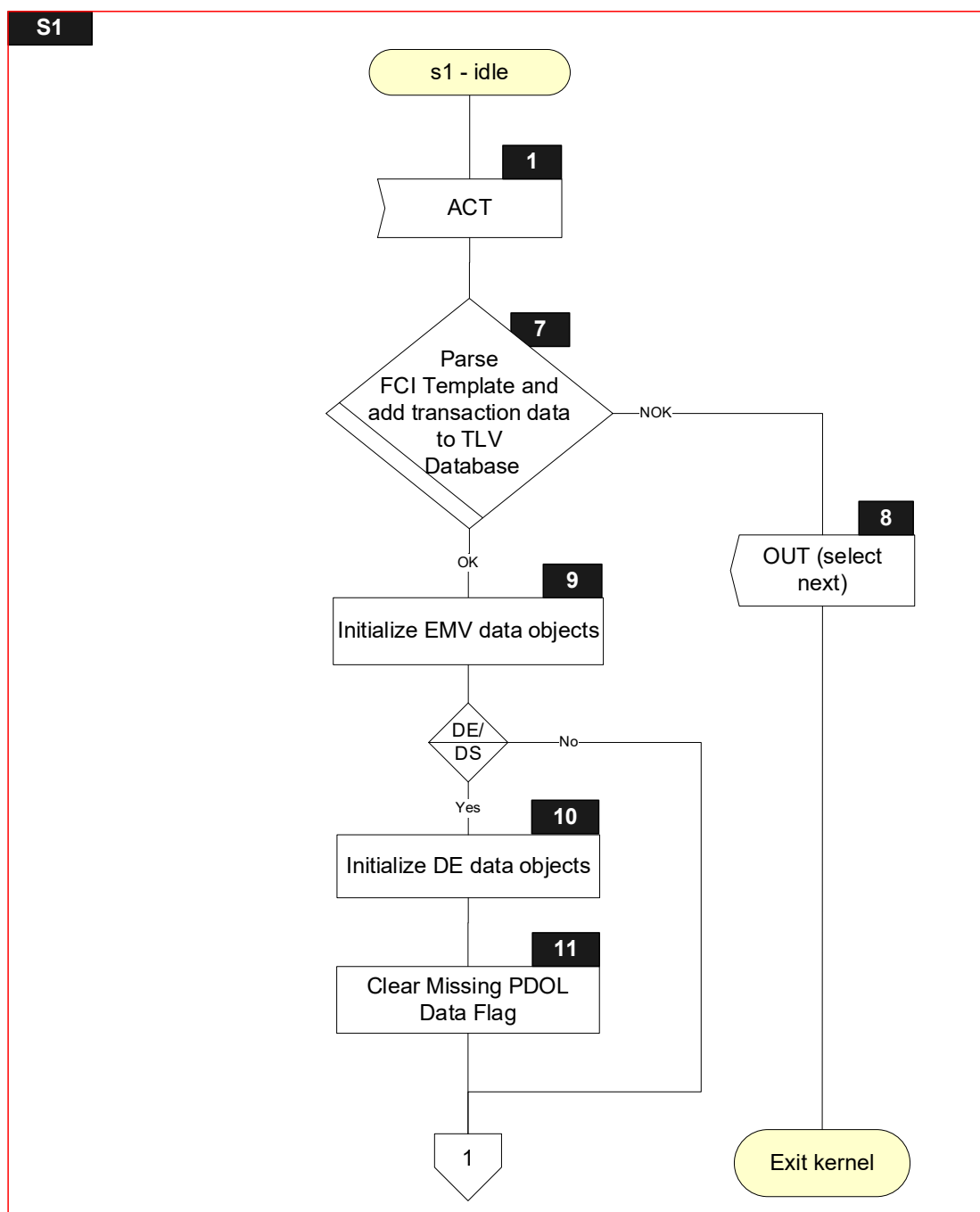
Name	Length	Format	Description
Sync Data	var.	b	List of data objects returned with ACT Signal
T	var.	b	Tag of TLV encoded string
L	var.	b	Length of TLV encoded string
Missing PDOL Data Flag ⁽¹⁾	1	b	Boolean used to indicate if data referenced in <i>PDOL</i> is not present in the TLV Database.

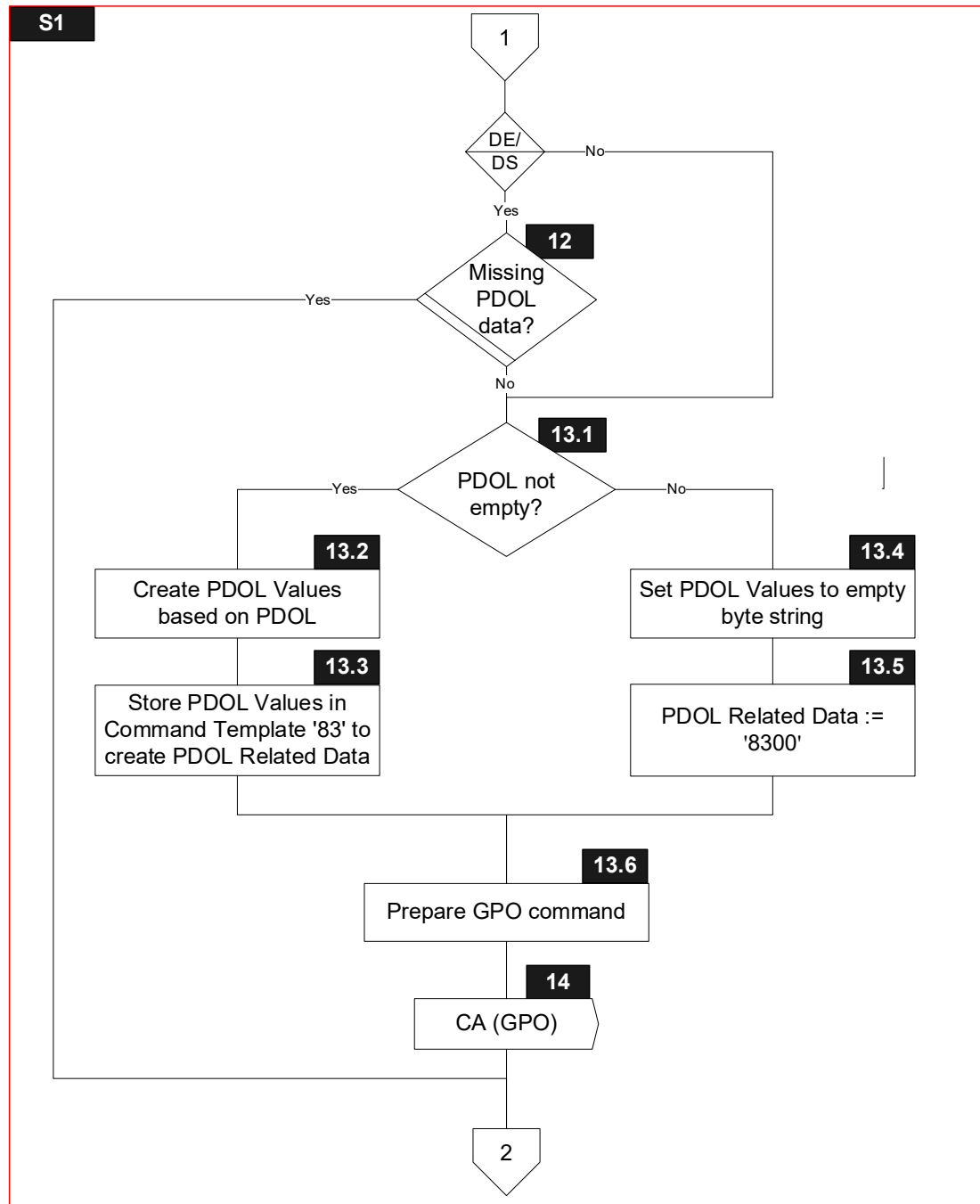
⁽¹⁾ Only for the DE/DS Implementation Option

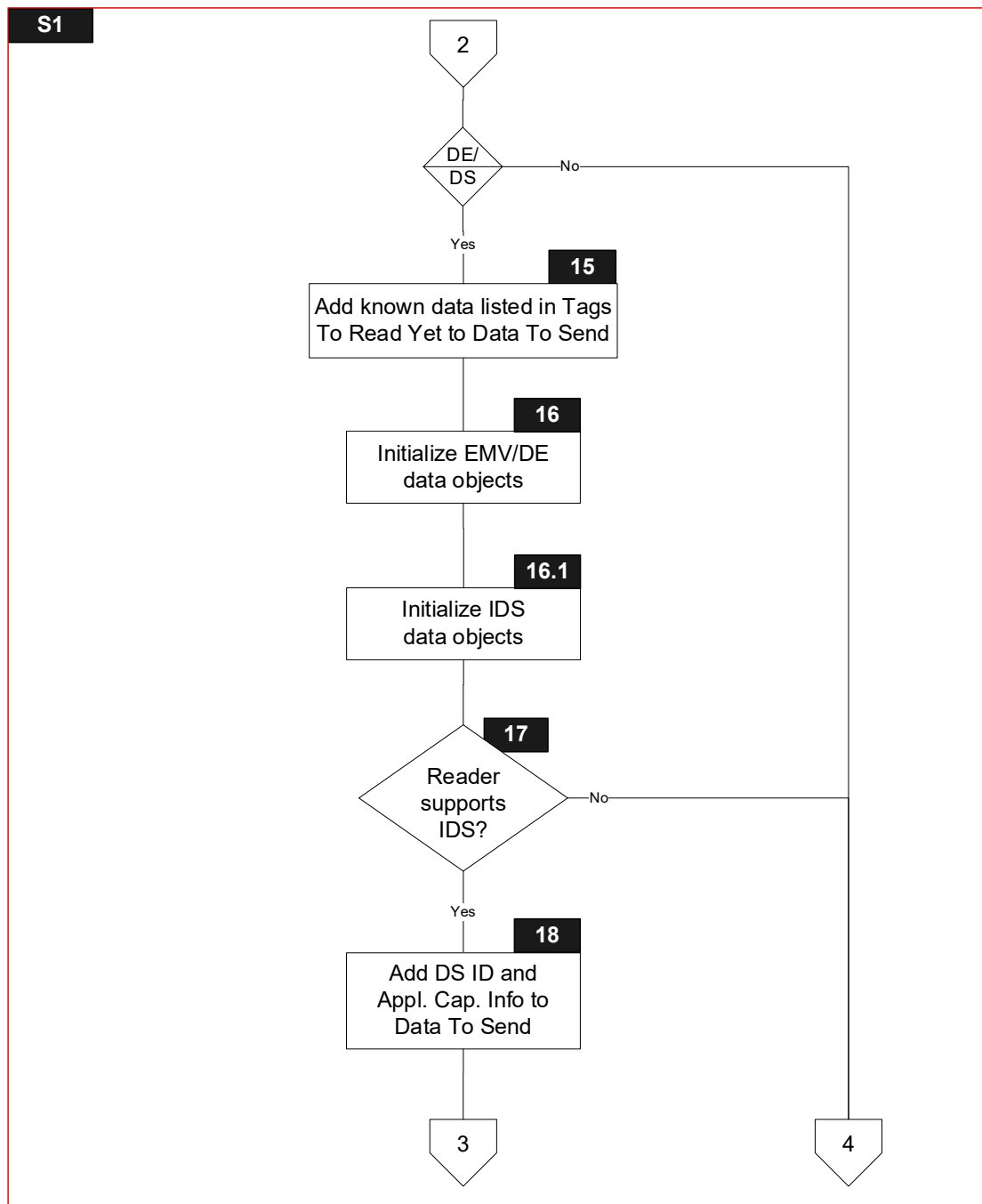
6.3.2 Flow Diagram

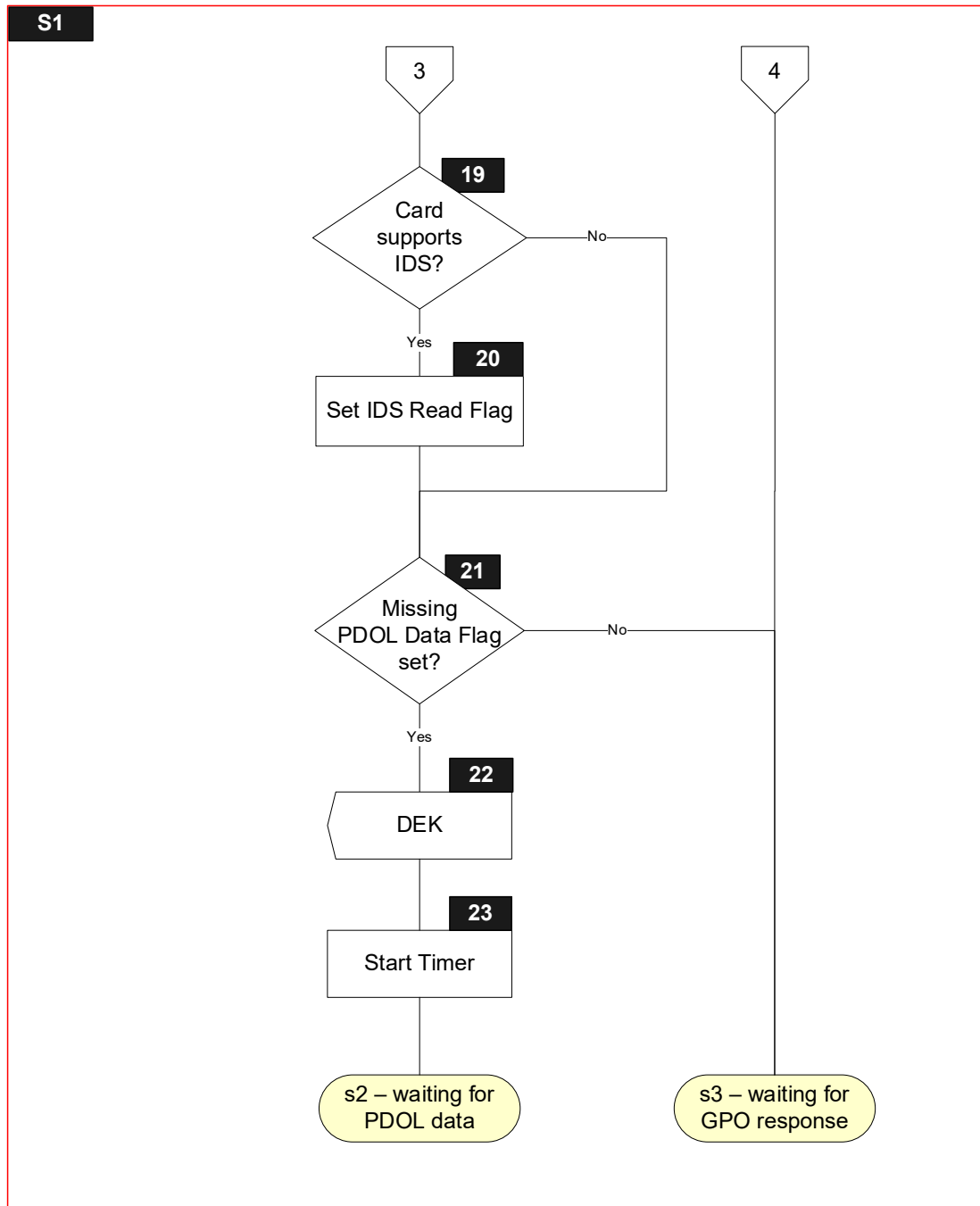
Figure 6.2 shows the flow diagram of *s1 – idle*. Symbols in this diagram are labelled S1.X.

Figure 6.2—State 1 Flow Diagram









6.3.3 Processing

Note that symbols S1.10, S1.11, S1.12, S1.15, S1.16, S1.16.1, S1.17, S1.18, S1.19, S1.20, S1.21, S1.22 and S1.23 are only implemented for the DS/DE Implementation Option.

S1.1

Receive ACT Signal with Sync Data

S1.7

Add the transaction data provided in the ACT Signal to the TLV Database

Parse and store the *File Control Information Template* if included in Sync Data

FOR every TLV in Sync Data

```
{
    IF      [T = TagOf(File Control Information Template)]
    THEN
        IF      [NOT ParseAndStoreCardResponse(TLV)]
        THEN
            'L2' in Error Indication := PARSING ERROR
            GOTO S1.8
        ENDIF
    ELSE
        IF      [(IsKnown(T) OR IsPresent(T)) AND
            update conditions of T include ACT Signal]
        THEN
            Store LV in the TLV Database for tag T
        ENDIF
    ENDIF
}
```

If the *Language Preference* is returned from the Card, then copy it to 'Language Preference' in *User Interface Request Data*:

```
IF      [IsNotEmpty(TagOf(Language Preference))]
THEN
    'Language Preference' in User Interface Request Data := Language Preference
    If the length of Language Preference is less than 8 bytes, then pad 'Language Preference' in User Interface Request Data with trailing hexadecimal zeroes to 8 bytes.
ENDIF
```

```
IF [IsNotPresent(TagOf(DF Name)) OR IsEmpty(TagOf(DF Name))]  
THEN  
    'L2' in Error Indication := CARD DATA MISSING  
    GOTO S1.8  
ENDIF  
IF [IsNotEmpty(TagOf(Application Capabilities Information))]  
THEN  
    IF    ['Support for field off detection' in Application Capabilities  
          Information is set]  
    THEN  
        'Field Off Request' in Outcome Parameter Set := Hold Time Value  
    ENDIF  
ENDIF  
GOTO S1.9
```

S1.8

```
'Status' in Outcome Parameter Set := SELECT NEXT  
'Start' in Outcome Parameter Set := C  
Initialize(Discretionary Data)  
AddToList(GetTLV(TagOf(Error Indication)), Discretionary Data)  
Send OUT(GetTLV(TagOf(Outcome Parameter Set)),  
          GetTLV(TagOf(Discretionary Data))) Signal
```

S1.9

```
CVM Results := '000000'  
'AC type' in AC Type := TC  
Terminal Verification Results := '0000000000'  
ODA Status := '00'  
RRP Counter := '00'  
Terminal Capabilities[1] := Card Data Input Capability  
Terminal Capabilities[2] := '00'  
Terminal Capabilities[3] := Security Capability  
Initialize(Static Data To Be Authenticated)  
Generate Unpredictable Number as specified in section 8.1 and store in the TLV  
Database for TagOf(Unpredictable Number)
```

S1.10

Initialize(*Data Needed*)

Initialize(*Data To Send*)

Initialize(*Tags To Read Yet*)

IF [IsEmpty(TagOf(*Tags To Read*))]

THEN

AddListToList(*Tags To Read*, *Tags To Read Yet*)

ENDIF

IF [IsEmpty(TagOf(*Tags To Read*))]

THEN

AddToList(TagOf(*Tags To Read*), *Data Needed*)

ENDIF

S1.11

CLEAR Missing PDOL Data Flag

S1.12

FOR every TL entry in the *PDOL*

{

IF [($L > 0$) AND IsEmpty(T) AND Update Conditions of T include
'DET']

THEN

SET Missing PDOL Data Flag

AddToList(T, *Data Needed*)

ENDIF

}

IF [Missing PDOL Data Flag]

THEN

GOTO S1.15

ELSE

GOTO S1.13.1

ENDIF

S1.13.1

IF [IsEmpty(TagOf(*PDOL*))]

THEN

GOTO S1.13.2

ELSE

GOTO S1.13.4

ENDIF

S1.13.2

Use *PDOL* to create *PDOL Values* as a concatenated list of data objects without tags or lengths following the rules specified in section 4.1.4.

S1.13.3

Store *PDOL Values* in Command Template '83' to create *PDOL Related Data*.

S1.13.4

Store empty byte string in *PDOL Values*

S1.13.5

PDOL Related Data := '8300'

S1.13.6

Prepare GET PROCESSING OPTIONS command as specified in section 5.6.

S1.14

Send CA(GET PROCESSING OPTIONS) Signal

S1.15

FOR every T in *Tags To Read Yet*

```
{  
    IF      [IsEmpty(T)]  
    THEN  
        AddToList(GetTLV(T), Data To Send)  
        RemoveFromList(T, Tags To Read Yet)  
    ENDIF  
}
```

S1.16

Post-Gen AC Put Data Status := '00'

Pre-Gen AC Put Data Status := '00'

Initialize(Tags To Write Yet After Gen AC)

Initialize(Tags To Write Yet Before Gen AC)

IF [IsEmpty(TagOf(Tags To Write Before Gen AC))]

THEN

AddListToList(Tags To Write Before Gen AC, Tags To Write Yet Before Gen AC)

ENDIF

IF [IsEmpty(TagOf(Tags To Write After Gen AC))]

THEN

AddListToList(Tags To Write After Gen AC, Tags To Write Yet After Gen AC)

ENDIF

IF [IsEmpty(TagOf(Tags To Write Before Gen AC))]

THEN

AddToList(TagOf(Tags To Write Before Gen AC), Data Needed)

ENDIF

IF [IsEmpty(TagOf(Tags To Write After Gen AC))]

THEN

AddToList(TagOf(Tags To Write After Gen AC), Data Needed)

ENDIF

S1.16.1

IDS Status := '00'

DS Summary Status := '00'

DS Digest H := '0000000000000000'

S1.17

IF [IsEmpty(TagOf(DSVN Term))

AND IsPresent(TagOf(DS Requested Operator ID))]

THEN

GOTO S1.18

ELSE

GOTO S1.21

ENDIF

S1.18

```
IF      [IsPresent(TagOf(DS ID))]
THEN
    AddToList(GetTLV(TagOf(DS ID)), Data To Send)
ELSE
    Add empty DS ID to Data To Send:
    AddToList(TagOf(DS ID) || '00', Data To Send)
ENDIF

IF      [IsPresent(TagOf(Application Capabilities Information))]
THEN
    AddToList(GetTLV(TagOf(Application Capabilities Information)), Data To Send)
ELSE
    Add empty Application Capabilities Information to Data To Send:
    AddToList(TagOf(Application Capabilities Information) || '00', Data To Send)
ENDIF
```

S1.19

```
IF      [IsEmpty(TagOf (Application Capabilities Information)) AND
        (('Data Storage Version Number' in Application Capabilities Information =
         VERSION 1)
        OR
        ('Data Storage Version Number' in Application Capabilities Information =
         VERSION 2))
        AND IsNotEmpty(TagOf (DS ID)) ]
THEN
    GOTO S1.20
ELSE
    GOTO S1.21
ENDIF
```

S1.20

SET 'Read' in *IDS Status*

S1.21

```
IF      [Missing PDOL Data Flag is set]
THEN
    GOTO S1.22
ELSE
    GOTO s3 - waiting for GPO response
ENDIF
```


S1.22

Send DEK(*Data To Send, Data Needed*) Signal

Initialize(*Data To Send*)

Initialize(*Data Needed*)

S1.23

Start Timer (*Time Out Value*)

6.4 State 2 – Waiting for PDOL Data

State 2 is a state specific to Data Exchange processing and is only implemented for the DE/DS Implementation Option.

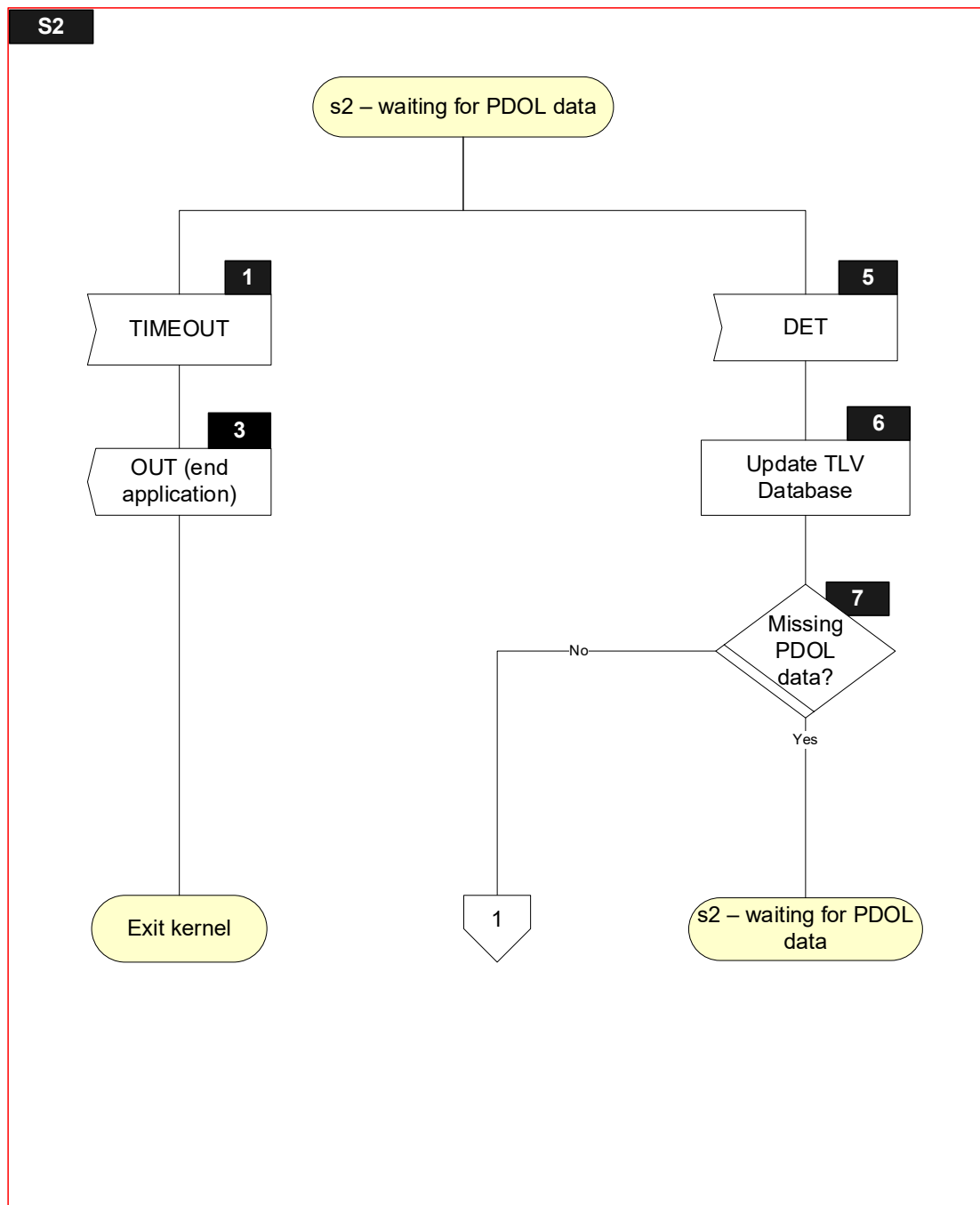
6.4.1 Local Variables

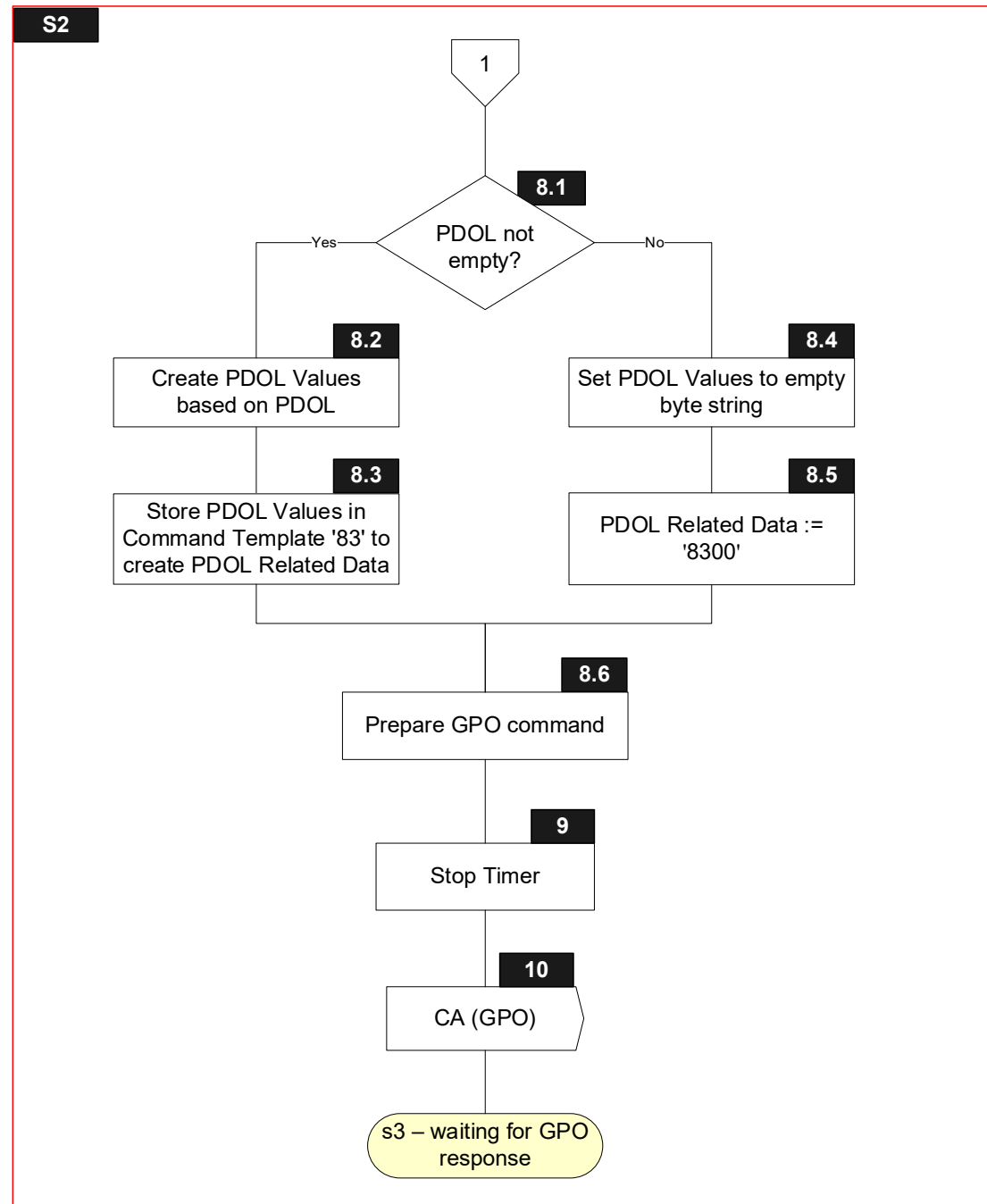
Name	Length	Format	Description
Sync Data	var.	b	List of data objects returned with DET Signal
T	var.	b	Tag of TLV encoded string
L	var.	b	Length of TLV encoded string
Missing PDOL Data Flag	1	b	Boolean used to indicate if data referenced in <i>PDOL</i> is not present in the TLV Database.

6.4.2 Flow Diagram

Figure 6.3 shows the flow diagram of `s2 – waiting for PDOL data`. Symbols in this diagram are labelled `S2.X`.

Figure 6.3—State 2 Flow Diagram





6.4.3 Processing

S2.1

Receive TIMEOUT Signal

S2.3

'Status' in *Outcome Parameter Set* := END APPLICATION

'L3' in *Error Indication* := TIME OUT

Initialize(*Discretionary Data*)

AddToList(GetTLV(TagOf(*Error Indication*)), *Discretionary Data*)

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*))) Signal

S2.5

Receive DET Signal with Sync Data

S2.6

UpdateWithDetData(Sync Data)

S2.7

CLEAR Missing PDOL Data Flag

FOR every TL entry in *PDOL*

```
{  
    IF      [IsEmpty(T) AND Update Conditions of T include 'DET']  
    THEN  
        SET Missing PDOL Data Flag  
    ENDIF  
}
```

```
IF      [Missing PDOL Data Flag]  
THEN  
    GOTO s2 - waiting for PDOL data  
ELSE  
    GOTO S2.8.1  
ENDIF
```

S2.8.1

IF [IsEmpty(TagOf(*PDOL*))]

THEN

GOTO S2.8.2

ELSE

GOTO S2.8.4

ENDIF

S2.8.2

Use *PDOL* to create *PDOL Values* as a concatenated list of data objects without tags or lengths following the rules specified in section 4.1.4.

S2.8.3

Store *PDOL Values* in Command Template '83' to create *PDOL Related Data*.

S2.8.4

Store empty byte string in *PDOL Values*

S2.8.5

PDOL Related Data := '8300'

S2.8.6

Prepare GET PROCESSING OPTIONS command as specified in section 5.6.

S2.9

Stop Timer

S2.10

Send CA(GET PROCESSING OPTIONS) Signal

6.5 State 3 – Waiting for GPO Response

6.5.1 Local Variables

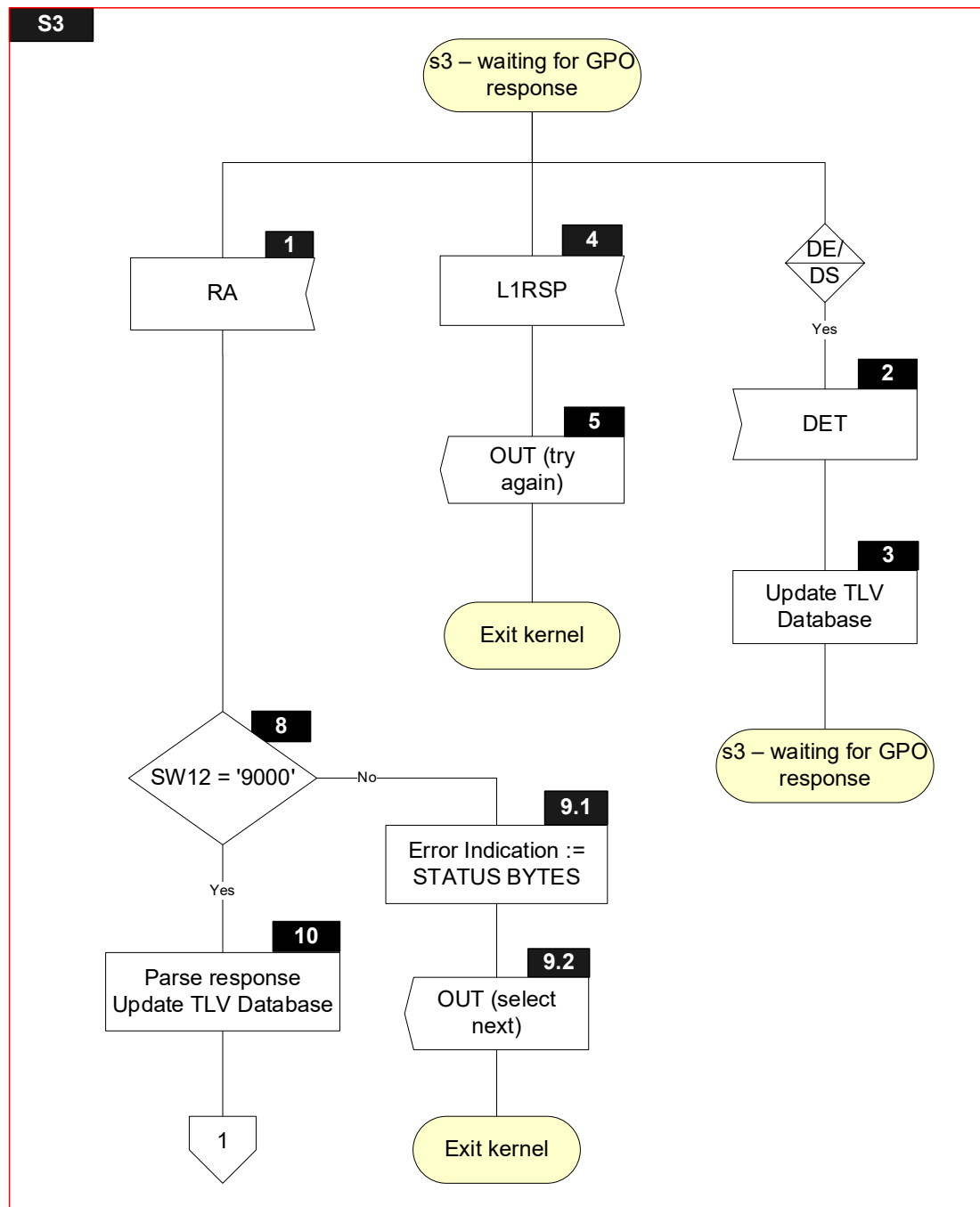
Name	Length	Format	Description
Return Code	1	b	Value returned with L1RSP Signal (TIME OUT ERROR, PROTOCOL ERROR, TRANSMISSION ERROR)
Sync Data ⁽¹⁾	var.	b	List of data objects returned with DET Signal
Parsing Result	1	b	Boolean used to store result of parsing a TLV string
SW12	2	b	Status bytes
Response Message Data Field	var. up to 256	b	TLV encoded string included in R-APDU of GET PROCESSING OPTIONS
T	var.	b	Tag of TLV encoded string
L	var.	b	Length of TLV encoded string
V	var. up to 253	b	Value of TLV encoded string

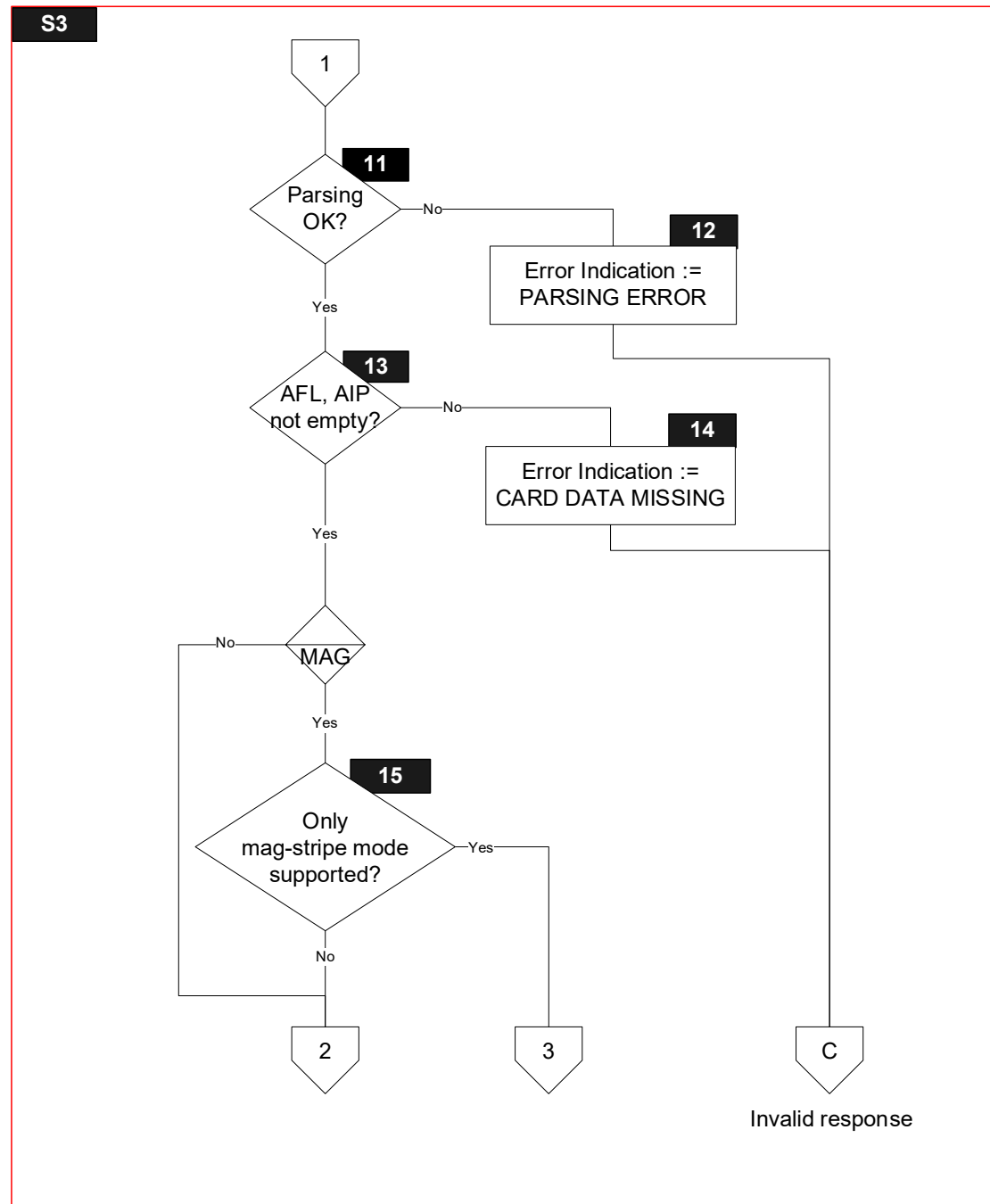
⁽¹⁾ Only for the DE/DS Implementation Option

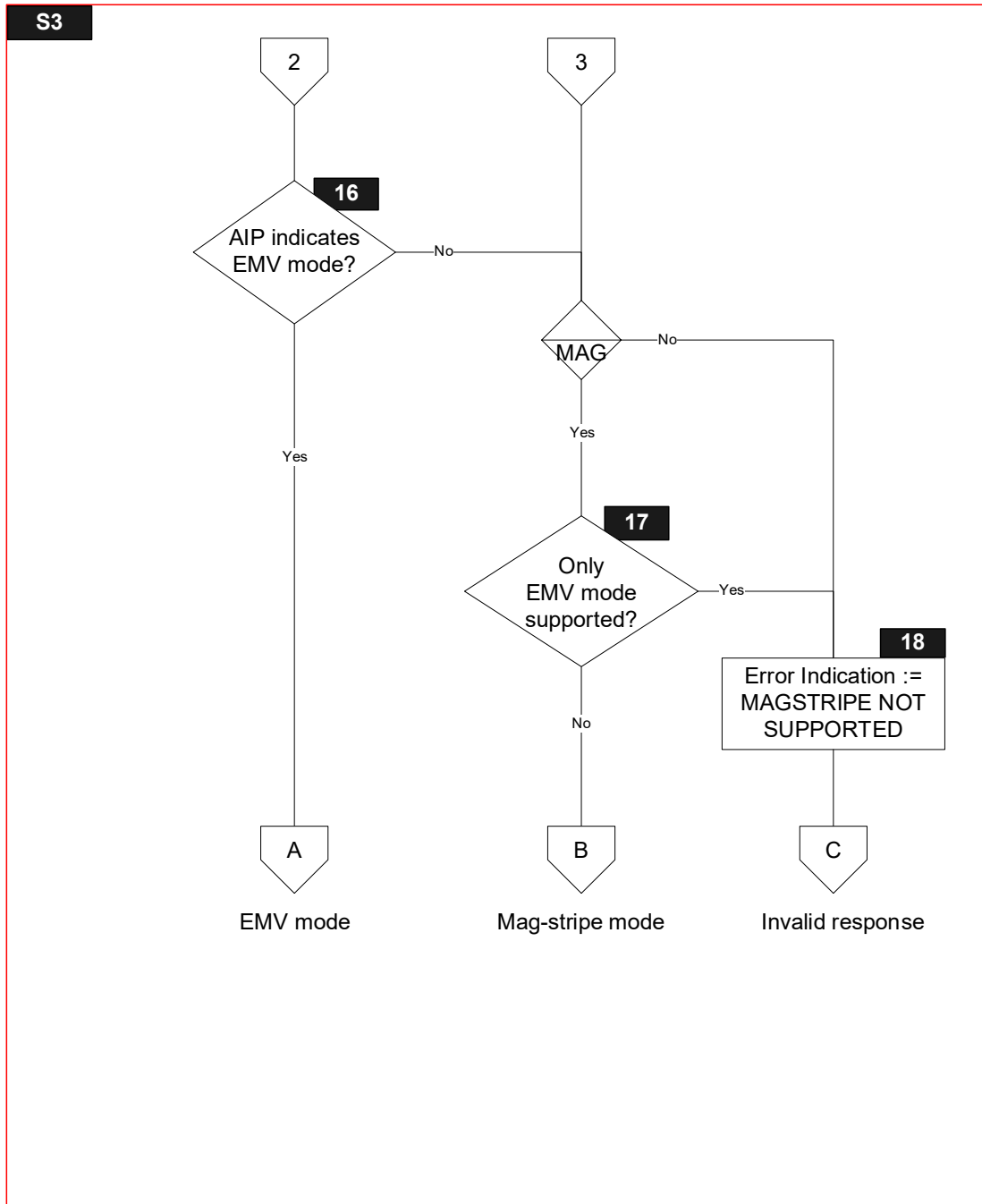
6.5.2 Flow Diagram

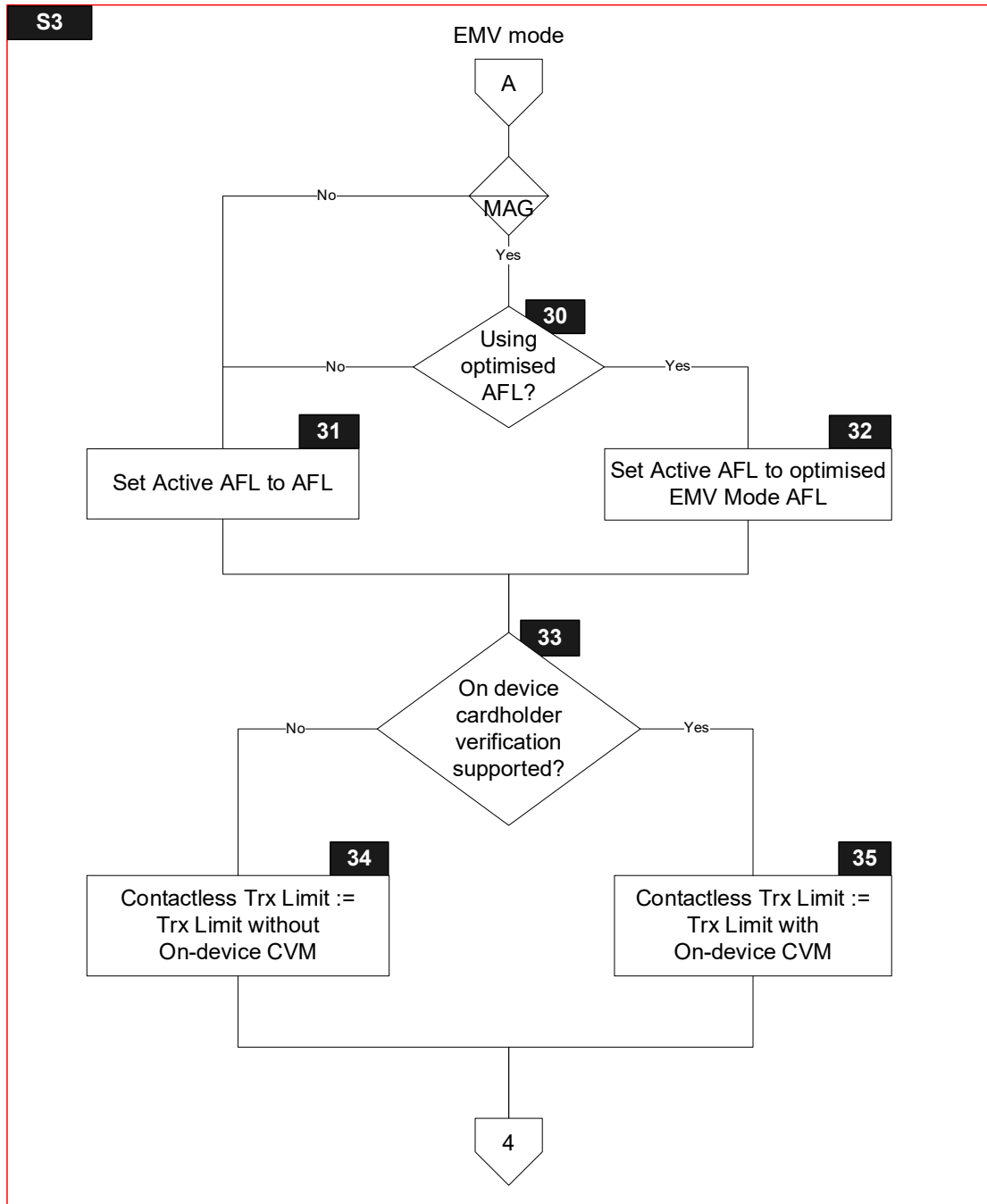
Figure 6.4 shows the flow diagram of `s3 - waiting for GPO response`. Symbols in this diagram are labelled S3.X.

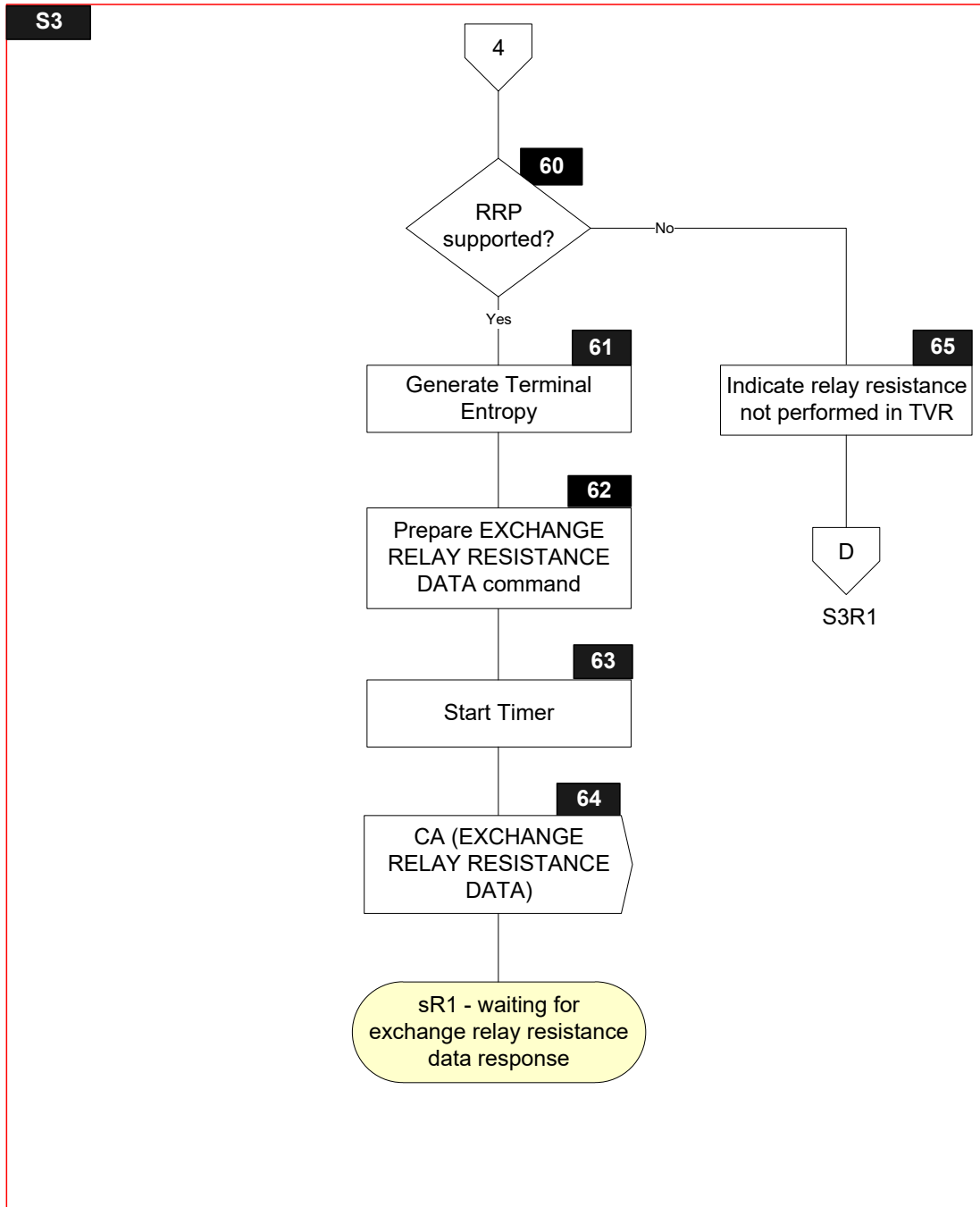
Figure 6.4—State 3 Flow Diagram

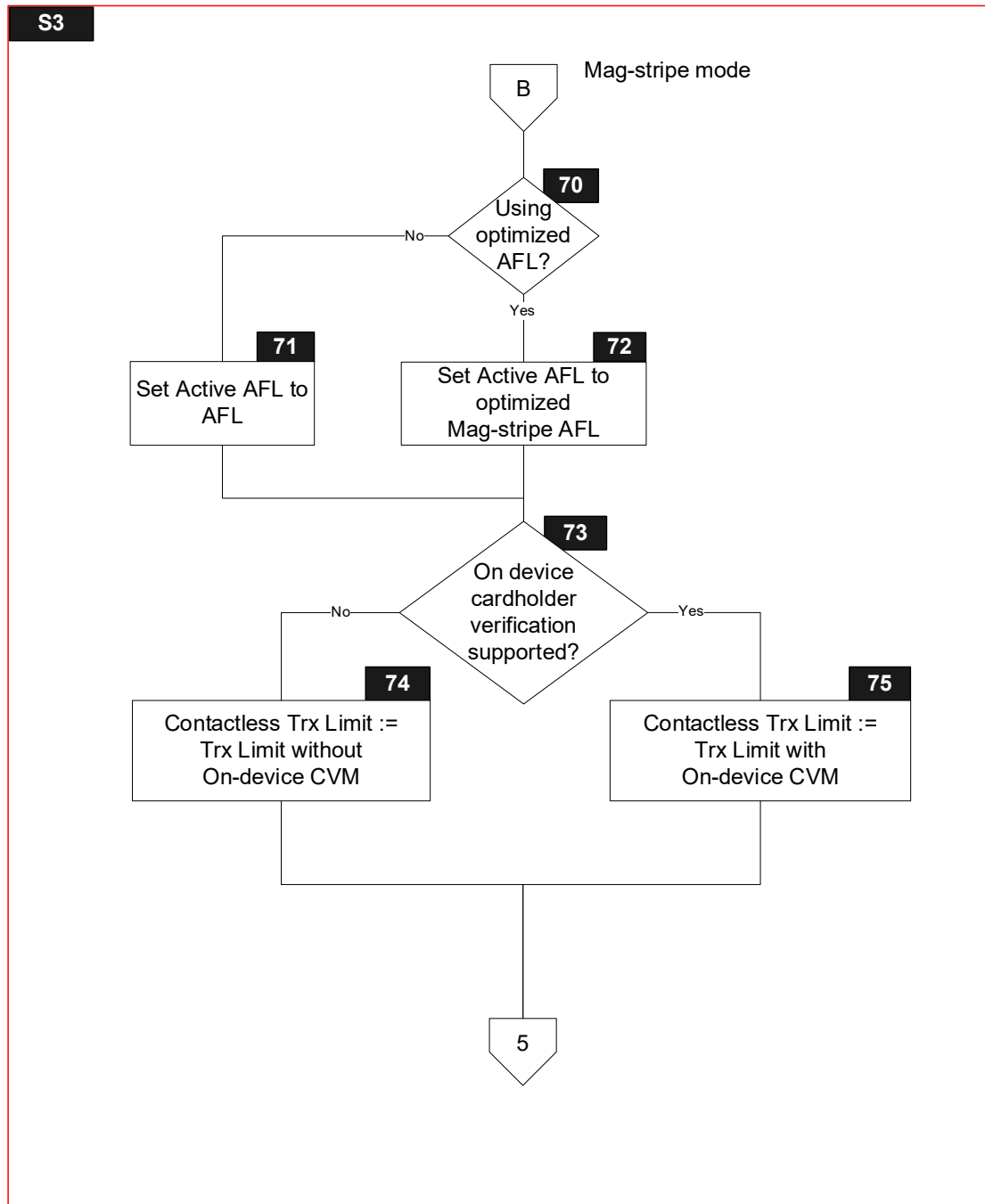


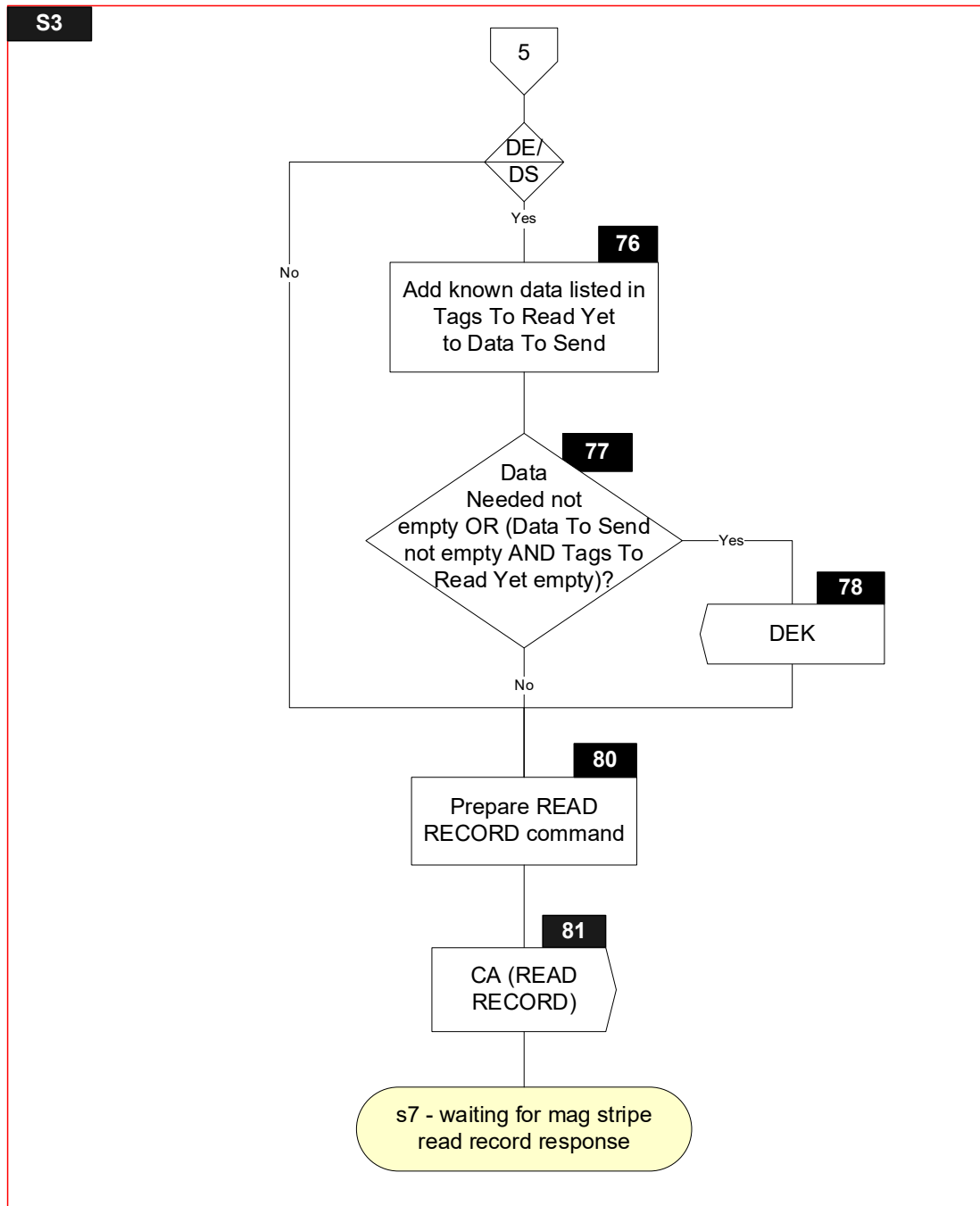




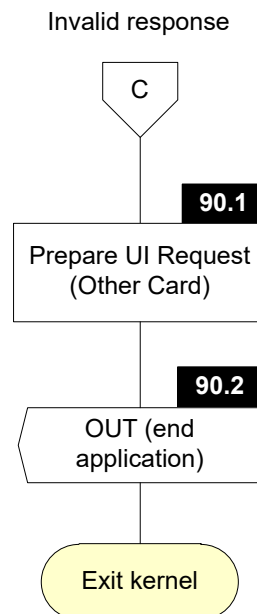








S3



6.5.3 Processing

Note that symbols S3.15, S3.17, S3.30, S3.32, S3.70, S3.71, S3.72, S3.73, S3.74, S3.75, S3.76, S3.77, S3.78, S3.80 and S3.81 are only implemented for the MAG Implementation Option.

Note that symbols S3.2, S3.3, S3.76, S3.77 and S3.78 are only implemented for the DE/DS Implementation Option.

S3.1

Receive RA Signal with Response Message Data Field and SW12

S3.2

Receive DET Signal with Sync Data

S3.3

UpdateWithDetData(Sync Data)

S3.4

Receive L1RSP Signal with Return Code

S3.5

'Status' in *Outcome Parameter Set* := TRY AGAIN

'Start' in *Outcome Parameter Set* := B

'L1' in *Error Indication* := Return Code

Initialize(*Discretionary Data*)

AddToList(GetTLV(TagOf(*Error Indication*)), *Discretionary Data*)

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*))) Signal

S3.8

IF [SW12 = '9000']

THEN

GOTO S3.10

ELSE

GOTO S3.9.1

ENDIF

S3.9.1

'L2' in *Error Indication* := STATUS BYTES

'SW12' in *Error Indication* := SW12

S3.9.2

'Field Off Request' in *Outcome Parameter Set* := N/A
'Status' in *Outcome Parameter Set* := SELECT NEXT
'Start' in *Outcome Parameter Set* := C
Initialize(*Discretionary Data*)
AddToList(GetTLV(TagOf(*Error Indication*)), *Discretionary Data*)
Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*))) Signal

S3.10

Parsing Result := FALSE
IF [(Length of Response Message Data Field > 0) AND
 (Response Message Data Field[1] = '77')]
THEN
 Parsing Result := ParseAndStoreCardResponse(Response Message Data Field)
ELSE
 IF [(Length of Response Message Data Field > 0) AND
 (Response Message Data Field[1] = '80')]
 THEN
 Parse the Response Message Data Field according to section 5.6.3 to
 retrieve the value field:
 IF [Response Message Data Field does not parse correctly OR
 The length of the value field of the Response Message Data
 Field is less than 6 OR
 The length of the value field of the Response Message Data
 Field – 2 is not a multiple of 4 OR
 IsEmpty(TagOf(*Application Interchange Profile*)) OR
 IsEmpty(TagOf(*Application File Locator*))]
 THEN
 Parsing Result := FALSE
 ELSE
 Store the first two bytes of the value field of Response Message
 Data Field in the TLV Database for tag TagOf(*Application
 Interchange Profile*).
 Store from the third up to the last byte of the value field of
 Response Message Data Field in the TLV Database for tag
 TagOf(*Application File Locator*).
 Parsing Result := TRUE
 ENDIF
 ENDIF
ENDIF
ENDIF

S3.11

```
IF      [Parsing Result]
THEN
    GOTO S3.13
ELSE
    GOTO S3.12
ENDIF
```

S3.12

'L2' in *Error Indication* := PARSING ERROR

S3.13

```
IF      [IsNotEmpty(TagOf(Application File Locator)) AND
        IsNotEmpty(TagOf(Application Interchange Profile))]
THEN
    GOTO S3.15 or S3.16
ELSE
    GOTO S3.14
ENDIF
```

S3.14

'L2' in *Error Indication* := CARD DATA MISSING

S3.15

```
IF      ['EMV mode contactless transactions not supported' in Kernel Configuration is
        set]
THEN
    GOTO S3.17
ELSE
    GOTO S3.16
ENDIF
```

S3.16

```
IF      ['EMV mode is supported' in Application Interchange Profile is set]
THEN
    GOTO S3.30 or S3.31
ELSE
    GOTO S3.17 or S3.18
ENDIF
```

S3.17

```
IF      ['Mag-stripe mode contactless transactions not supported' in Kernel
        Configuration is set]
THEN
    GOTO S3.18
ELSE
    GOTO S3.70
ENDIF
```

S3.18

'L2' in *Error Indication* := MAGSTRIPE NOT SUPPORTED

EMV Mode

S3.30

IF [(GetLength(TagOf(*Application File Locator*)) \geq 4) AND
(*Application File Locator*[1:4] = '08010100') AND
'Mag-stripe mode contactless transactions not supported' in *Kernel Configuration* is not set]

THEN

GOTO S3.32

ELSE

GOTO S3.31

ENDIF

S3.31

Active AFL := *Application File Locator*

S3.32

Active AFL := *Application File Locator*[5:n], where n =
GetLength(TagOf(*Application File Locator*))

S3.33

IF ['On device cardholder verification is supported' in *Application Interchange Profile* is set AND
'On device cardholder verification supported' in *Kernel Configuration* is set]

THEN

GOTO S3.35

ELSE

GOTO S3.34

ENDIF

S3.34

Reader Contactless Transaction Limit := *Reader Contactless Transaction Limit (No On-device CVM)*

S3.35

Reader Contactless Transaction Limit := *Reader Contactless Transaction Limit (On-device CVM)*

S3.60

IF ['Relay resistance protocol supported' in *Kernel Configuration* is set AND
'Relay resistance protocol is supported' in *Application Interchange Profile* is set]

THEN

GOTO S3.61

ELSE

GOTO S3.65

ENDIF

S3.61

Generate *Unpredictable Number* as specified in section 8.1 and store in the TLV Database for TagOf(*Unpredictable Number*)

Terminal Relay Resistance Entropy := *Unpredictable Number*

S3.62

Prepare EXCHANGE RELAY RESISTANCE DATA command as specified in section 5.3.

S3.63

A timer is started to measure the time taken by the EXCHANGE RELAY RESISTANCE DATA command

Start Timer ()

S3.64

Send CA(EXCHANGE RELAY RESISTANCE DATA) Signal

S3.65

'Relay resistance performed' in *Terminal Verification Results* :=
RRP NOT PERFORMED

Mag-stripe Mode

S3.70

IF [(GetLength(TagOf(*Application File Locator*))) \geq 4) AND
(*Application File Locator*[1:4] = '08010100')]

THEN

GOTO S3.72

ELSE

GOTO S3.71

ENDIF

S3.71

Active AFL := *Application File Locator*

S3.72

Active AFL := *Application File Locator*[1:4]

S3.73

IF ['On device cardholder verification is supported' in *Application Interchange Profile* is set AND
'On device cardholder verification supported' in *Kernel Configuration* is set]

THEN

GOTO S3.75

ELSE

GOTO S3.74

ENDIF

S3.74

Reader Contactless Transaction Limit := *Reader Contactless Transaction Limit (No On-device CVM)*

S3.75

Reader Contactless Transaction Limit := *Reader Contactless Transaction Limit (On-device CVM)*

S3.76

FOR every entry T in *Tags To Read Yet*

{

IF [IsEmpty(T)]

THEN

AddToList(GetTLV(T), *Data To Send*)

RemoveFromList(T, *Tags To Read Yet*)

ENDIF

}

S3.77

```
IF      [IsEmptyList(Data Needed) OR
        (IsEmptyList(Data To Send) AND IsEmptyList(Tags To Read Yet))]
THEN
    GOTO S3.78
ELSE
    GOTO S3.80
ENDIF
```

S3.78

Send DEK(*Data To Send*, *Data Needed*) Signal
Initialize(*Data To Send*)
Initialize(*Data Needed*)

S3.80

Build command data for READ RECORD for the first record indicated by *Active AFL* as defined in section 5.8

S3.81

Send CA(READ RECORD) Signal

Invalid Response

S3.90.1

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

S3.90.2

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

Initialize(*Discretionary Data*)

AddToList(GetTLV(TagOf(*Error Indication*)), *Discretionary Data*)

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*))) Signal

6.6 State R1 – Waiting for Exchange Relay Resistance Data Response

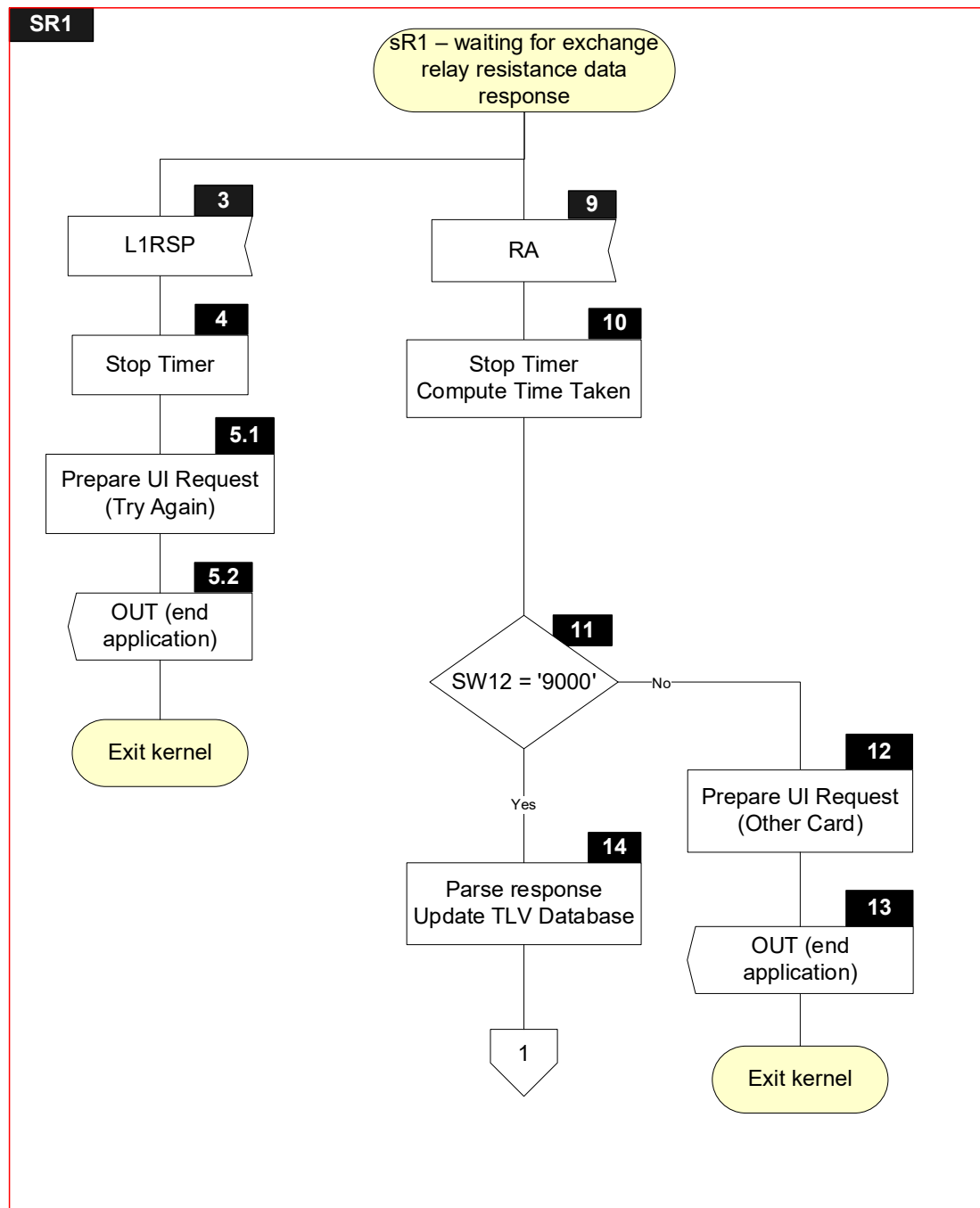
6.6.1 Local Variables

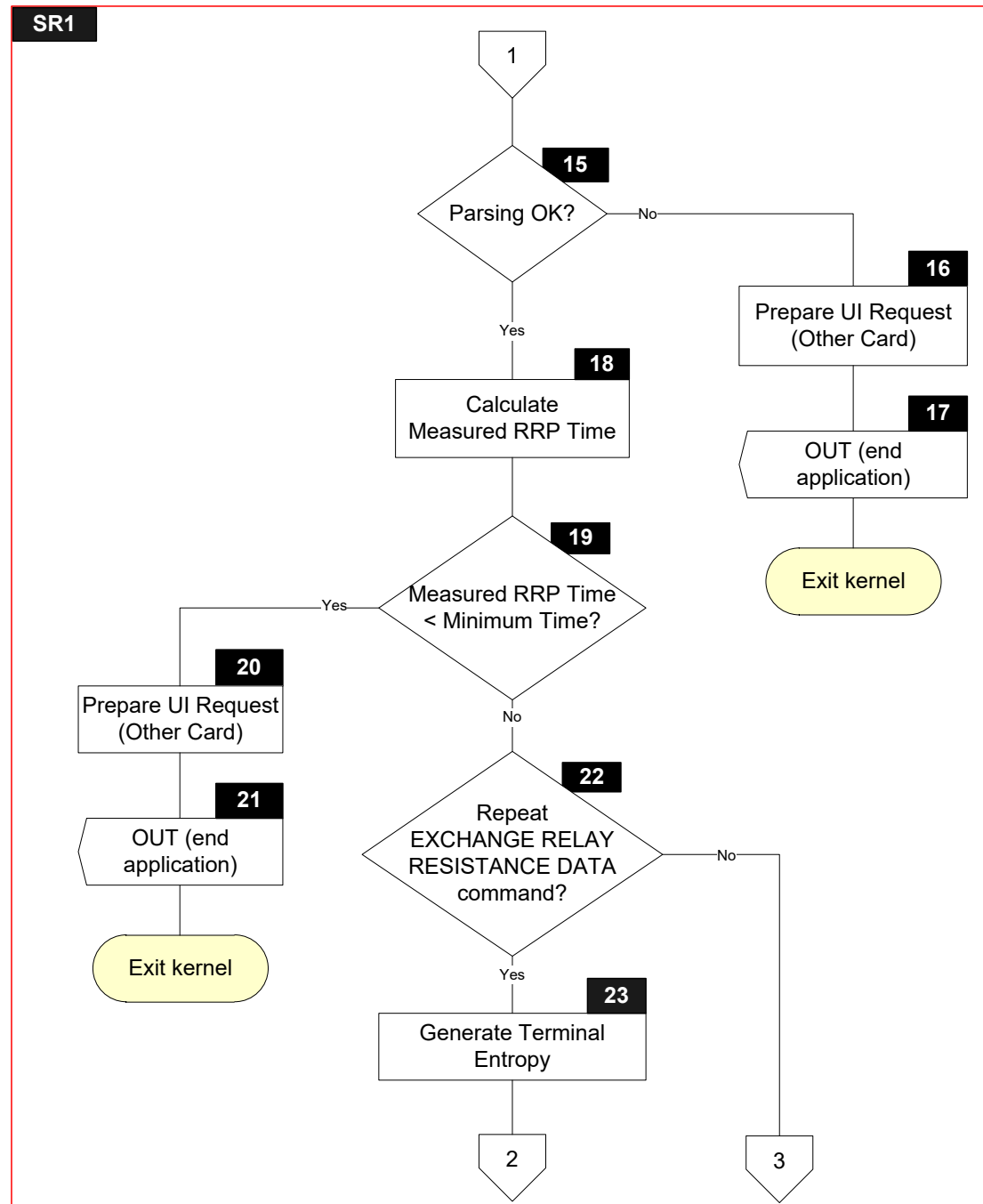
Name	Length	Format	Description
Return Code	1	b	Value returned with L1RSP Signal (TIME OUT ERROR, PROTOCOL ERROR, TRANSMISSION ERROR)
Sync Data ⁽¹⁾	var.	b	List of data objects returned with DET Signal
Parsing Result	1	b	Boolean used to store result of parsing a TLV string
SW12	2	b	Status bytes
Response Message Data Field	var. up to 256	b	TLV encoded string included in R-APDU of EXCHANGE RELAY RESISTANCE DATA
Time Taken	4	b	Time of processing the EXCHANGE RELAY RESISTANCE DATA command measured by Timer. Time Taken is expressed in microseconds.
T	var.	b	Tag of TLV encoded string
L	var.	b	Length of TLV encoded string
V	var. up to 253	b	Value of TLV encoded string

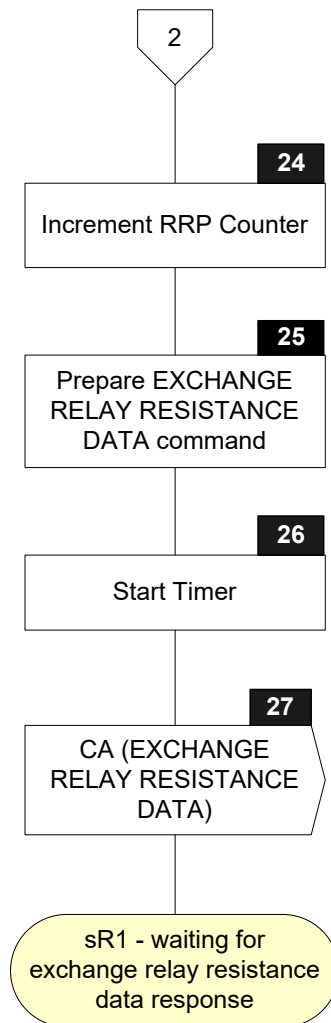
⁽¹⁾ Only for the DE/DS Implementation Option

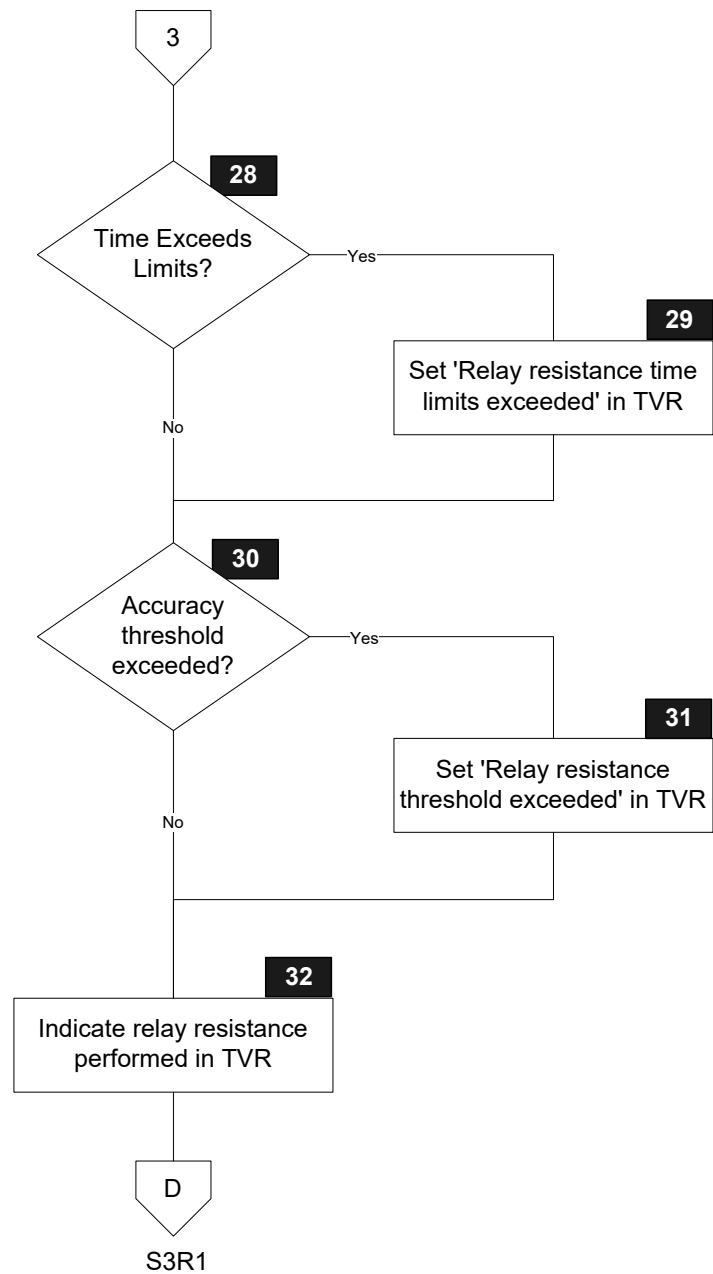
6.6.2 Flow Diagram

Figure 6.5 shows the flow diagram of SR1 – waiting for exchange relay resistance data response. Symbols in this diagram are labelled SR1.X.

Figure 6.5—State R1 Flow Diagram



SR1

SR1

6.6.3 Processing

SR1.3

Receive L1RSP Signal with Return Code

SR1.4

Stop Timer

SR1.5.1

'Message Identifier' in *User Interface Request Data* := TRY AGAIN

'Status' in *User Interface Request Data* := READY TO READ

'Hold Time' in *User Interface Request Data* := '000000'

SR1.5.2

'Status' in *Outcome Parameter Set* := END APPLICATION

'Start' in *Outcome Parameter Set* := B

SET 'UI Request on Restart Present' in *Outcome Parameter Set*

'L1' in *Error Indication* := Return Code

'Msg On Error' in *Error Indication* := TRY AGAIN

CreateEMVDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*))) Signal

SR1.9

Receive RA Signal with Response Message Data Field and SW12

SR1.10

Stop Timer

Compute Time Taken from the start and stop times.

SR1.11

IF [SW12 = '9000']

THEN

GOTO SR1.14

ELSE

GOTO SR1.12

ENDIF

SR1.12

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

SR1.13

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

'L2' in *Error Indication* := STATUS BYTES

'SW12' in *Error Indication* := SW12

CreateEMVDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*)))) Signal

SR1.14

Parsing Result := FALSE

IF [(Length of Response Message Data Field > 11) AND
(Response Message Data Field[1] = '80') AND
Length of the value field of the Response Message Data Field = 10]

THEN

Store the first four bytes of the value field of Response Message Data Field in the TLV Database for tag TagOf(*Device Relay Resistance Entropy*).

Store from the fifth up to the sixth byte of the value field of Response Message Data Field in the TLV Database for tag TagOf(*Min Time For Processing Relay Resistance APDU*).

Store from the seventh up to the eighth byte of the value field of Response Message Data Field in the TLV Database for tag TagOf(*Max Time For Processing Relay Resistance APDU*).

Store from the ninth up to the tenth byte of the value field of Response Message Data Field in the TLV Database for tag TagOf(*Device Estimated Transmission Time For Relay Resistance R-APDU*).

Parsing Result := TRUE

ENDIF

SR1.15

IF [Parsing Result]

THEN

GOTO SR1.18

ELSE

GOTO SR1.16

ENDIF

SR1.16

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

SR1.17

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

'L2' in *Error Indication* := PARSING ERROR

CreateEMVDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*))) Signal

SR1.18

Measured Relay Resistance Processing Time := MAX (0, (Time Taken div 100) –
Terminal Expected Transmission Time For Relay Resistance C-APDU – MIN (*Device*
Estimated Transmission Time For Relay Resistance R-APDU , *Terminal Expected*
Transmission Time For Relay Resistance R-APDU))

Note that the implementation should compensate for any known fixed timing latency. All implementations will have some inevitable delay between starting the timer and sending the C-APDU and between receiving the R-APDU and stopping the timer. If this latency is predictable and can be compensated for by the implementation then it does not need to be compensated by increasing the maximum grace period.

SR1.19

IF [*Measured Relay Resistance Processing Time* < MAX (0, (*Minimum Time For*
Processing Relay Resistance APDU – *Minimum Relay Resistance Grace*
Period))]

THEN

GOTO SR1.20

ELSE

GOTO SR1.22

ENDIF

SR1.20

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

SR1.21

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

'L2' in *Error Indication* := CARD DATA ERROR

CreateEMVDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*))) Signal

SR1.22

IF [(RRP Counter < 2) AND
Measured Relay Resistance Processing Time > *Max Time For Processing
 Relay Resistance APDU + Maximum Relay Resistance Grace Period*]

THEN

GOTO SR1.23

ELSE

GOTO SR1.28

ENDIF

SR1.23

Generate *Unpredictable Number* as specified in section 8.1 and store in the TLV Database for TagOf(*Unpredictable Number*)

Terminal Relay Resistance Entropy := *Unpredictable Number*

SR1.24

RRP Counter := *RRP Counter* + 1

SR1.25

Prepare EXCHANGE RELAY RESISTANCE DATA command as specified in section 5.3.

SR1.26

A timer is started to measure the time taken by the EXCHANGE RELAY RESISTANCE DATA command

Start Timer ()

SR1.27

Send CA(EXCHANGE RELAY RESISTANCE DATA) Signal

SR1.28

IF [*Measured Relay Resistance Processing Time* > (*Max Time For Processing
 Relay Resistance APDU + Maximum Relay Resistance Grace Period*)]

THEN

GOTO SR1.29

ELSE

GOTO SR1.30

ENDIF

SR1.29

SET 'Relay resistance time limits exceeded' in *Terminal Verification Results*

SR1.30

```

IF      [(Device Estimated Transmission Time For Relay Resistance R-APDU  $\neq$  0)
AND
(Terminal Expected Transmission Time For Relay Resistance R-APDU  $\neq$  0)]
THEN
    IF      [((( Device Estimated Transmission Time For Relay Resistance R-APDU * 100) div Terminal Expected Transmission Time For Relay Resistance R-APDU) < Relay Resistance Transmission Time Mismatch Threshold)
OR
(( Terminal Expected Transmission Time For Relay Resistance R-APDU * 100) div Device Estimated Transmission Time For Relay Resistance R-APDU) < Relay Resistance Transmission Time Mismatch Threshold)
OR
(MAX (0, (Measured Relay Resistance Processing Time – Min Time For Processing Relay Resistance APDU) ) > Relay Resistance Accuracy Threshold)]
    THEN
        GOTO SR1.31
    ELSE
        GOTO SR1.32
    ENDIF
ELSE
    GOTO SR1.31
ENDIF

```

SR1.31

SET 'Relay resistance threshold exceeded' in *Terminal Verification Results*

SR1.32

'Relay resistance performed' in *Terminal Verification Results* := RRP PERFORMED

6.7 States 3, R1 – Common Processing

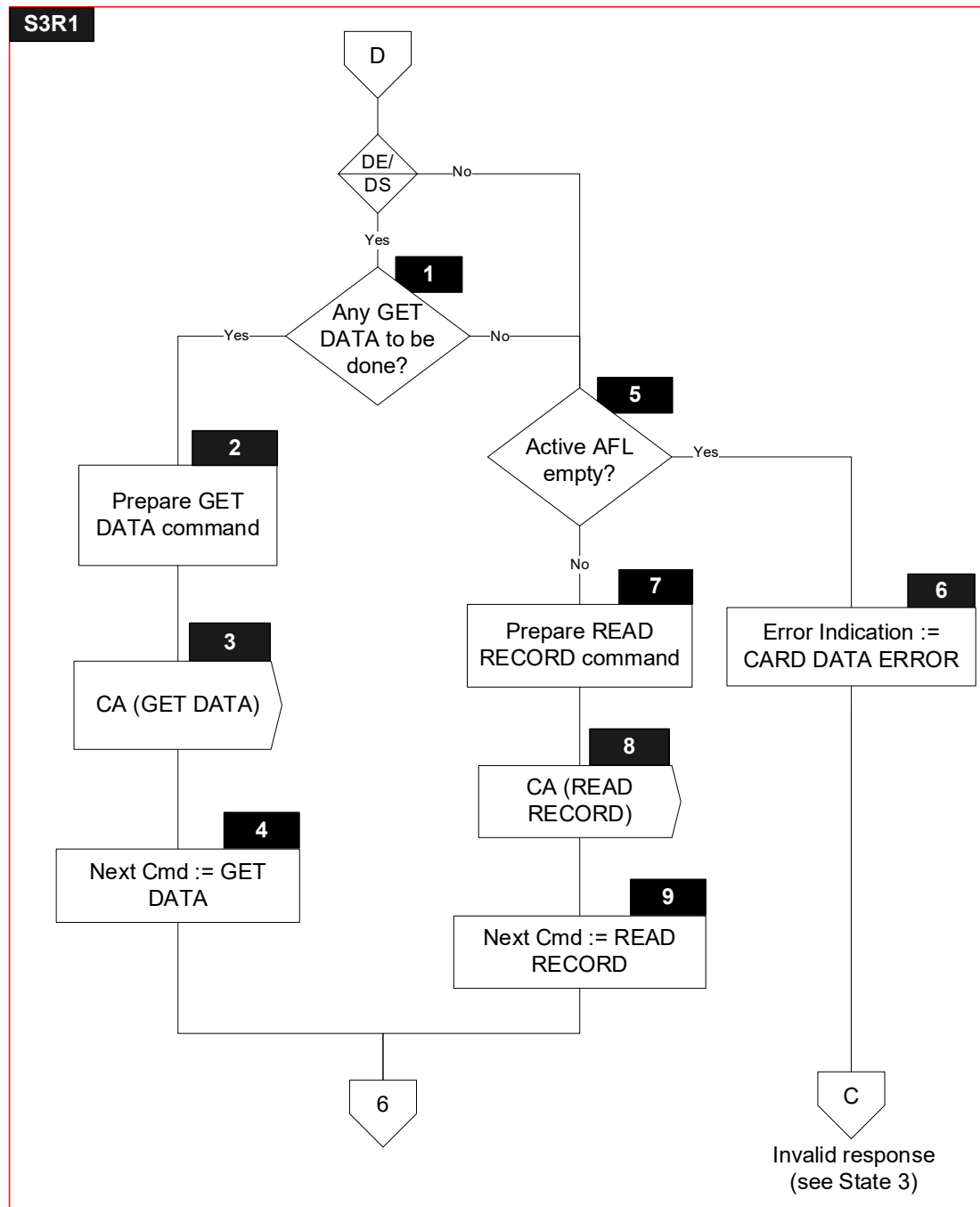
6.7.1 Local Variables

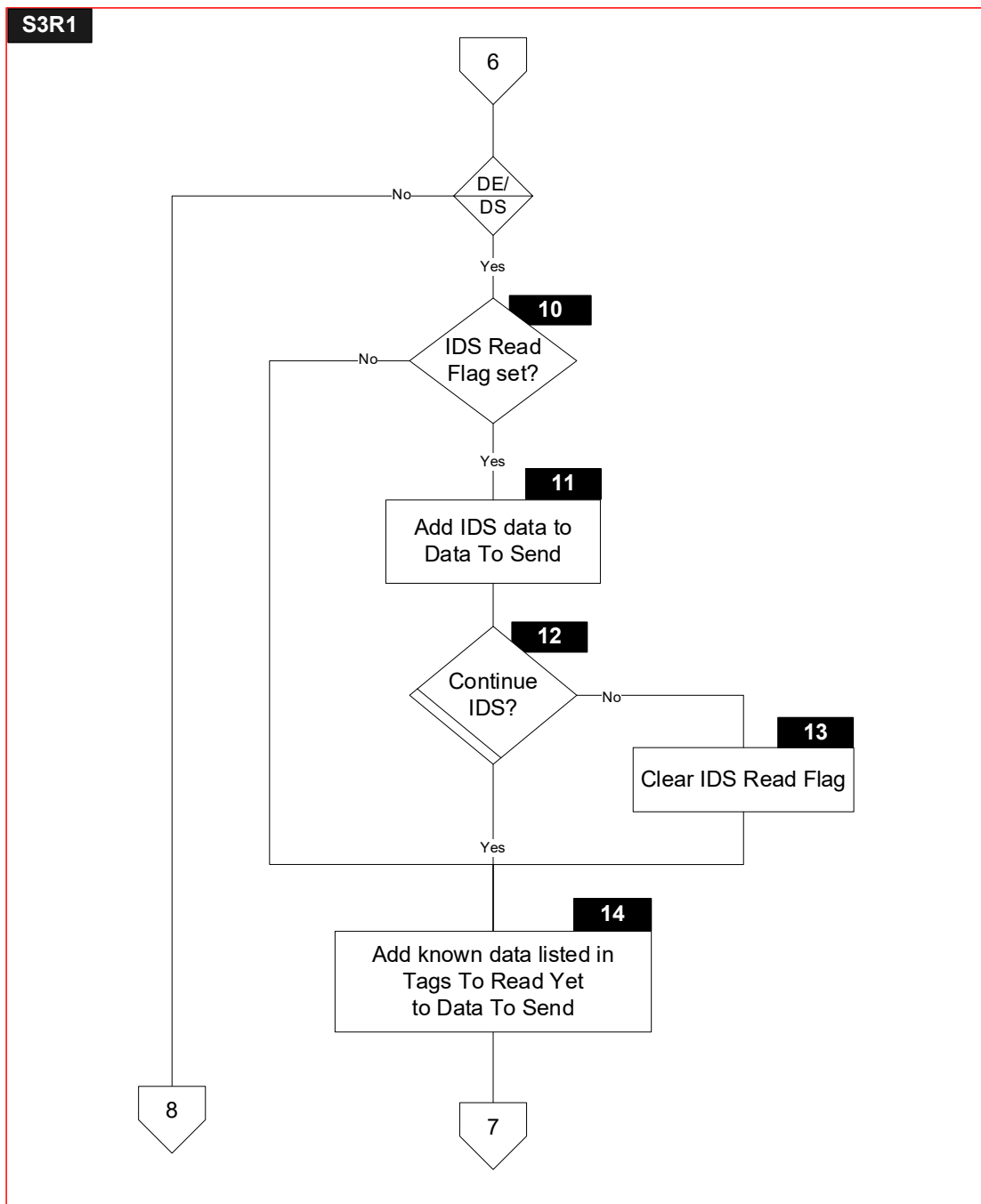
Local variables for common processing are defined in states 3 and R1.

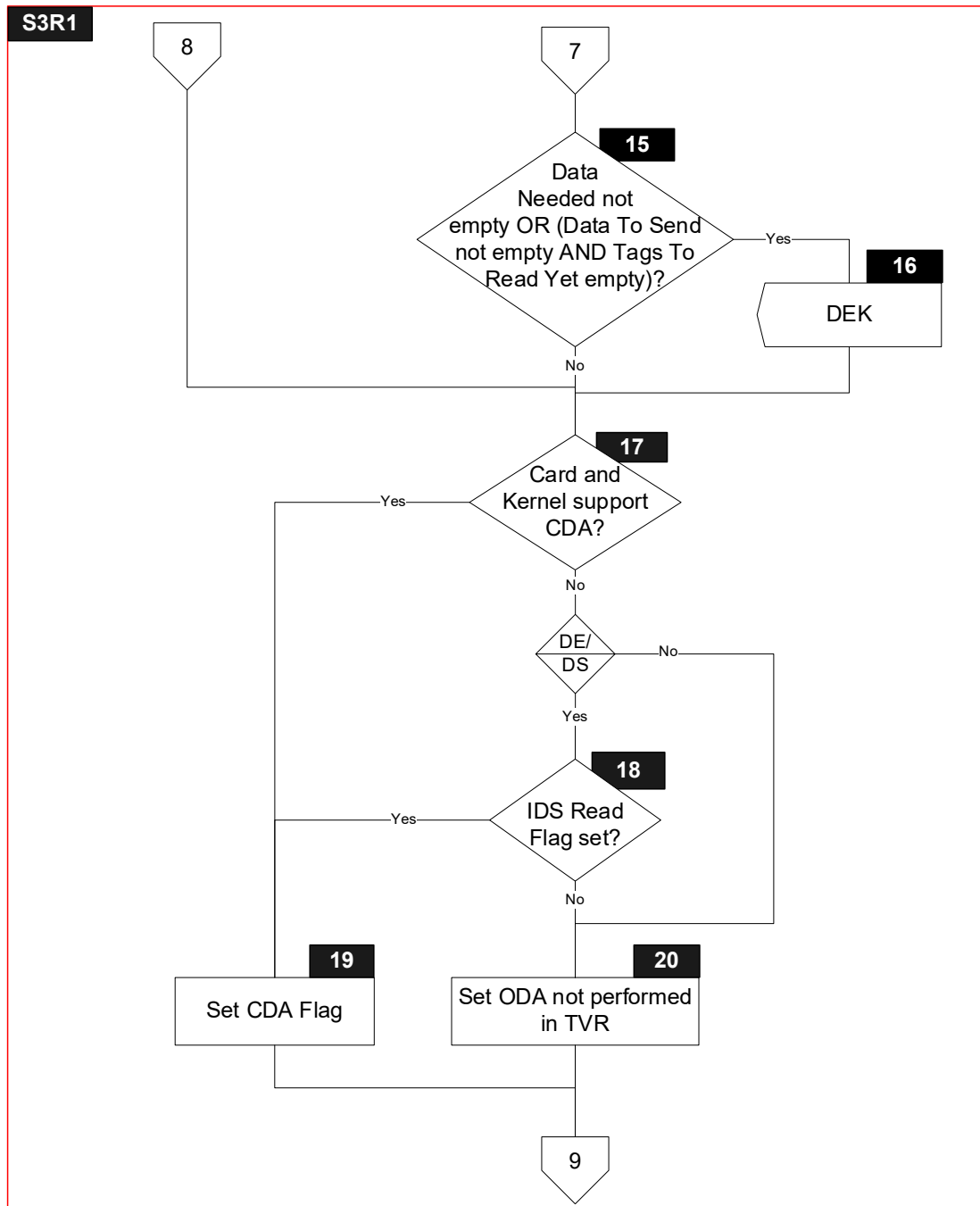
6.7.2 Flow Diagram

Figure 6.6 shows the flow diagram for common processing between states 3 and R1. Symbols in this diagram are labelled S3R1.X.

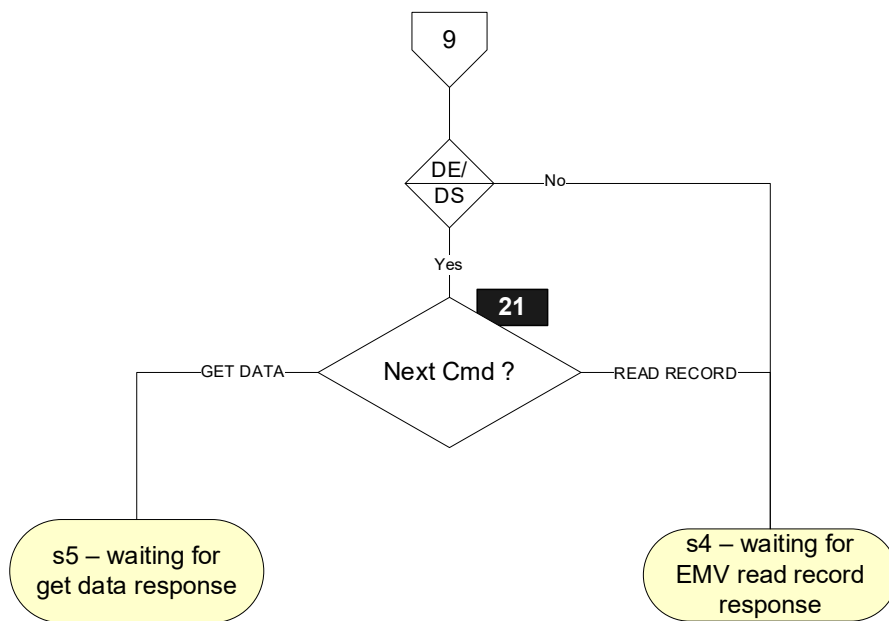
Figure 6.6—States 3 and R1 – Common Processing – Flow Diagram







S3R1



6.7.3 Processing

Note that symbols S3R1.1, S3R1.2, S3R1.3, S3R1.4, S3R1.10, S3R1.11, S3R1.12, S3R1.13, S3R1.14, S3R1.15, S3R1.16, S3R1.18 and S3R1.21 are only implemented for the DE/DS Implementation Option.

S3R1.1

Active Tag := GetNextGetDataTagFromList(*Tags To Read Yet*)

IF [*Active Tag* = NULL]

THEN

GOTO S3R1.5

ELSE

GOTO S3R1.2

ENDIF

S3R1.2

Build GET DATA command for *Active Tag* as defined in section 5.5

S3R1.3

Send CA(GET DATA) Signal

S3R1.4

'Next Cmd' in *Next Cmd* := GET DATA

S3R1.5

IF [*Active AFL* is empty]

THEN

GOTO S3R1.6

ELSE

GOTO S3R1.7

ENDIF

S3R1.6

'L2' in *Error Indication* := CARD DATA ERROR

S3R1.7

Build READ RECORD command for the first record indicated by *Active AFL* as defined in section 5.8

S3R1.8

Send CA(READ RECORD) Signal

S3R1.9

'Next Cmd' in *Next Cmd* := READ RECORD

S3R1.10

```
IF      ['Read' in IDS Status is set]
THEN
    GOTO S3R1.11
ELSE
    GOTO S3R1.14
ENDIF
```

S3R1.11

```
IF      [IsEmpty(TagOf(DS Slot Availability))]
THEN
    AddToList(GetTLV(TagOf(DS Slot Availability)), Data To Send)
ENDIF
IF      [IsEmpty(TagOf(DS Summary 1))]
THEN
    AddToList(GetTLV(TagOf(DS Summary 1)), Data To Send)
ENDIF
IF      [IsEmpty(TagOf(DS Unpredictable Number))]
THEN
    AddToList(GetTLV(TagOf(DS Unpredictable Number)), Data To Send)
ENDIF
IF      [IsEmpty(TagOf(DS Slot Management Control))]
THEN
    AddToList(GetTLV(TagOf(DS Slot Management Control)), Data To Send)
ENDIF
IF      [IsPresent(TagOf(DS ODS Card))]
THEN
    AddToList(GetTLV(TagOf(DS ODS Card)), Data To Send)
ENDIF
AddToList(GetTLV(TagOf(Unpredictable Number)), Data To Send)
```

S3R1.12

Continue with IDS when:

- *DS Requested Operator ID* is not known by the Card, but all the necessary data objects are returned by the Card to perform an IDS write, or
- *DS Requested Operator ID* is known by the Card

This is done as follows:

```
IF      [(IsNotEmpty(TagOf(DS Slot Availability)) AND
          IsNotEmpty(TagOf(DS Summary 1)) AND
          IsNotEmpty(TagOf(DS Unpredictable Number)) AND
          IsNotPresent(TagOf(DS ODS Card)))
        OR
        (IsNotEmpty(TagOf(DS Summary 1)) AND
          IsPresent(TagOf(DS ODS Card))) ]
THEN
    GOTO S3R1.14
ELSE
    GOTO S3R1.13
ENDIF
```

S3R1.13

CLEAR 'Read' in *IDS Status*

S3R1.14

FOR every entry T in *Tags To Read Yet*

```
{
    IF      [IsNotEmpty(T)]
    THEN
        AddToList(GetTLV(T), Data To Send)
        RemoveFromList(T, Tags To Read Yet)
    ENDIF
}
```

S3R1.15

```
IF      [IsNotEmptyList(Data Needed) OR
        (IsNotEmptyList(Data To Send) AND IsEmptyList(Tags To Read Yet))]
THEN
    GOTO S3R1.16
ELSE
    GOTO S3R1.17
ENDIF
```

S3R1.16

Send DEK(*Data To Send, Data Needed*) Signal

Initialize(*Data To Send*)

Initialize(*Data Needed*)

S3R1.17

IF ['CDA supported' in *Application Interchange Profile* is set
AND 'CDA' in *Terminal Capabilities* is set]

THEN

GOTO S3R1.19

ELSE

GOTO S3R1.18

ENDIF

S3R1.18

IF ['Read' in *IDS Status* is set]

THEN

GOTO S3R1.19

ELSE

GOTO S3R1.20

ENDIF

S3R1.19

SET 'CDA' in *ODA Status*

S3R1.20

SET 'Offline data authentication was not performed' in *Terminal Verification Results*

S3R1.21

IF ['Next Cmd' in *Next Cmd* = READ RECORD]

THEN

GOTO s4 – waiting for EMV read record response

ELSE

GOTO s5 – waiting for get data response

ENDIF

6.8 State 4 – Waiting for EMV Read Record Response

6.8.1 Local Variables

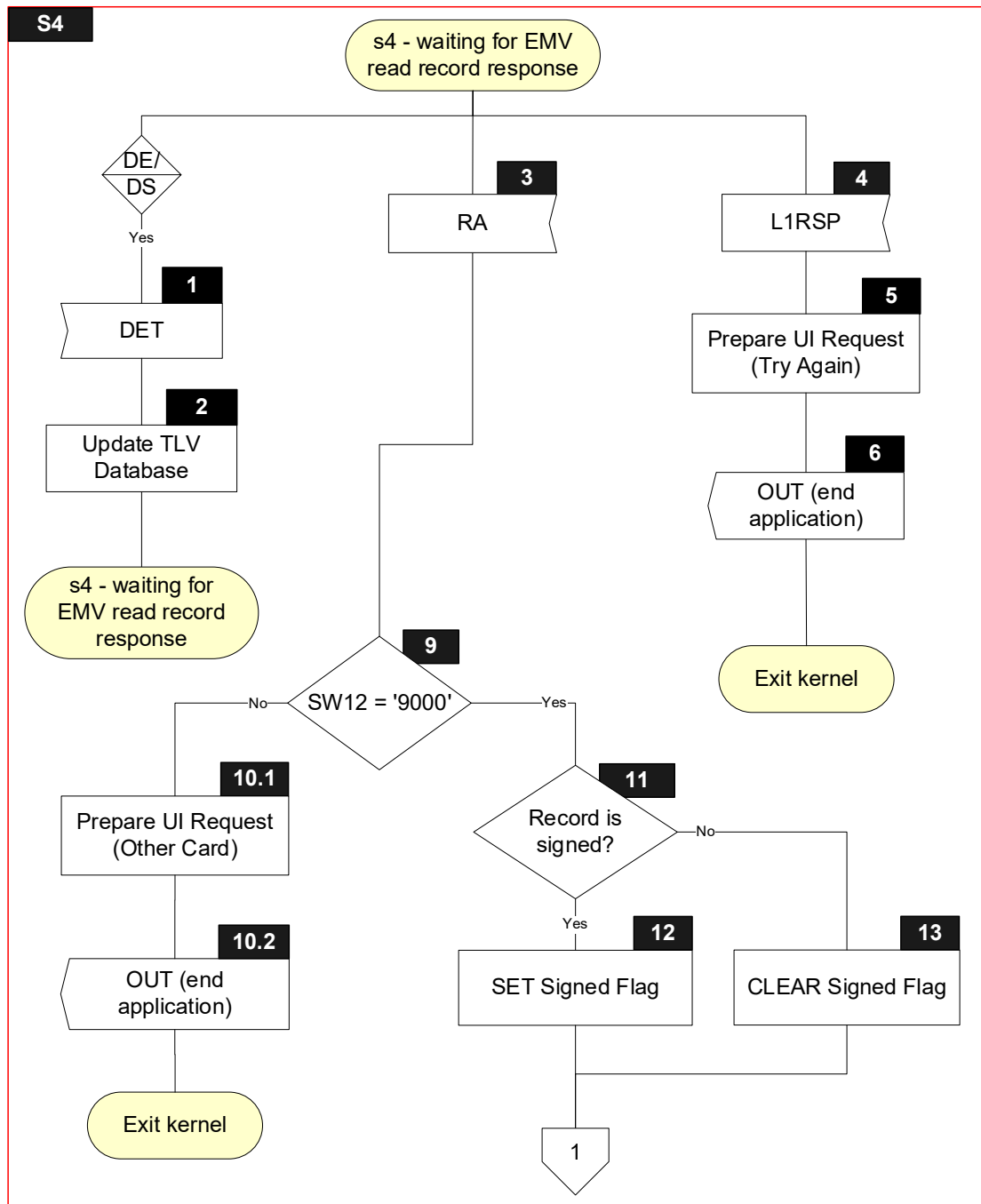
Name	Length	Format	Description
Return Code	1	b	Value returned with L1RSP Signal (TIME OUT ERROR, PROTOCOL ERROR, TRANSMISSION ERROR)
Sync Data ⁽¹⁾	var.	b	List of data objects returned with DET Signal
Parsing Result	1	b	Boolean used to store result of parsing a TLV string
SW12	2	b	Status bytes
Record	var. up to 254	b	Response Message Data Field of the R-APDU of READ RECORD
Signed Flag	1	b	Boolean used to indicate if current record is signed
Sfi	1	b	SFI of current record
T	var.	b	Tag of TLV encoded string
L	var.	b	Length of TLV encoded string
V	var. up to 253	b	Value of TLV encoded string

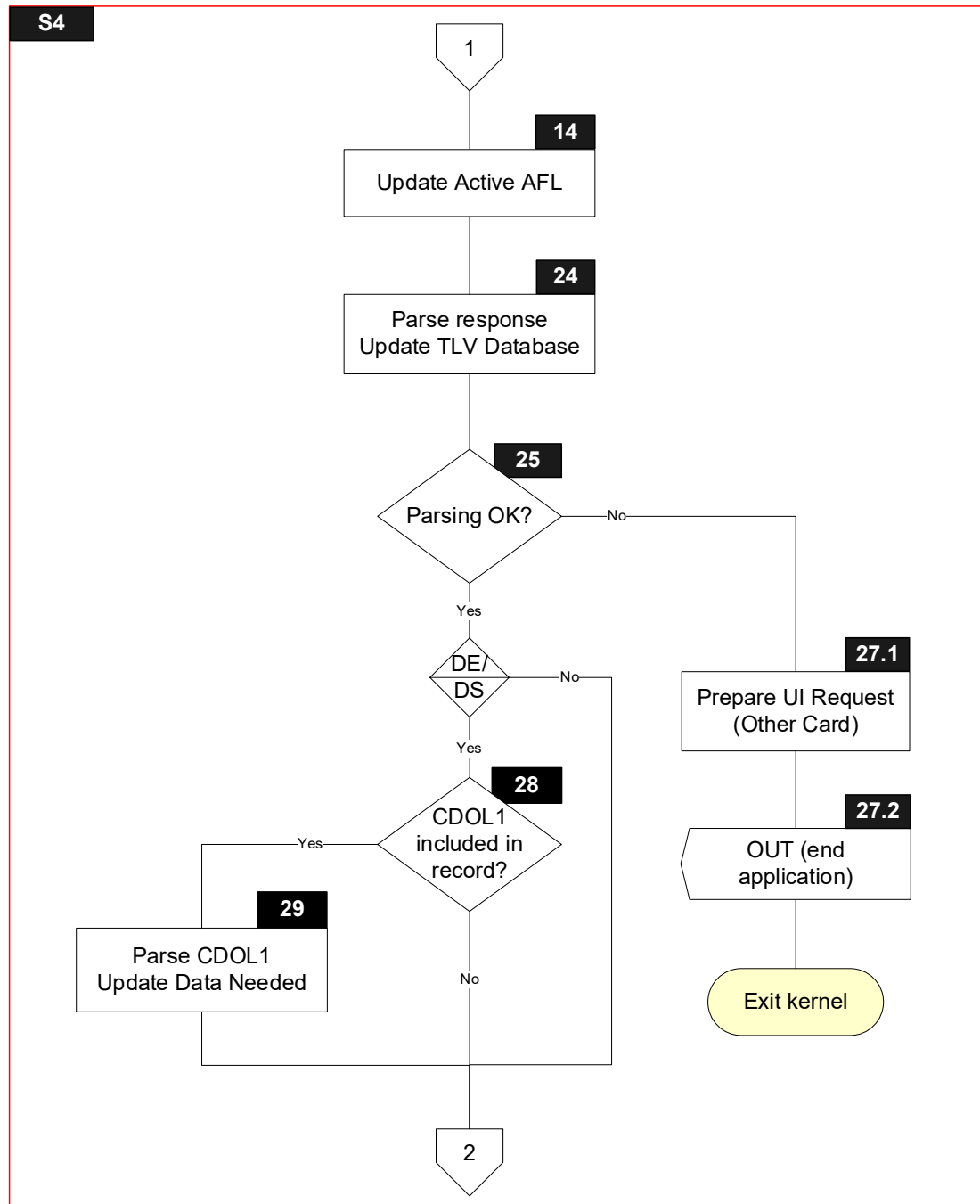
⁽¹⁾ Only for the DE/DS Implementation Option

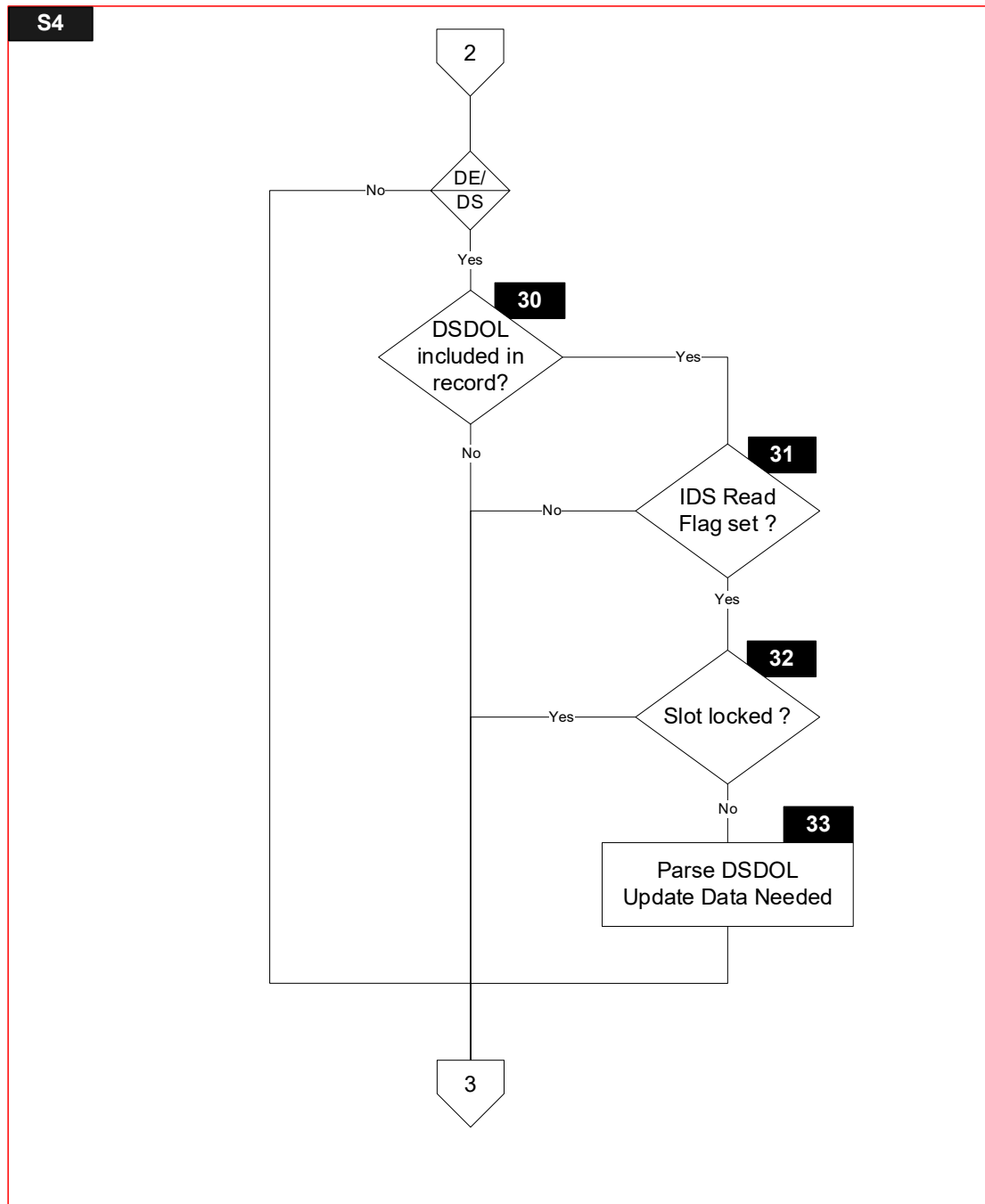
6.8.2 Flow Diagram

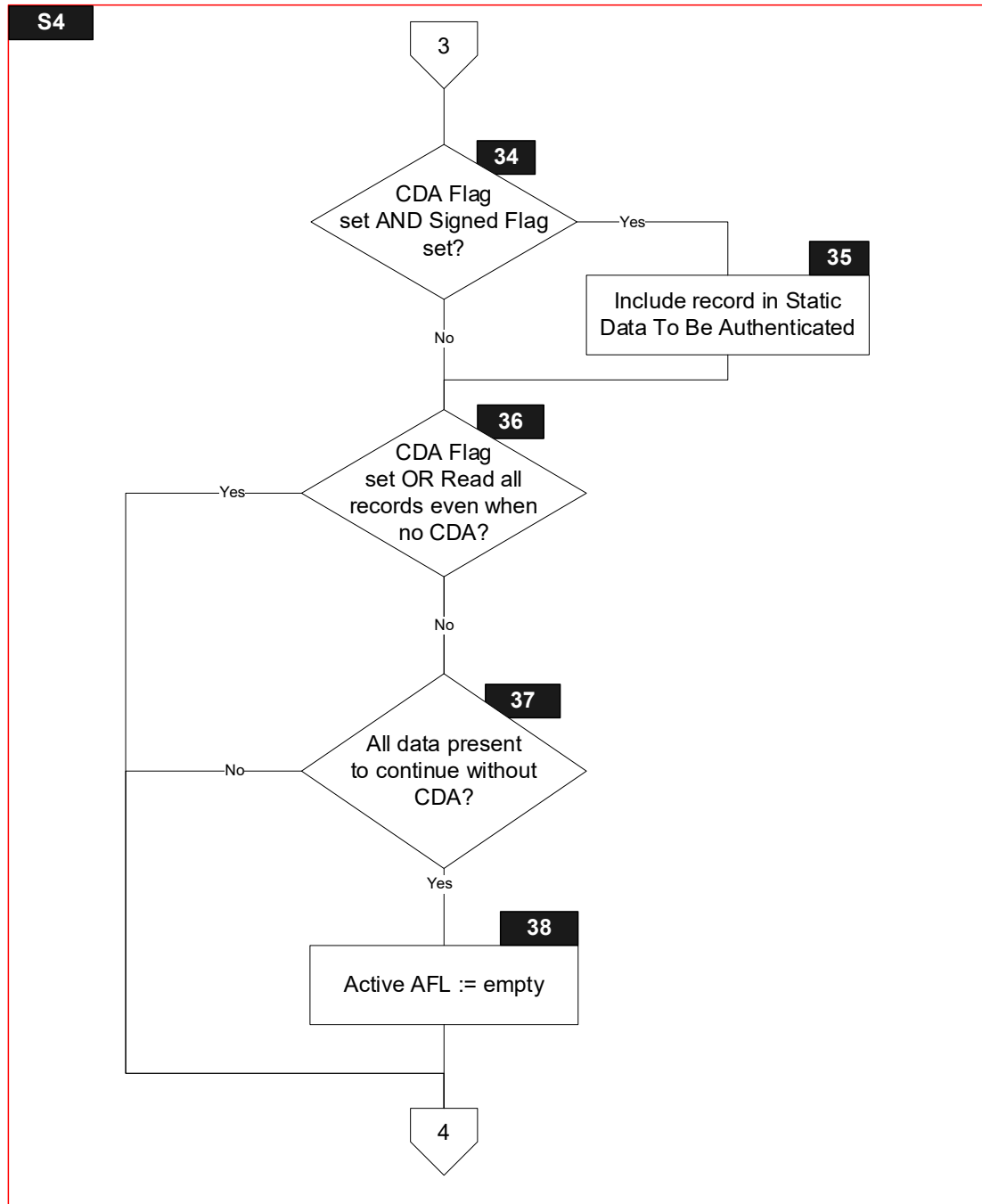
Figure 6.7 shows the flow diagram of s4 – waiting for EMV read record response. Symbols in this diagram are labelled S4.X.

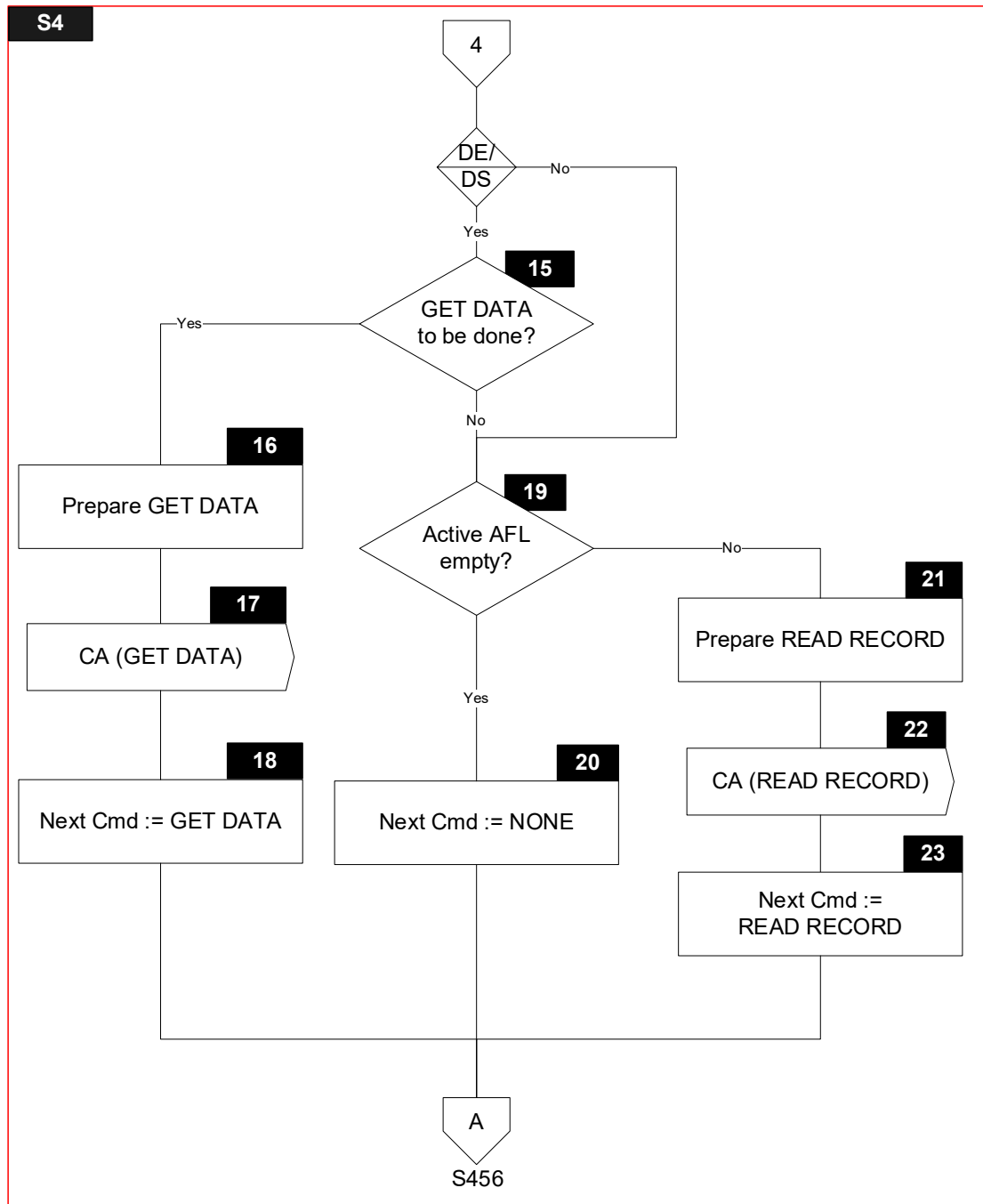
Figure 6.7—State 4 Flow Diagram











6.8.3 Processing

Note that symbols S4.1, S4.2, S4.15, S4.16, S4.17, S4.18, S4.28, S4.29, S4.30, S4.31, S4.32 and S4.33 are only implemented for the DE/DS Implementation Option.

S4.1

Receive DET Signal with Sync Data

S4.2

UpdateWithDetData(Sync Data)

S4.3

Receive RA Signal with Record and SW12

S4.4

Receive L1RSP Signal with Return Code

S4.5

'Message Identifier' in *User Interface Request Data* := TRY AGAIN

'Status' in *User Interface Request Data* := READY TO READ

'Hold Time' in *User Interface Request Data* := '000000'

S4.6

'Status' in *Outcome Parameter Set* := END APPLICATION

'Start' in *Outcome Parameter Set* := B

SET 'UI Request on Restart Present' in *Outcome Parameter Set*

'L1' in *Error Indication* := Return Code

'Msg On Error' in *Error Indication* := TRY AGAIN

CreateEMVDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*))) Signal

S4.9

IF [SW12 = '9000']

THEN

GOTO S4.11

ELSE

GOTO S4.10.1

ENDIF

S4.10.1

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

S4.10.2

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

'L2' in *Error Indication* := STATUS BYTES

'SW12' in *Error Indication* := SW12

CreateEMVDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*)))) Signal

S4.11

IF [Active AFL indicates first record (i.e. current record) is signed]

THEN

 GOTO S4.12

ELSE

 GOTO S4.13

ENDIF

S4.12

SET Signed Flag

S4.13

CLEAR Signed Flag

S4.14

Sfi := SFI of first record in *Active AFL*

Remove first record from *Active AFL*

S4.24

IF [Sfi ≤ 10]

THEN

 IF [(Length of Record > 0) AND (Record[1] = '70')]

 THEN

 Parsing Result := ParseAndStoreCardResponse(Record)

 ELSE

 Parsing Result := FALSE

 ENDIF

ELSE

 Processing of records in proprietary files is beyond the scope of this
 specification

ENDIF

S4.25

```
IF      [Parsing Result]
THEN
    GOTO S4.28 or S4.34
ELSE
    GOTO S4.27.1
ENDIF
```

S4.27.1

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

S4.27.2

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

'L2' in *Error Indication* := PARSING ERROR

CreateEMVDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*)))) Signal

S4.28

```
IF      [Record includes data object with tag equal to TagOf(CDOLI)]
THEN
    GOTO S4.29
ELSE
    GOTO S4.30
ENDIF
```

S4.29

FOR every TL in *CDOLI*

```
{
    IF      [IsEmpty(T)]
    THEN
        AddToList(T, Data Needed)
    ENDIF
}
```

S4.30

```
IF      [Record includes data object with tag equal to TagOf(DSDOL)]
THEN
    GOTO S4.31
ELSE
    GOTO S4.34
ENDIF
```

S4.31

```
IF      ['Read' in IDS Status is set]
THEN
    GOTO S4.32
ELSE
    GOTO S4.34
ENDIF
```

S4.32

```
IF      [IsEmpty(TagOf(DS Slot Management Control)) AND
        'Locked slot' in DS Slot Management Control is set]
THEN
    GOTO S4.34
ELSE
    GOTO S4.33
ENDIF
```

S4.33

```
FOR every TL in DSDOL
{
    IF      [IsEmpty(T)]
    THEN
        AddToList(T, Data Needed)
    ENDIF
}
```

S4.34

```
IF      [Signed Flag AND 'CDA' in ODA Status is set]
THEN
    GOTO S4.35
ELSE
    GOTO S4.36
ENDIF
```

S4.35

```
IF      [Sfi ≤ 10]
THEN
    IF      [Enough space left in Static Data To Be Authenticated to append
              Record (without tag '70' and length)]
    THEN
        Append Record (excluding tag '70' and length) at the end of Static
        Data To Be Authenticated
    ELSE
        SET 'CDA failed' in Terminal Verification Results
    ENDIF
ELSE
    IF      [(Record[1] = '70') AND
              Record is TLV encoded AND
              Enough space left in Static Data To Be Authenticated to append
              Record]
    THEN
        Append Record (including tag '70' and length) at the end of Static Data
        To Be Authenticated
    ELSE
        SET 'CDA failed' in Terminal Verification Results
    ENDIF
ENDIF
```

S4.36

```
IF      ['CDA' in ODA Status is set OR 'Read all records even when no CDA' in
        Kernel Configuration is set]
THEN
    GOTO S4.15 or S4.19
ELSE
    GOTO S4.37
ENDIF
```

S4.37

IF [IsEmpty(TagOf(*Application Expiration Date*)) AND
IsEmpty(TagOf(*Application PAN*)) AND
IsEmpty(TagOf(*Application PAN Sequence Number*)) AND
IsEmpty(TagOf(*Application Usage Control*)) AND
IsEmpty(TagOf(*CVM List*)) AND
IsEmpty(TagOf(*Issuer Action Code – Default*)) AND
IsEmpty(TagOf(*Issuer Action Code – Denial*)) AND
IsEmpty(TagOf(*Issuer Action Code – Online*)) AND
IsEmpty(TagOf(*Issuer Country Code*)) AND
IsEmpty(TagOf(*Track 2 Equivalent Data*)) AND
IsEmpty(TagOf(*CDOL1*))]
THEN
GOTO S4.38
ELSE
GOTO S4.15 or S4.19
ENDIF

S4.38

Active AFL := empty

S4.15

Active Tag := GetNextGetDataTagFromList (*Tags To Read Yet*)

IF [*Active Tag* is not NULL]
THEN
GOTO S4.16
ELSE
GOTO S4.19
ENDIF

S4.16

Prepare GET DATA command for *Active Tag* as specified in section 5.5

S4.17

Send CA(GET DATA command) Signal

S4.18

'Next Cmd' in *Next Cmd* := GET DATA
GOTO S456.1

S4.19

IF [Active AFL is empty]
THEN
 GOTO S4.20
ELSE
 GOTO S4.21
ENDIF

S4.20

'Next Cmd' in *Next Cmd* := NONE
GOTO S456.1

S4.21

Prepare READ RECORD command for first record in *Active AFL* as specified in section 5.8

S4.22

Send CA(READ RECORD command) Signal

S4.23

'Next Cmd' in *Next Cmd* := READ RECORD
GOTO S456.1

6.9 State 5 – Waiting for Get Data Response

State 5 is a state specific to SDS processing and is only implemented for the DE/DS Implementation Option.

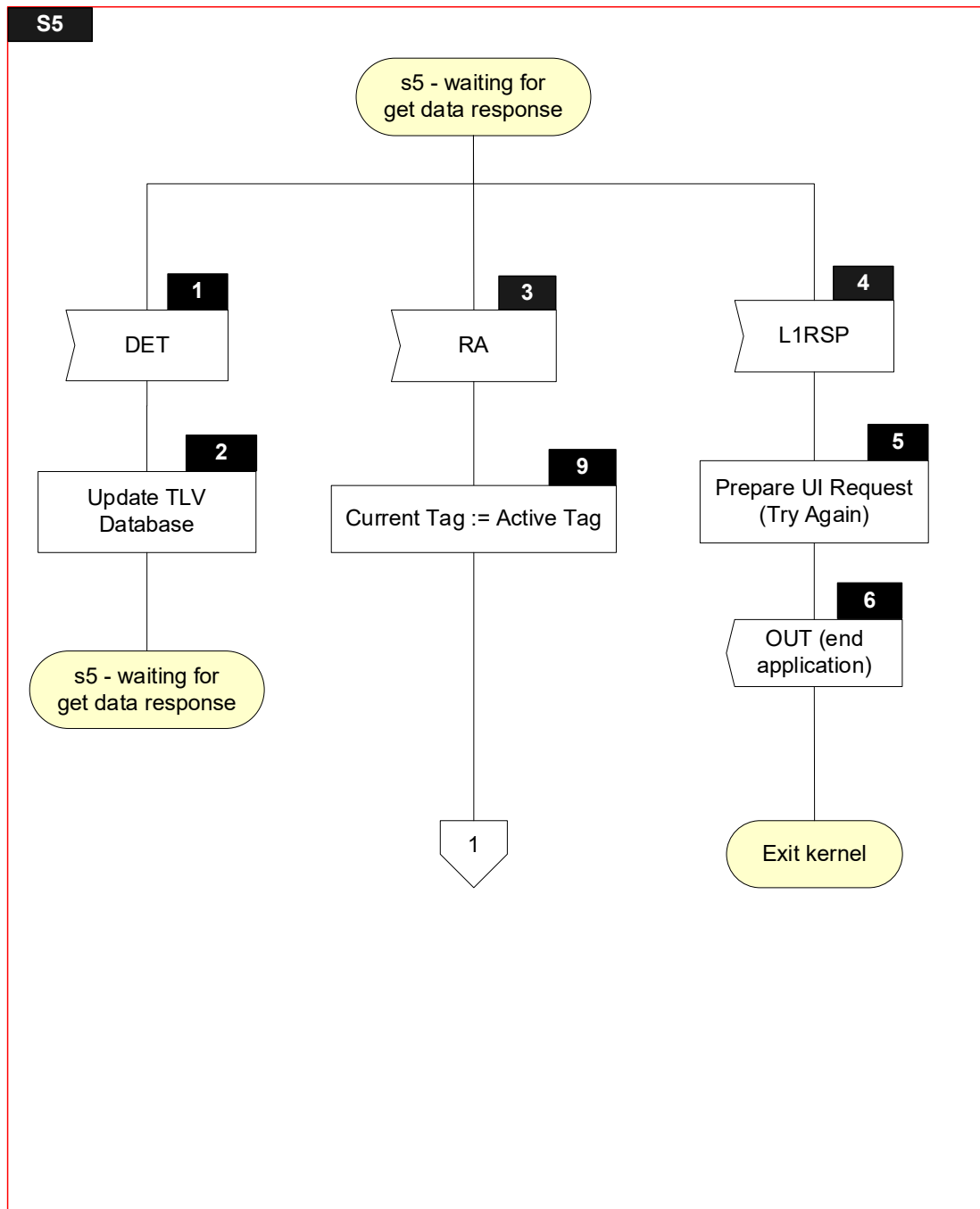
6.9.1 Local Variables

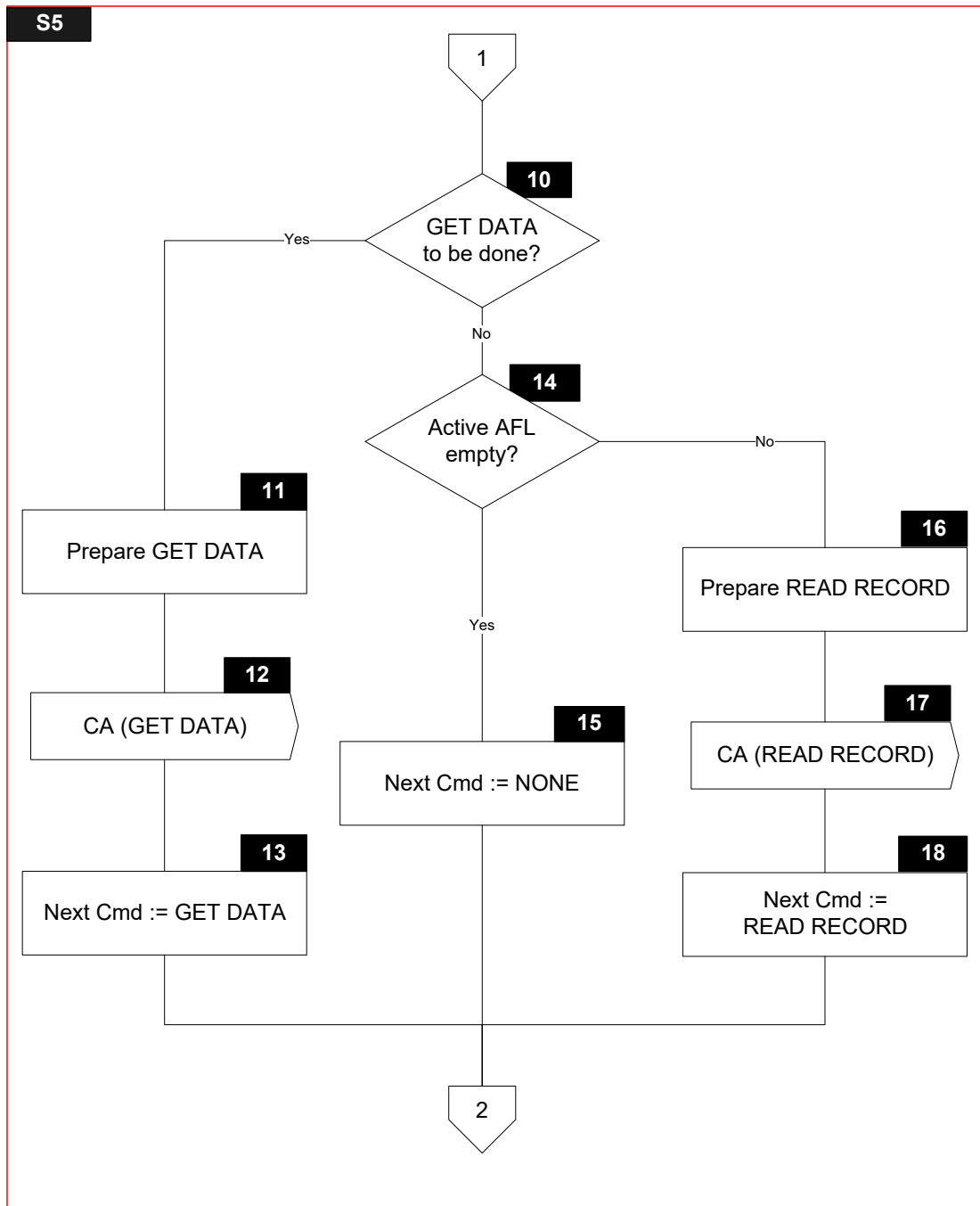
Name	Length	Format	Description
Return Code	1	b	Value returned with L1RSP Signal (TIME OUT ERROR, PROTOCOL ERROR, TRANSMISSION ERROR)
Sync Data	var.	b	List of data objects returned with DET Signal
Parsing Result	1	b	Boolean used to store result of parsing a TLV string
SW12	2	b	Status bytes
Response Message Data Field	var. up to 256	b	TLV encoded string included in R-APDU of GET DATA
Current Tag	var.	b	Tag indicating the tag of the current GET DATA
T	var.	b	Tag of TLV encoded string
L	var.	b	Length of TLV encoded string
V	var. up to 252	b	Value of TLV encoded string

6.9.2 Flow Diagram

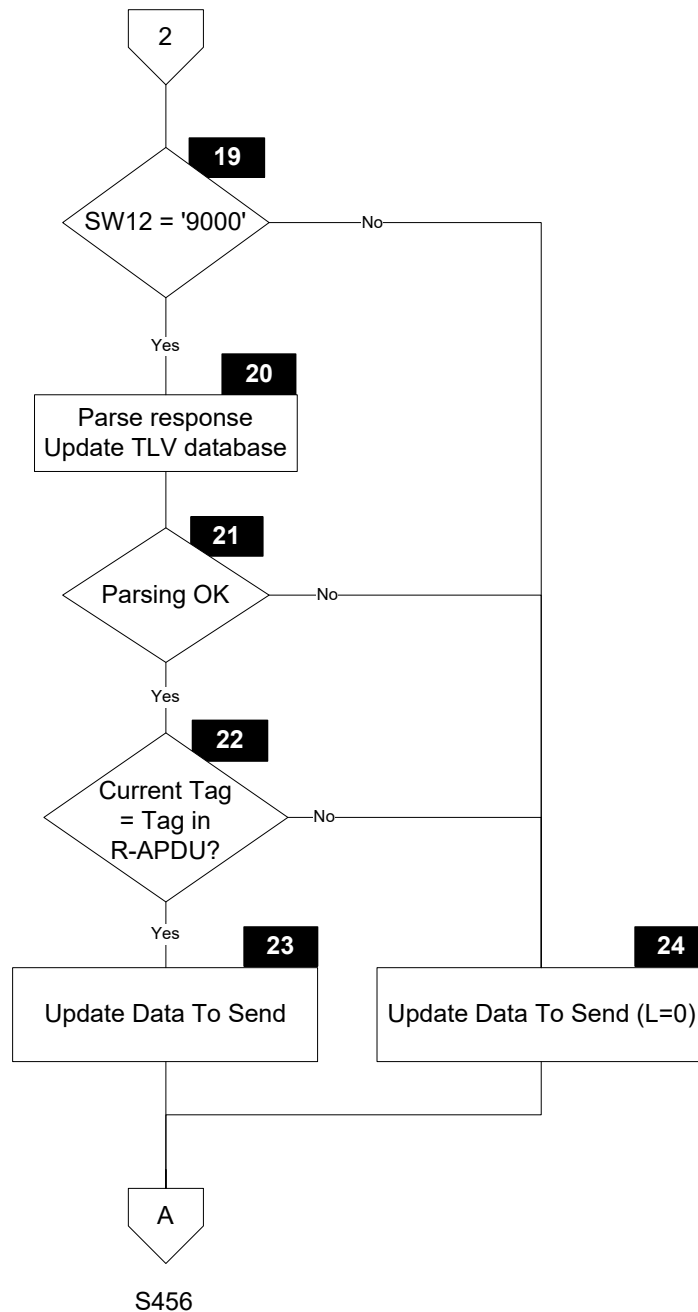
Figure 6.8 shows the flow diagram of s5 – waiting for get data response. Symbols in this diagram are labelled S5.X.

Figure 6.8—State 5 Flow Diagram





S5



6.9.3 Processing

S5.1

Receive DET Signal with Sync Data

S5.2

UpdateWithDetData(Sync Data)

S5.3

Receive RA Signal with Response Message Data Field and SW12

S5.4

Receive L1RSP Signal with Return Code

S5.5

'Message Identifier' in *User Interface Request Data* := TRY AGAIN

'Status' in *User Interface Request Data* := READY TO READ

'Hold Time' in *User Interface Request Data* := '000000'

S5.6

'Status' in *Outcome Parameter Set* := END APPLICATION

'Start' in *Outcome Parameter Set* := B

SET 'UI Request on Restart Present' in *Outcome Parameter Set*

'L1' in *Error Indication* := Return Code

'Msg On Error' in *Error Indication* := TRY AGAIN

CreateEMVDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*)))) Signal

S5.9

Current Tag := *Active Tag*

S5.10

Active Tag := GetNextGetDataTagFromList (*Tags To Read Yet*)

IF [*Active Tag* is not NULL]

THEN

 GOTO S5.11

ELSE

 GOTO S5.14

ENDIF

S5.11

Prepare GET DATA command for *Active Tag* as specified in section 5.5

S5.12

Send CA(GET DATA command) Signal

S5.13

'Next Cmd' in *Next Cmd* := GET DATA

S5.14

IF [Active AFL is empty]

THEN

GOTO S5.15

ELSE

GOTO S5.16

ENDIF

S5.15

'Next Cmd' in *Next Cmd* := NONE

S5.16

Prepare READ RECORD command for first record in *Active AFL* as specified in section 5.8

S5.17

Send CA(READ RECORD command) Signal

S5.18

'Next Cmd' in *Next Cmd* := READ RECORD

S5.19

IF [SW12 = '9000']

THEN

GOTO S5.20

ELSE

GOTO S5.24

ENDIF

S5.20

Parsing Result := ParseAndStoreCardResponse(Response Message Data Field)

Retrieve T, L and V from Response Message Data Field

Table 6.1—Response Message Data Field

T	L	V
---	---	---

S5.21

```
IF      [Parsing Result]
THEN
    GOTO S5.22
ELSE
    GOTO S5.24
ENDIF
```

S5.22

```
IF      [Current Tag = T]
THEN
    GOTO S5.23
ELSE
    GOTO S5.24
ENDIF
```

S5.23

AddToList(TLV, *Data To Send*)

S5.24

Add Current Tag with zero length to *Data To Send*:
AddToList(Current Tag || '00', *Data To Send*)

6.10 State 6 – Waiting for EMV Mode First Write Flag

State 6 is a state specific to Data Exchange processing and is only implemented for the DE/DS Implementation Option.

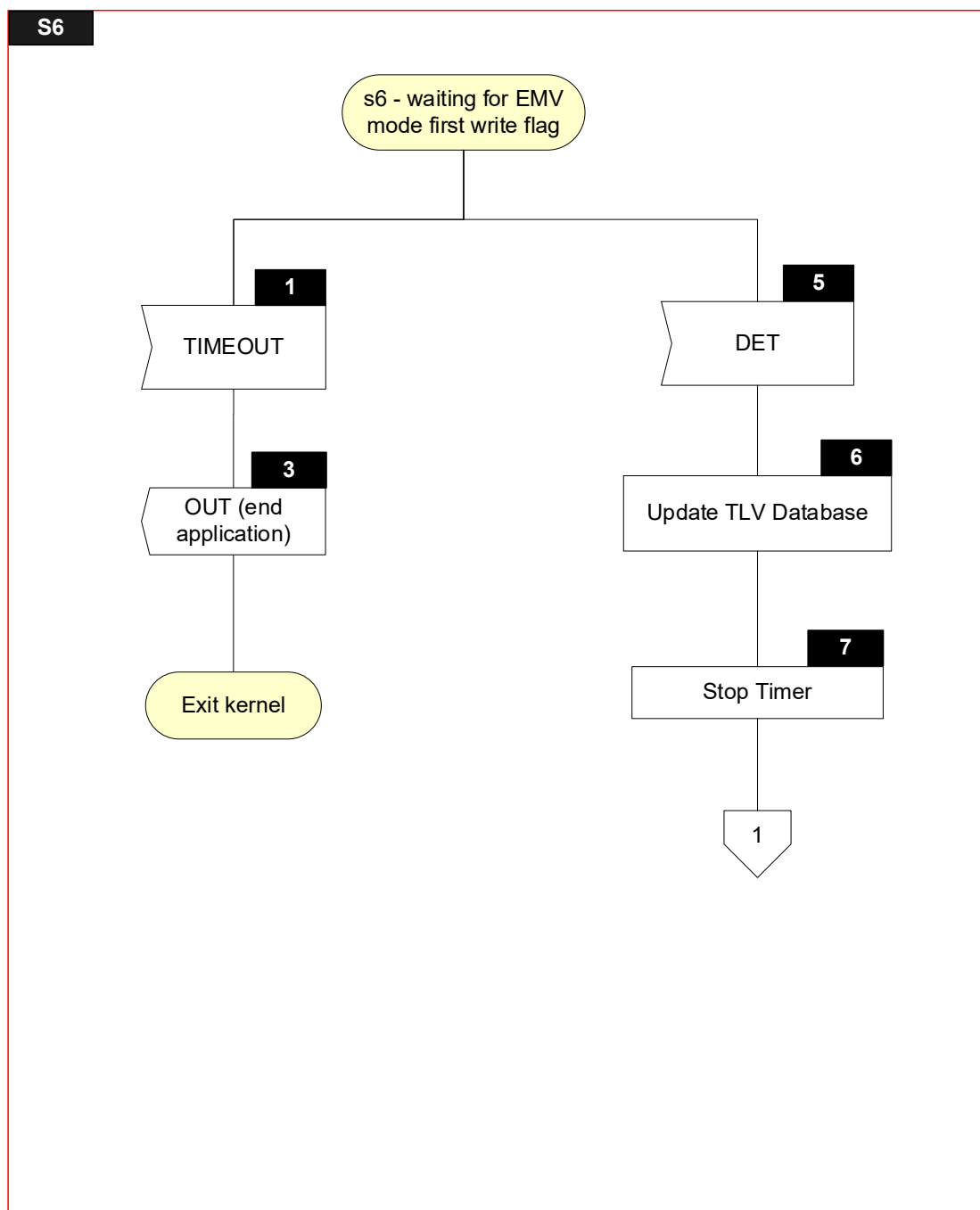
6.10.1 Local Variables

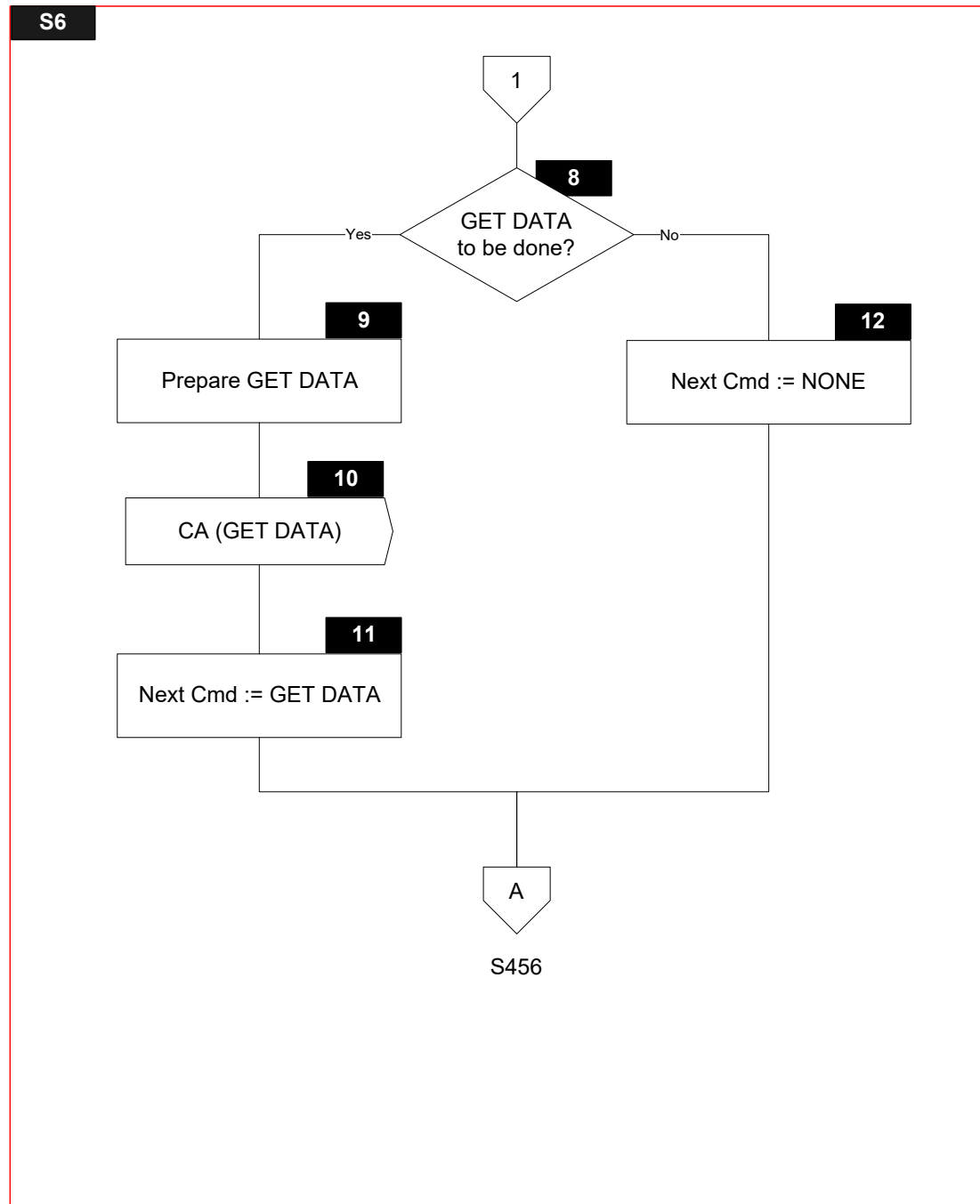
Name	Length	Type	Description
Sync Data	var.	b	List of data objects returned with DET Signal
T	var.	b	Tag of TLV encoded string
L	var.	b	Length of TLV encoded string
V	var. up to 252	b	Value of TLV encoded string

6.10.2 Flow Diagram

Figure 6.9 shows the flow diagram of `s6 – waiting for EMV mode first write flag`. Symbols in this diagram are labelled `S6.X`.

Figure 6.9—State 6 Flow Diagram





6.10.3 Processing

S6.1

Receive TIMEOUT Signal

S6.3

'Status' in *Outcome Parameter Set* := END APPLICATION

'L3' in *Error Indication* := TIME OUT

CreateEMVDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*))) Signal

S6.5

Receive DET Signal with Sync Data

S6.6

UpdateWithDetData(Sync Data)

S6.7

Stop Timer

S6.8

Active Tag := GetNextGetDataTagFromList (*Tags To Read Yet*)

IF [*Active Tag* is not NULL]

THEN

GOTO S6.9

ELSE

GOTO S6.12

ENDIF

S6.9

Prepare GET DATA command for *Active Tag* as specified in section 5.5

S6.10

Send CA(GET DATA command) Signal

S6.11

'Next Cmd' in *Next Cmd* := GET DATA

S6.12

'Next Cmd' in *Next Cmd* := NONE

6.11 States 4, 5, and 6 – Common Processing

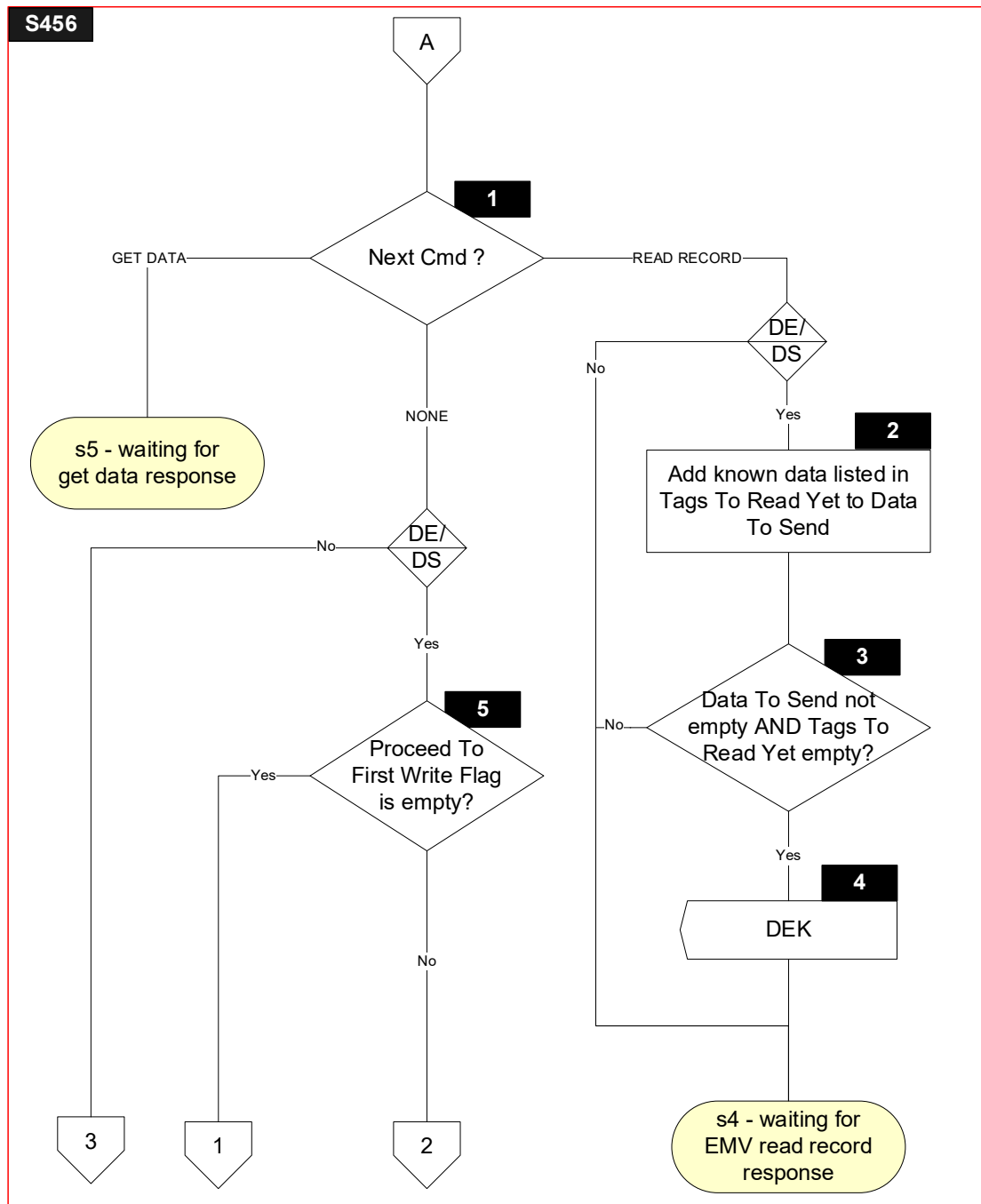
6.11.1 Local Variables

Local variables for common processing are defined in states 4, 5, and 6.

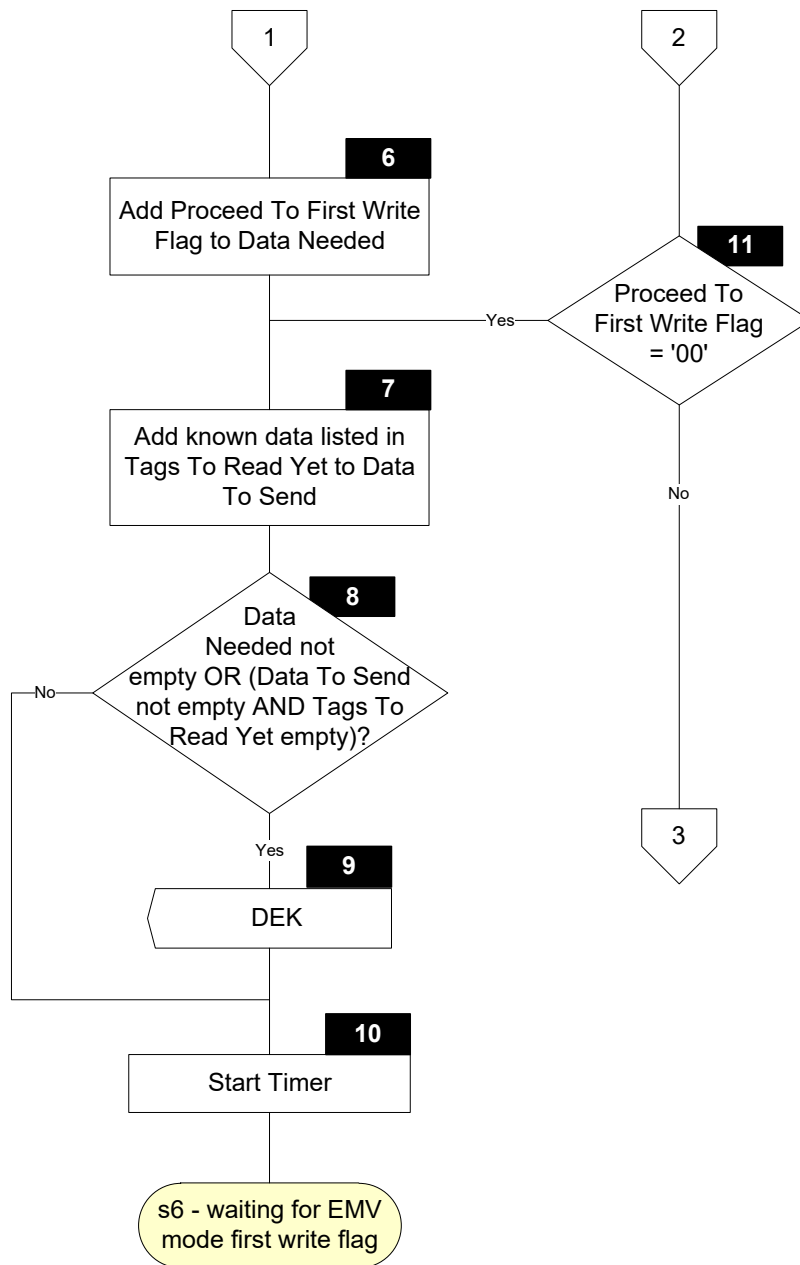
6.11.2 Flow Diagram

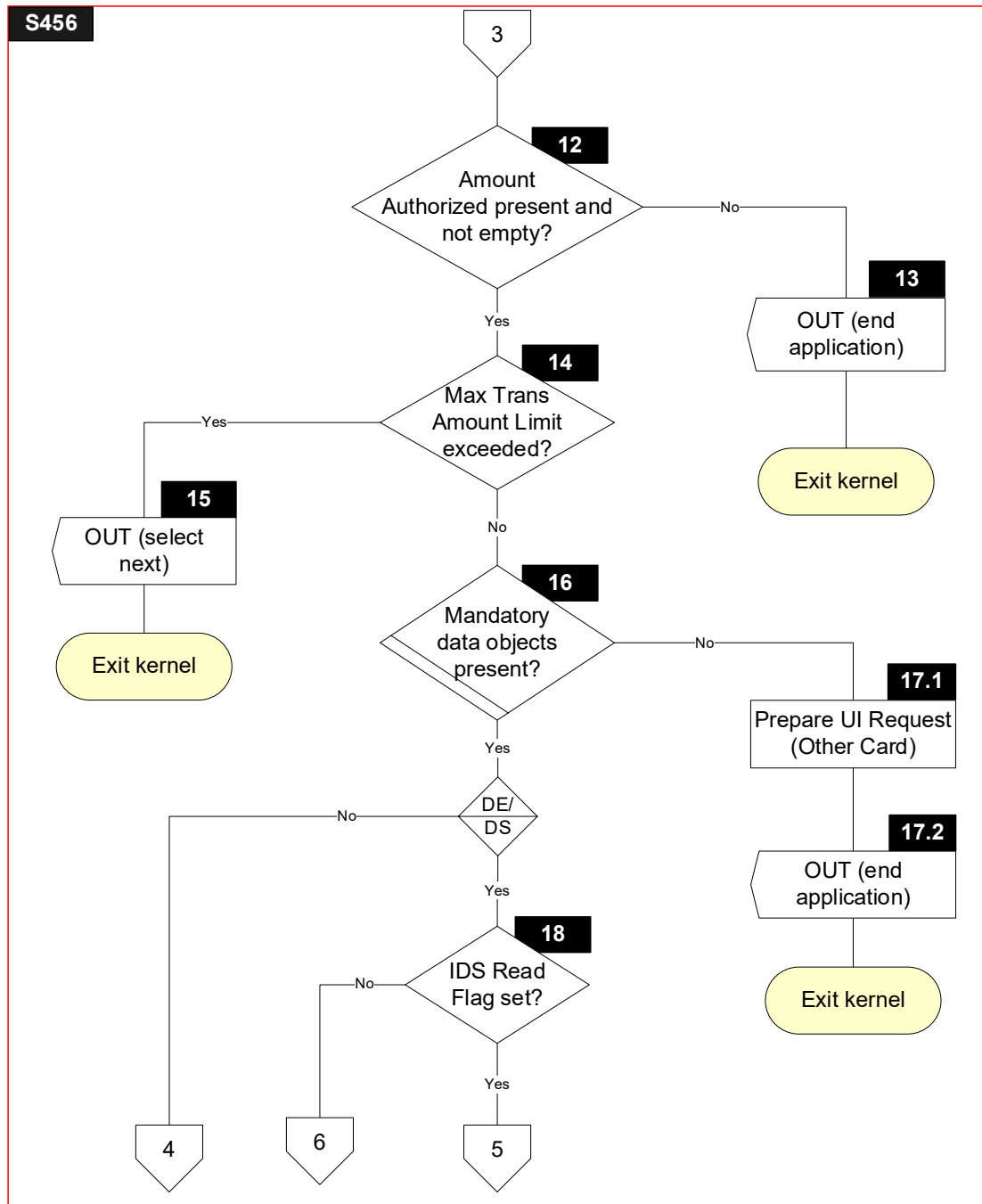
Figure 6.10 shows the flow diagram for common processing between states 4, 5, and 6. Symbols in this diagram are labelled S456.X.

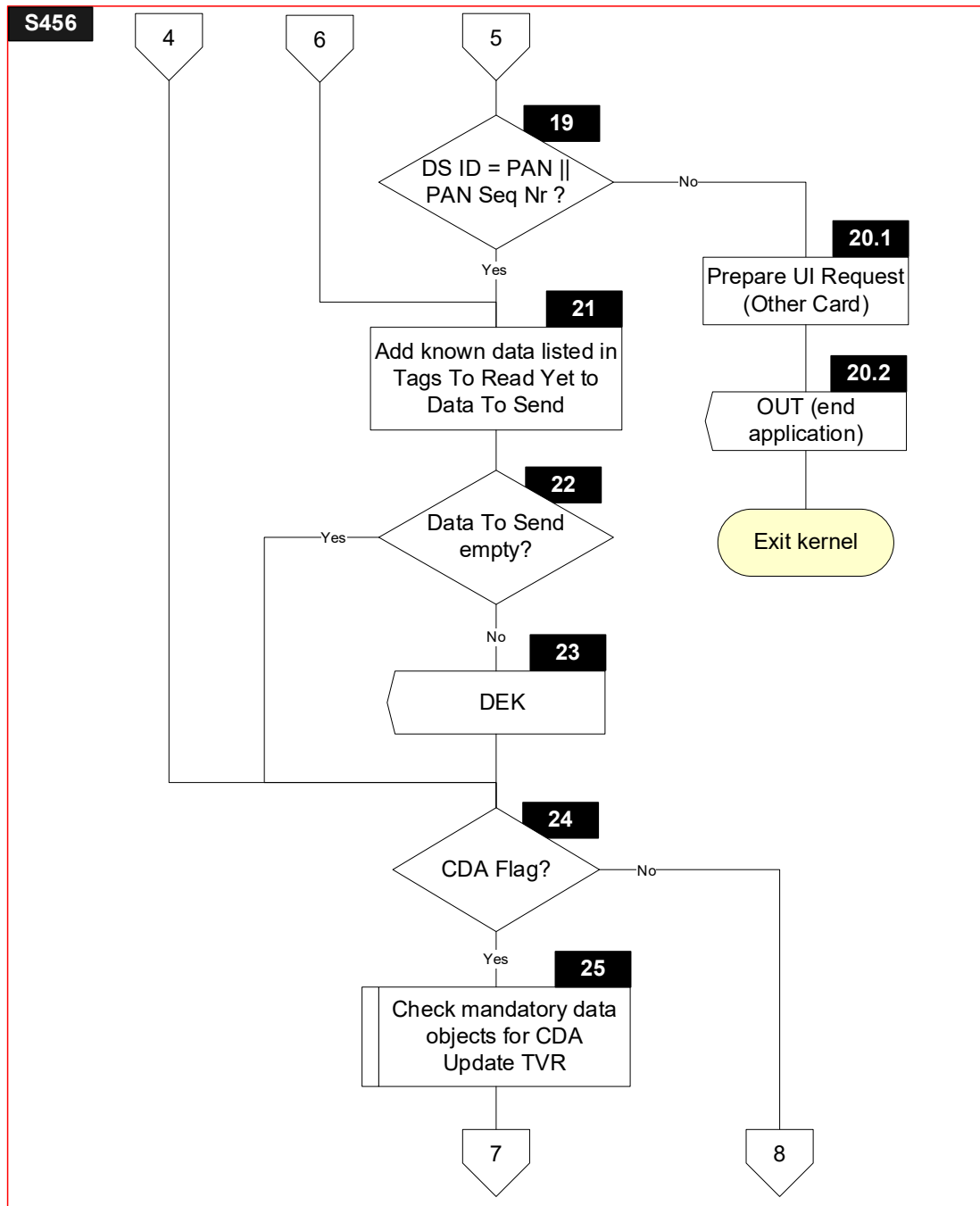
Figure 6.10—States 4, 5, and 6 – Common Processing – Flow Diagram

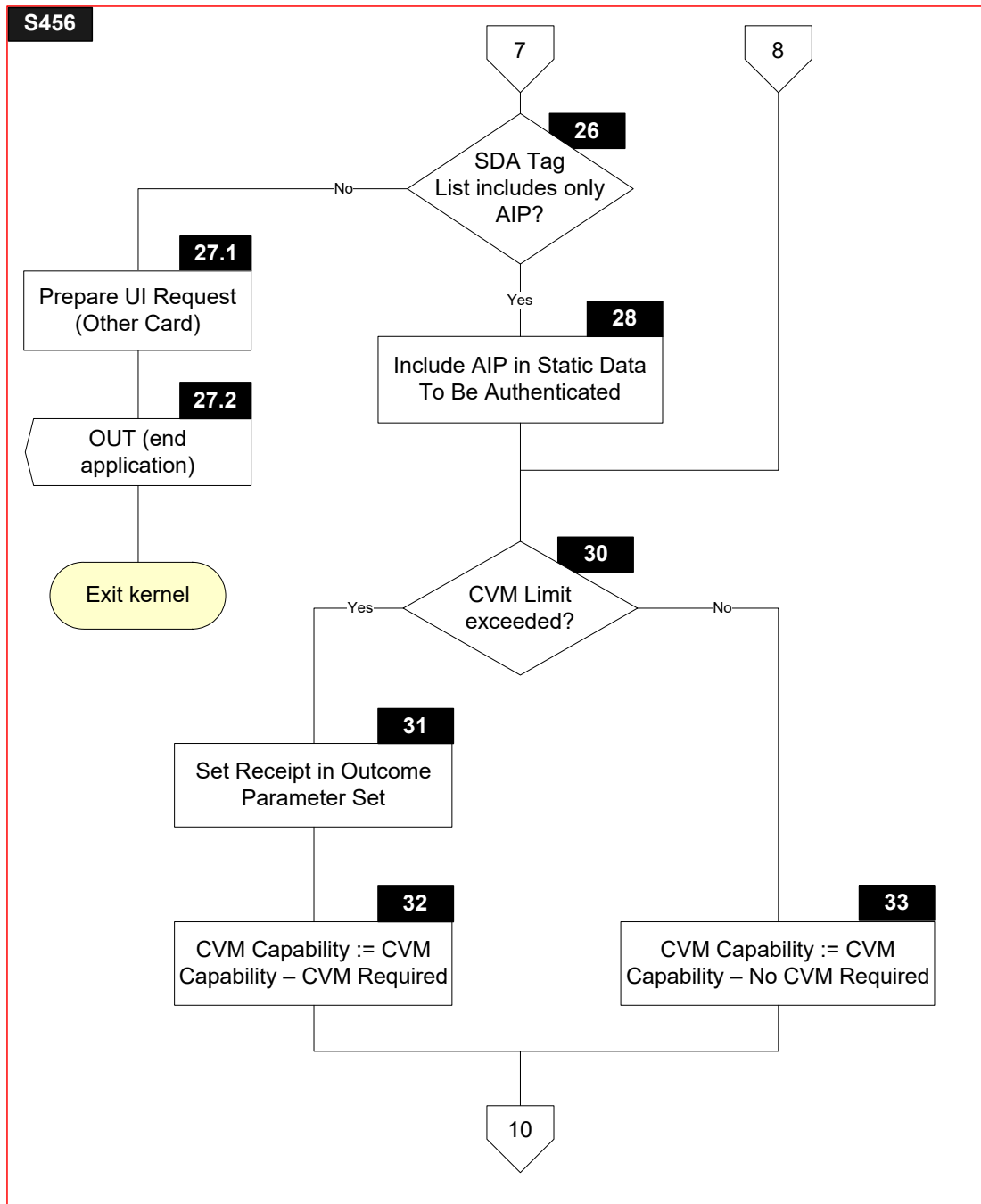


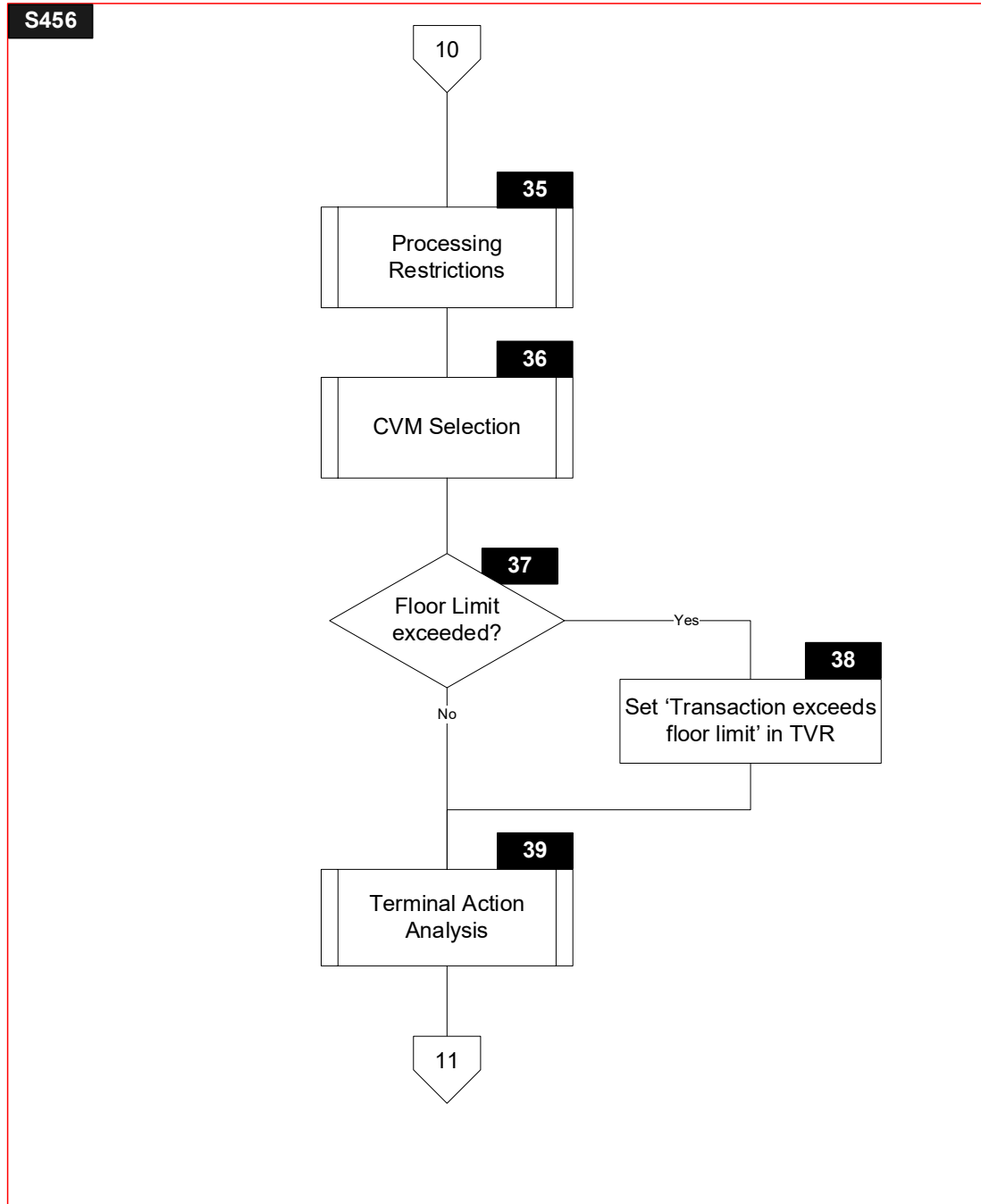
S456

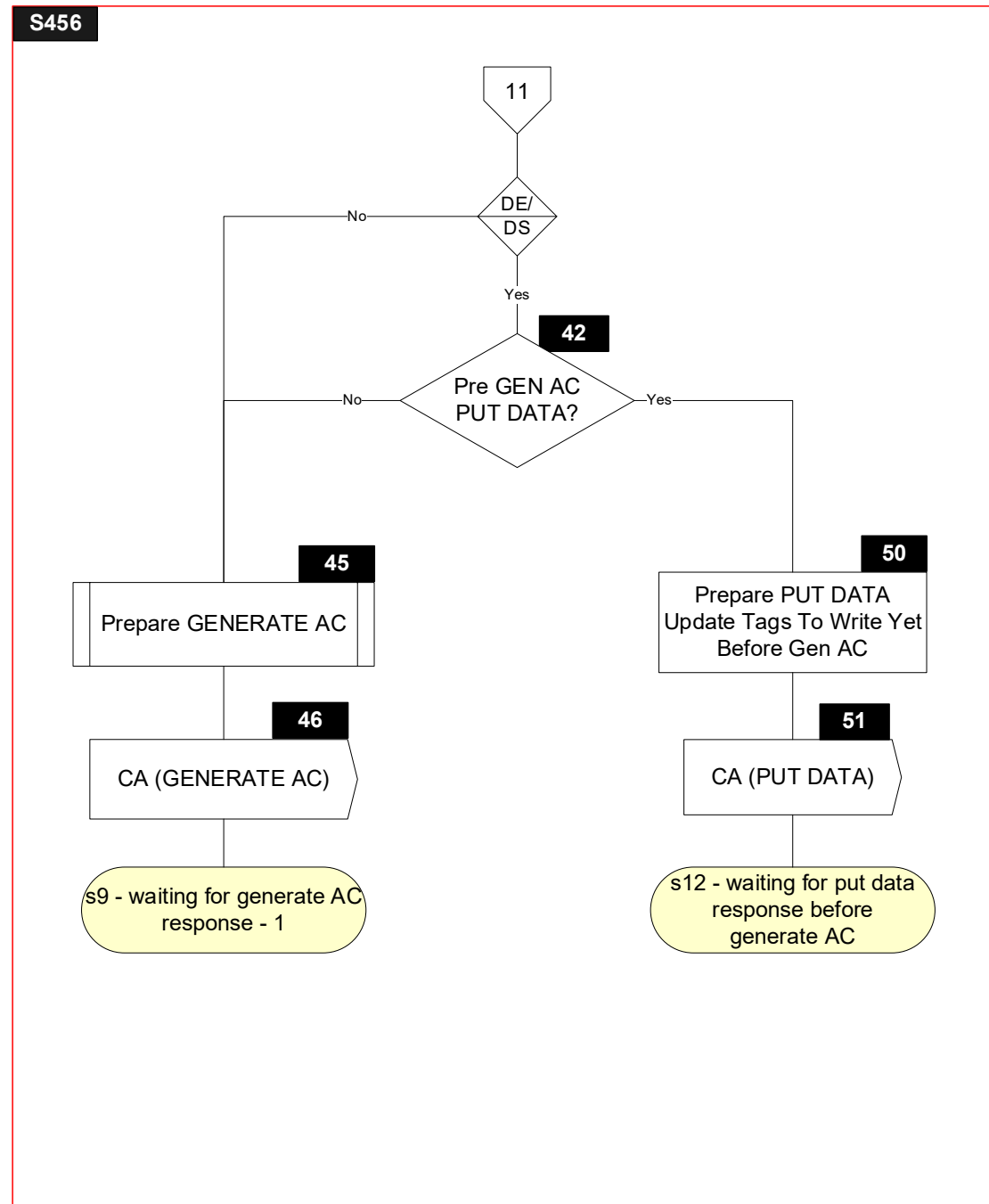












6.11.3 Processing

Note that symbols S456.2, S456.3, S456.4, S456.5, S456.6, S456.7, S456.8, S456.9, S456.10, S456.11, S456.18, S456.19, S456.20.1, S456.20.2, S456.21, S456.22, S456.23, S456.42, S456.50 and S456.51 are only implemented for the DE/DS Implementation Option.

S456.1

If DE/DS implemented:

IF ['Next Cmd' in *Next Cmd* = READ RECORD]

THEN

GOTO S456.2

ELSE

IF ['Next Cmd' in *Next Cmd* = GET DATA]

THEN

GOTO s5 - waiting for get data response

ELSE

GOTO S456.5

ENDIF

ENDIF

If DE/DS not implemented:

IF ['Next Cmd' in *Next Cmd* = READ RECORD]

THEN

GOTO s4 - waiting for read record response

ELSE

GOTO S456.12

ENDIF

S456.2

FOR every T in *Tags To Read Yet*

{

IF [IsEmpty(T)]

THEN

AddToList(GetTLV(T), *Data To Send*)

RemoveFromList(T, *Tags To Read Yet*)

ENDIF

}

S456.3

```
IF      [IsNotEmptyList(Data To Send) AND IsEmptyList(Tags To Read Yet)]
THEN
    GOTO S456.4
ELSE
    GOTO s4 - waiting for EMV read record response
ENDIF
```

S456.4

```
Send DEK(Data To Send, Data Needed) Signal
Initialize(Data To Send)
Initialize(Data Needed)
```

S456.5

```
IF      [IsEmpty(TagOf(Proceed To First Write Flag))]
THEN
    GOTO S456.6
ELSE
    GOTO S456.11
ENDIF
```

S456.6

```
AddToList (TagOf (Proceed To First Write Flag), Data Needed)
```

S456.7

```
FOR every T in Tags To Read Yet
{
    IF      [IsNotEmpty(T)]
    THEN
        AddToList(GetTLV(T), Data To Send)
        RemoveFromList(T, Tags To Read Yet)
    ENDIF
}
```

S456.8

```
IF      [IsNotEmptyList(Data Needed) OR
        (IsNotEmptyList(Data To Send) AND IsEmptyList(Tags To Read Yet))]
THEN
    GOTO S456.9
ELSE
    GOTO S456.10
ENDIF
```

S456.9

Send DEK(*Data To Send*, *Data Needed*) Signal

Initialize(*Data To Send*)

Initialize(*Data Needed*)

S456.10

Start Timer (*Time Out Value*)

S456.11

IF [IsPresent(TagOf(*Proceed To First Write Flag*)) AND
(*Proceed To First Write Flag* = '00')]

THEN

GOTO S456.7

ELSE

GOTO S456.12

ENDIF

S456.12

IF [IsEmpty(TagOf(*Amount*, *Authorized (Numeric)*))]

THEN

GOTO S456.14

ELSE

GOTO S456.13

ENDIF

S456.13

'Status' in *Outcome Parameter Set* := END APPLICATION

'L3' in *Error Indication* := AMOUNT NOT PRESENT

CreateEMVDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*))) Signal

S456.14

IF [*Amount*, *Authorized (Numeric)* > *Reader Contactless Transaction Limit*]

THEN

GOTO S456.15

ELSE

GOTO S456.16

ENDIF

S456.15

'Field Off Request' in *Outcome Parameter Set* := N/A
 'Status' in *Outcome Parameter Set* := SELECT NEXT
 'Start' in *Outcome Parameter Set* := C
 'L2' in *Error Indication* := MAX LIMIT EXCEEDED
 CreateEMVDiscretionaryData ()
 Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*))) Signal

S456.16

Check if all mandatory data objects are present in the TLV Database

Table 6.2—Mandatory EMV Mode Data Objects

Data Object
<i>Application Expiration Date</i>
<i>Application PAN</i>
<i>CDOL1</i>

IF [IsEmpty(TagOf(*Application Expiration Date*)) AND
 IsEmpty(TagOf(*Application PAN*)) AND
 IsEmpty(TagOf(*CDOL1*))]
 THEN
 GOTO S456.18 or S456.24
 ELSE
 GOTO S456.17.1
 ENDIF

S456.17.1

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD
 'Status' in *User Interface Request Data* := NOT READY

S456.17.2

'Status' in *Outcome Parameter Set* := END APPLICATION
 'Msg On Error' in *Error Indication* := ERROR – OTHER CARD
 'L2' in *Error Indication* := CARD DATA MISSING
 CreateEMVDiscretionaryData ()
 SET 'UI Request on Outcome Present' in *Outcome Parameter Set*
 Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*))) Signal

S456.18

```
IF      ['Read' in IDS Status is set]
THEN
    GOTO S456.19
ELSE
    GOTO S456.21
ENDIF
```

S456.19

Concatenate from left to right the *Application PAN* (without any 'F' padding) with the *Application PAN Sequence Number* (if the *Application PAN Sequence Number* is not present, then it is replaced by a '00' byte). The result, Y, must be padded to the left with a hexadecimal zero if necessary to ensure whole bytes. It must also be padded to the left with hexadecimal zeroes if necessary to ensure a minimum length of 8 bytes.

```
IF      [DS ID = Y]
THEN
    GOTO S456.21
ELSE
    GOTO S456.20.1
ENDIF
```

S456.20.1

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD
'Status' in *User Interface Request Data* := NOT READY

S456.20.2

'Status' in *Outcome Parameter Set* := END APPLICATION
'Msg On Error' in *Error Indication* := ERROR – OTHER CARD
'L2' in *Error Indication* := CARD DATA ERROR
CreateEMVDiscretionaryData ()
SET 'UI Request on Outcome Present' in *Outcome Parameter Set*
Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*)))) Signal

S456.21

FOR every T in *Tags To Read Yet*

```
{
    IF      [IsPresent(T)]
    THEN
        AddToList(GetTLV(T), Data To Send)
    ELSE
        IF [IsKnown(T)]
        THEN
            Add an empty data object with tag T to Data To Send if the
            TLV Database does not include a data object with tag T:
            AddToList(T || '00', Data To Send)
        ENDIF
    ENDIF
    RemoveFromList(T, Tags To Read Yet)
}
```

S456.22

```
IF      [IsEmptyList(Data To Send)]
THEN
    GOTO S456.24
ELSE
    GOTO S456.23
ENDIF
```

S456.23

Send DEK(*Data To Send*) Signal
Initialize(*Data To Send*)

S456.24

```
IF      ['CDA' in ODA Status is set]
THEN
    GOTO S456.25
ELSE
    GOTO S456.30
ENDIF
```

S456.25

Check if all mandatory Card data objects for CDA are present in the TLV Database

Table 6.3—Mandatory Card CDA Data Objects

Data Object
<i>CA Public Key Index (Card)</i>
<i>Issuer Public Key Certificate</i>
<i>Issuer Public Key Exponent</i>
<i>ICC Public Key Certificate</i>
<i>ICC Public Key Exponent</i>
<i>Static Data Authentication Tag List</i>

```

IF      [NOT (
        IsNotEmpty(TagOf(CA Public Key Index (Card))) AND
        IsNotEmpty(TagOf(Issuer Public Key Certificate)) AND
        IsNotEmpty(TagOf(Issuer Public Key Exponent)) AND
        IsNotEmpty(TagOf(ICC Public Key Certificate)) AND
        IsNotEmpty(TagOf(ICC Public Key Exponent)) AND
        IsNotEmpty(TagOf(Static Data Authentication Tag List))
        )]
THEN
    SET 'ICC data missing' in Terminal Verification Results
    SET 'CDA failed' in Terminal Verification Results
ENDIF
IF      [The CA Public Key Index (Card) is not present in the CA Public Key
        Database]
THEN
    SET 'CDA failed' in Terminal Verification Results
ENDIF

```

S456.26

```

IF      [IsNotEmpty(TagOf(Static Data Authentication Tag List)) AND
        (Static Data Authentication Tag List = '82')]
THEN
    GOTO S456.28
ELSE
    GOTO S456.27.1
ENDIF

```

S456.27.1

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

S456.27.2

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

'L2' in *Error Indication* := CARD DATA ERROR

CreateEMVDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)) ,
 GetTLV(TagOf(*User Interface Request Data*)))) Signal

S456.28

IF [Enough space left in *Static Data To Be Authenticated* to append *Application Interchange Profile*]

THEN

Append *Application Interchange Profile* at the end of *Static Data To Be Authenticated*

ELSE

SET 'CDA failed' in *Terminal Verification Results*

ENDIF

S456.30

IF [*Amount, Authorized (Numeric)* > *Reader CVM Required Limit*]

THEN

GOTO S456.31

ELSE

GOTO S456.33

ENDIF

S456.31

'Receipt' in *Outcome Parameter Set* := YES

S456.32

Terminal Capabilities[2] := *CVM Capability – CVM Required*

S456.33

Terminal Capabilities[2] := *CVM Capability – No CVM Required*

S456.35

Process Processing Restrictions as specified in section 7.3

S456.36

Process CVM Selection as specified in section 7.1

S456.37

```
IF      [Amount, Authorized (Numeric) > Reader Contactless Floor Limit]  
THEN  
    GOTO S456.38  
ELSE  
    GOTO S456.39  
ENDIF
```

S456.38

SET 'Transaction exceeds floor limit' in *Terminal Verification Results*

S456.39

Process Terminal Action Analysis as specified in section 7.4

S456.42

```
IF      [IsEmptyList(Tags To Write Yet Before Gen AC)]  
THEN  
    GOTO S456.50  
ELSE  
    GOTO S456.45  
ENDIF
```

S456.45

Prepare GENERATE AC command as specified in section 7.2

S456.46

Send CA(GENERATE AC command) Signal

S456.50

TLV := GetAndRemoveFromList(*Tags To Write Yet Before Gen AC*)

Prepare PUT DATA command for TLV as specified in section 5.7

S456.51

Send CA(PUT DATA command) Signal

6.12 State 7 – Waiting for Mag-stripe Read Record Response

State 7 is a state specific to mag-stripe mode and is only implemented for the MAG Implementation Option.

6.12.1 Local Variables

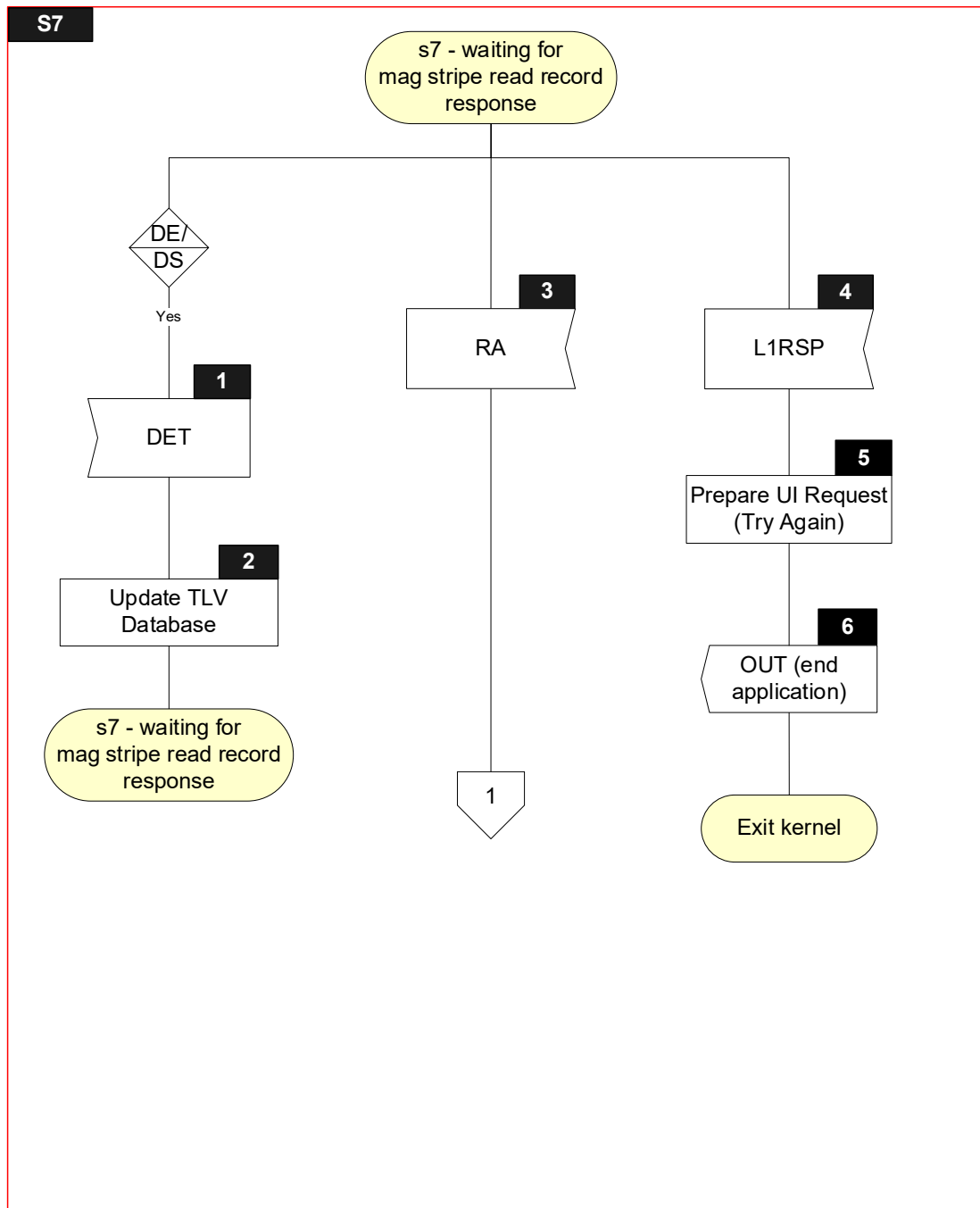
Name	Length	Format	Description
Return Code	1	b	Value returned with L1RSP Signal (TIME OUT ERROR, PROTOCOL ERROR, TRANSMISSION ERROR)
Sync Data ⁽¹⁾	var.	b	List of data objects returned with DET Signal
Parsing Result	1	b	Boolean used to store result of parsing a TLV string
SW12	2	b	Status bytes
Record	var. up to 256	b	Response Message Data Field of the R-APDU of READ RECORD
T	var.	b	Tag of TLV encoded string
L	var.	b	Length of TLV encoded string
V	var. up to 253	b	Value of TLV encoded string

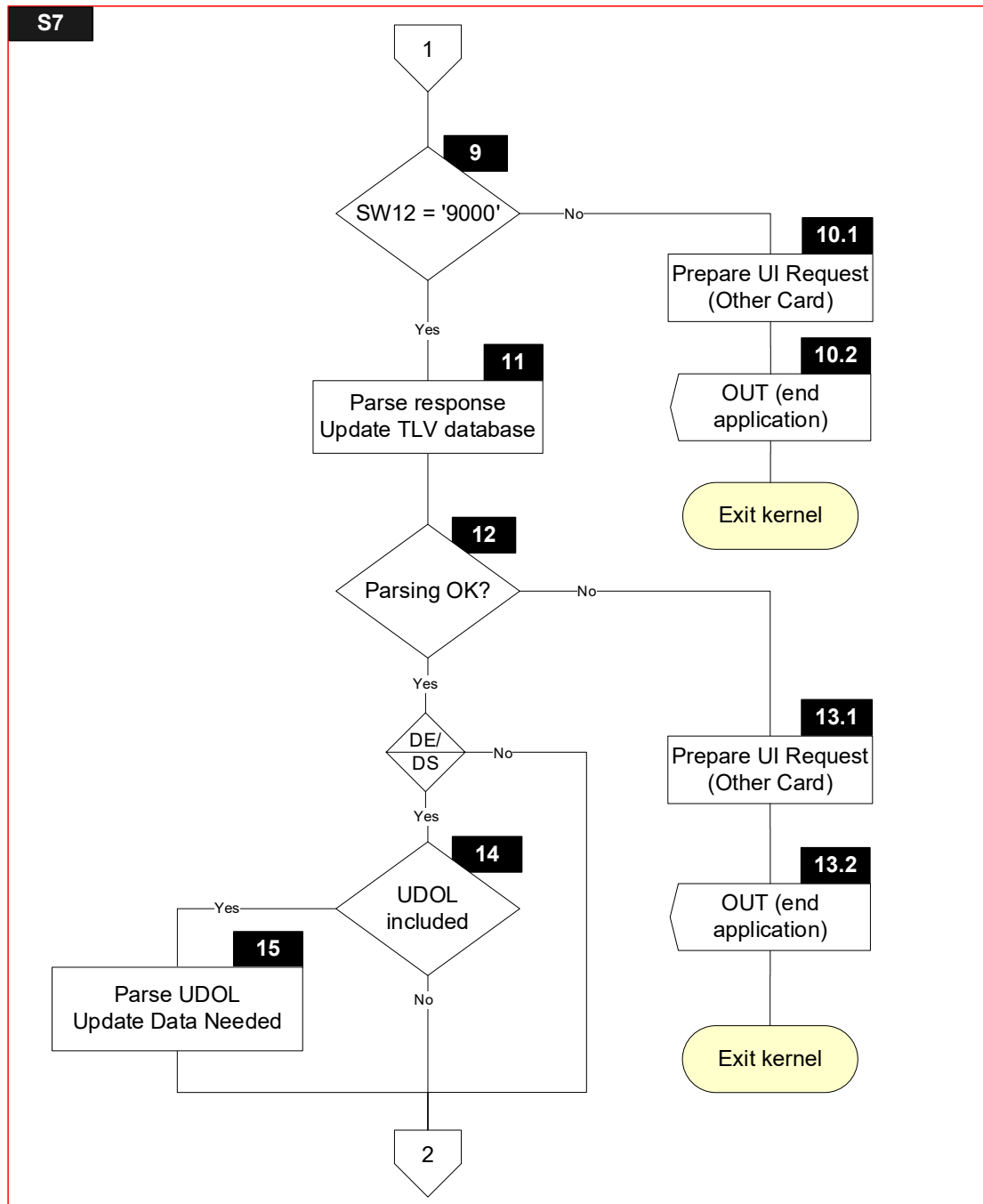
⁽¹⁾ Only for the DE/DS Implementation Option

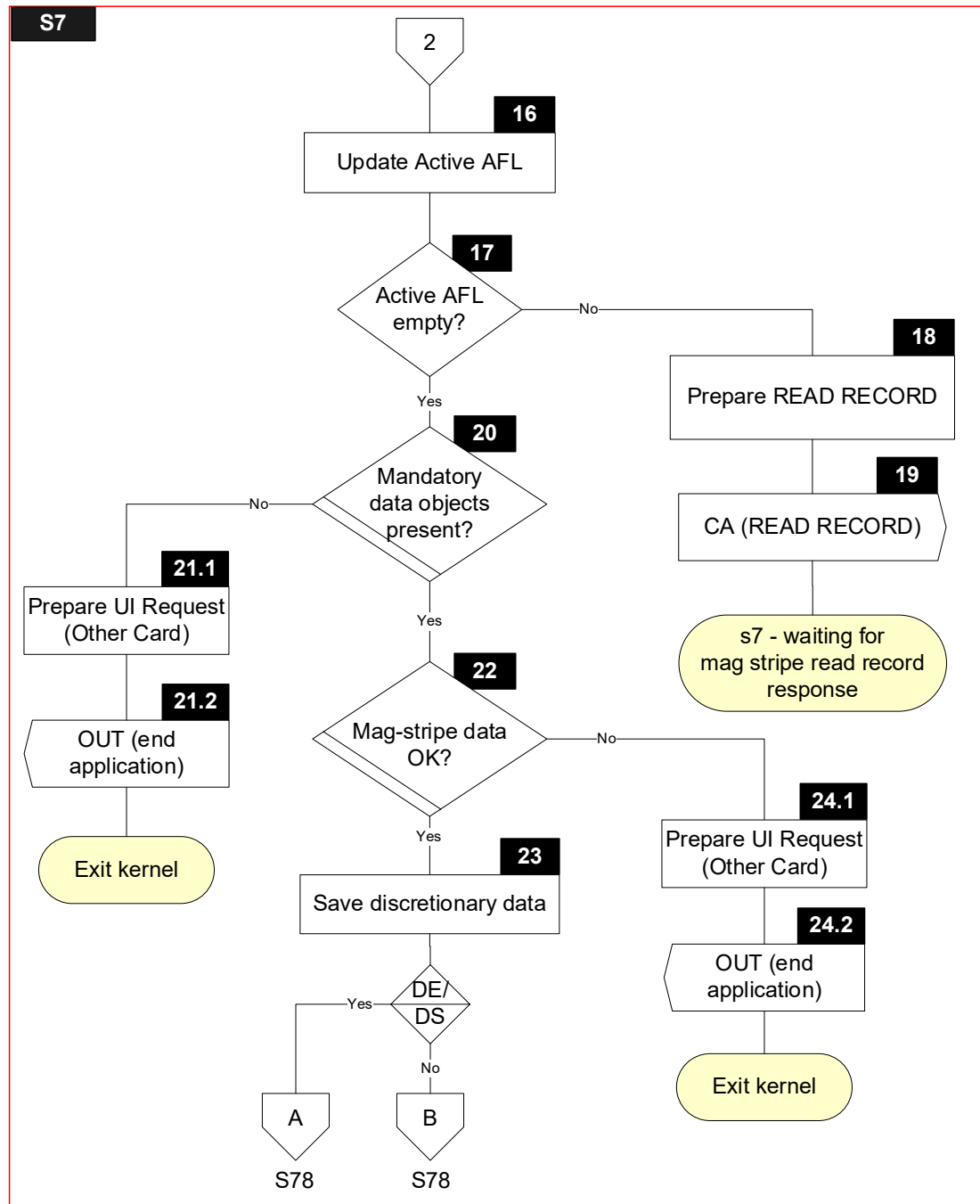
6.12.2 Flow Diagram

Figure 6.11 shows the flow diagram of s7 – waiting for mag stripe read record response. Symbols in this diagram are labelled S7.X.

Figure 6.11—State 7 Flow Diagram







6.12.3 Processing

Note that symbols S7.1, S7.2, S7.14 and S7.15 are only implemented for the DE/DS Implementation Option.

S7.1

Receive DET Signal with Sync Data

S7.2

UpdateWithDetData(Sync Data)

S7.3

Receive RA Signal with Record and SW12

S7.4

Receive L1RSP Signal with Return Code

S7.5

'Message Identifier' in *User Interface Request Data* := TRY AGAIN

'Status' in *User Interface Request Data* := READY TO READ

'Hold Time' in *User Interface Request Data* := '000000'

S7.6

'Status' in *Outcome Parameter Set* := END APPLICATION

'Start' in *Outcome Parameter Set* := B

SET 'UI Request on Restart Present' in *Outcome Parameter Set*

'L1' in *Error Indication* := Return Code

'Msg On Error' in *Error Indication* := TRY AGAIN

CreateMSDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*)))) Signal

S7.9

IF [SW12 = '9000']

THEN

GOTO S7.11

ELSE

GOTO S7.10.1

ENDIF

S7.10.1

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

S7.10.2

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

'L2' in *Error Indication* := STATUS BYTES

'SW12' in *Error Indication* := SW12

CreateMSDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*))) Signal

S7.11

IF [SFI of file of Record \leq 10]

THEN

IF [(Length of Record > 0) AND (Record[1] = '70')]

THEN

Parsing Result := ParseAndStoreCardResponse(Record)

ELSE

Parsing Result := FALSE

ENDIF

ELSE

Processing of records in proprietary files is beyond the scope of this specification

ENDIF

S7.12

IF [Parsing Result]

THEN

GOTO S7.14 or S7.16

ELSE

GOTO S7.13.1

ENDIF

S7.13.1

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

S7.13.2

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

'L2' in *Error Indication* := PARSING ERROR

CreateMSDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*)))) Signal

S7.14

IF [Record includes *UDOL*]

THEN

GOTO S7.15

ELSE

GOTO S7.16

ENDIF

S7.15

FOR every TL entry in the *UDOL*

{

IF [IsEmpty(T)]

THEN

AddToList(T, *Data Needed*)

ENDIF

}

S7.16

Remove first record from *Active AFL*

S7.17

IF [*Active AFL* is empty]

THEN

GOTO S7.20

ELSE

GOTO S7.18

ENDIF

S7.18

Prepare READ RECORD command for first record in *Active AFL* as specified in section 5.8

S7.19

Send CA(READ RECORD command) Signal

S7.20

Check if all mandatory data objects are present in the TLV Database

Table 6.4—Mandatory Mag-stripe Mode Data Objects

Data Object
<i>Track 2 Data</i>
<i>PUNATC(Track2)</i>
<i>PCVC3(Track2)</i>
<i>NATC(Track2)</i>

IF [IsNotEmpty(TagOf(*Track 2 Data*)) AND
IsNotEmpty(TagOf(*PUNATC(Track2)*)) AND
IsNotEmpty(TagOf(*PCVC3(Track2)*)) AND
IsNotEmpty(TagOf(*NATC(Track2)*))]

THEN

GOTO S7.22

ELSE

GOTO S7.21.1

ENDIF

S7.21.1

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

S7.21.2

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

'L2' in *Error Indication* := CARD DATA MISSING

CreateMSDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*))) Signal

S7.22

Verify correctness of the mag-stripe mode data objects as follows:

$nUN := (\text{Number of non-zero bits in } PUNATC(Track2)) - NATC(Track2)$

IF $[(nUN < 0) \text{ OR } (nUN > 8)]$

THEN

GOTO S7.24.1

ENDIF

IF $[\text{IsNotEmpty}(\text{TagOf}(Track\ 1\ Data))]$

THEN

IF $[(\text{IsNotPresent}(\text{TagOf}(NATC(Track1))) \text{ OR } \text{IsEmpty}(\text{TagOf}(NATC(Track1))))$

OR

$(\text{IsNotPresent}(\text{TagOf}(PCVC3(Track1))) \text{ OR } \text{IsEmpty}(\text{TagOf}(PCVC3(Track1))))$

OR

$(\text{IsNotPresent}(\text{TagOf}(PUNATC(Track1))) \text{ OR } \text{IsEmpty}(\text{TagOf}(PUNATC(Track1))))$

OR

$(\text{Number of non-zero bits in } PUNATC(Track1) - NATC(Track1) \neq nUN)]$

THEN

GOTO S7.24.1

ELSE

GOTO S7.23

ENDIF

ELSE

GOTO S7.23

ENDIF

Note that the Kernel must not validate the individual data fields in *Track 1 Data* and *Track 2 Data*. Specifically:

- Validation of the values 2 and 6 in the first digit of the service code present in *Track 1 Data* or *Track 2 Data* to determine if a contact chip transaction is required must not be performed.
- Validation of the cardholder name, including the presence of the surname separator, must not be performed.

Any existing data validation carried out to support individual payment products is outside the scope of this specification.

However, if the Kernel is not able to localize the discretionary part of *Track 1 Data* or *Track 2 Data* due to one or more format errors, the Kernel must terminate the transaction as described in S7.24.1

S7.23

DD Card (Track2) := 'Discretionary Data' in *Track 2 Data*

IF [IsEmpty(TagOf(*Track 1 Data*))]

THEN

DD Card (Track1) := 'Discretionary Data' in *Track 1 Data*

ENDIF

S7.24.1

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

S7.24.2

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

'L2' in *Error Indication* := CARD DATA ERROR

CreateMSDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*))) Signal

6.13 State 8 – Waiting for Mag-stripe First Write Flag

State 8 is a state specific to Data Exchange processing and Mag-stripe Mode and is only implemented if both the MAG and DE/DS Implementation Options are implemented.

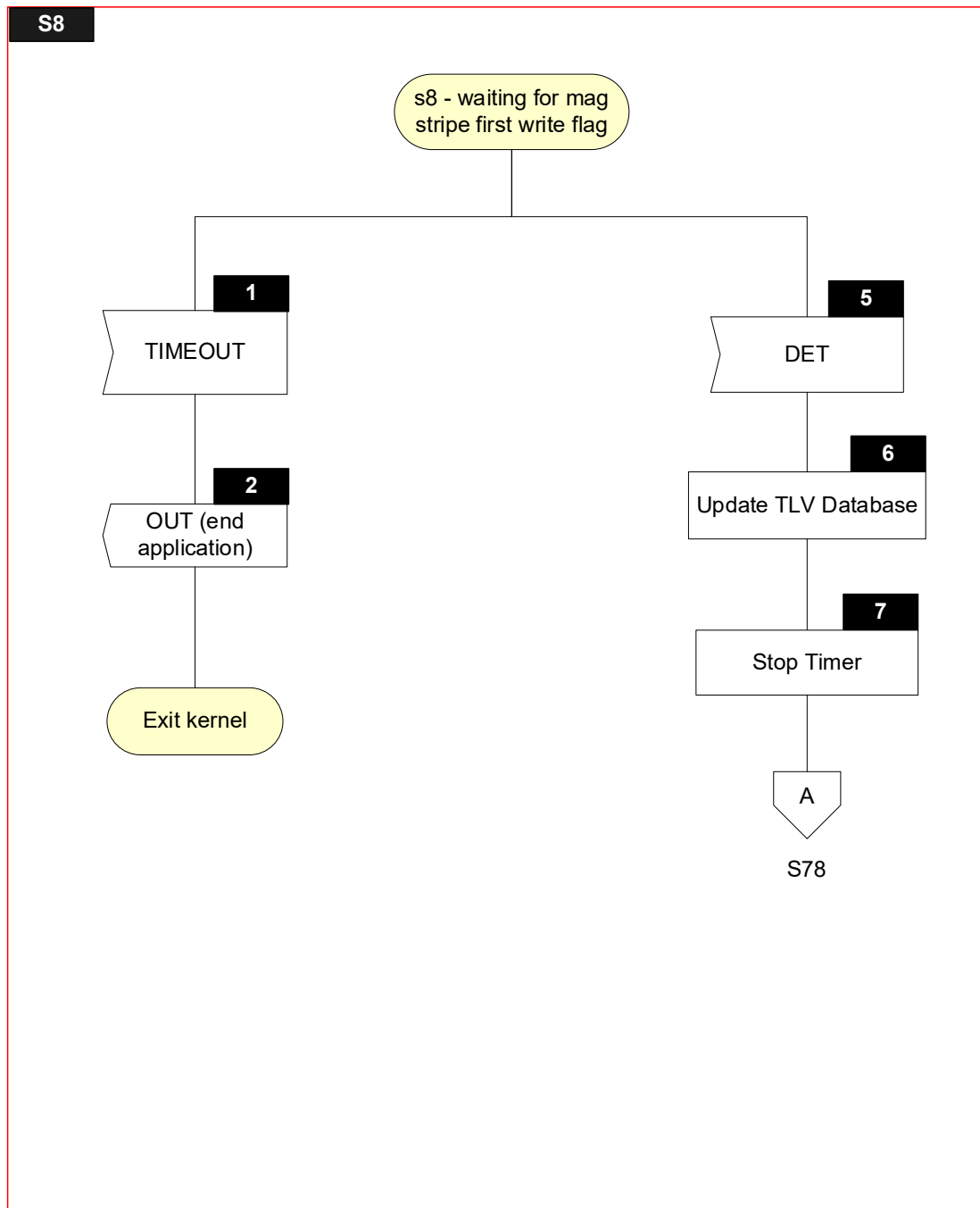
6.13.1 Local Variables

Name	Length	Format	Description
Sync Data ⁽	var.	b	List of data objects returned with DET Signal
T	var.	b	Tag of TLV encoded string
L	var.	b	Length of TLV encoded string
V	var. up to 253	b	Value of TLV encoded string

6.13.2 Flow Diagram

Figure 6.12 shows the flow diagram of s8 – waiting for mag stripe first write flag. Symbols in this diagram are labelled S8.X.

Figure 6.12—State 8 Flow Diagram



6.13.3 Processing

S8.1

Receive TIMEOUT Signal

S8.2

'Status' in *Outcome Parameter Set* := END APPLICATION

'L3' in *Error Indication* := TIME OUT

CreateMSDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*))) Signal

S8.5

Receive DET Signal with Sync Data

S8.6

UpdateWithDetData(Sync Data)

S8.7

Stop Timer

6.14 States 7 and 8 – Common Processing

State 7 and 8 are states specific to Mag-stripe Mode and are only implemented for the MAG Implementation Option.

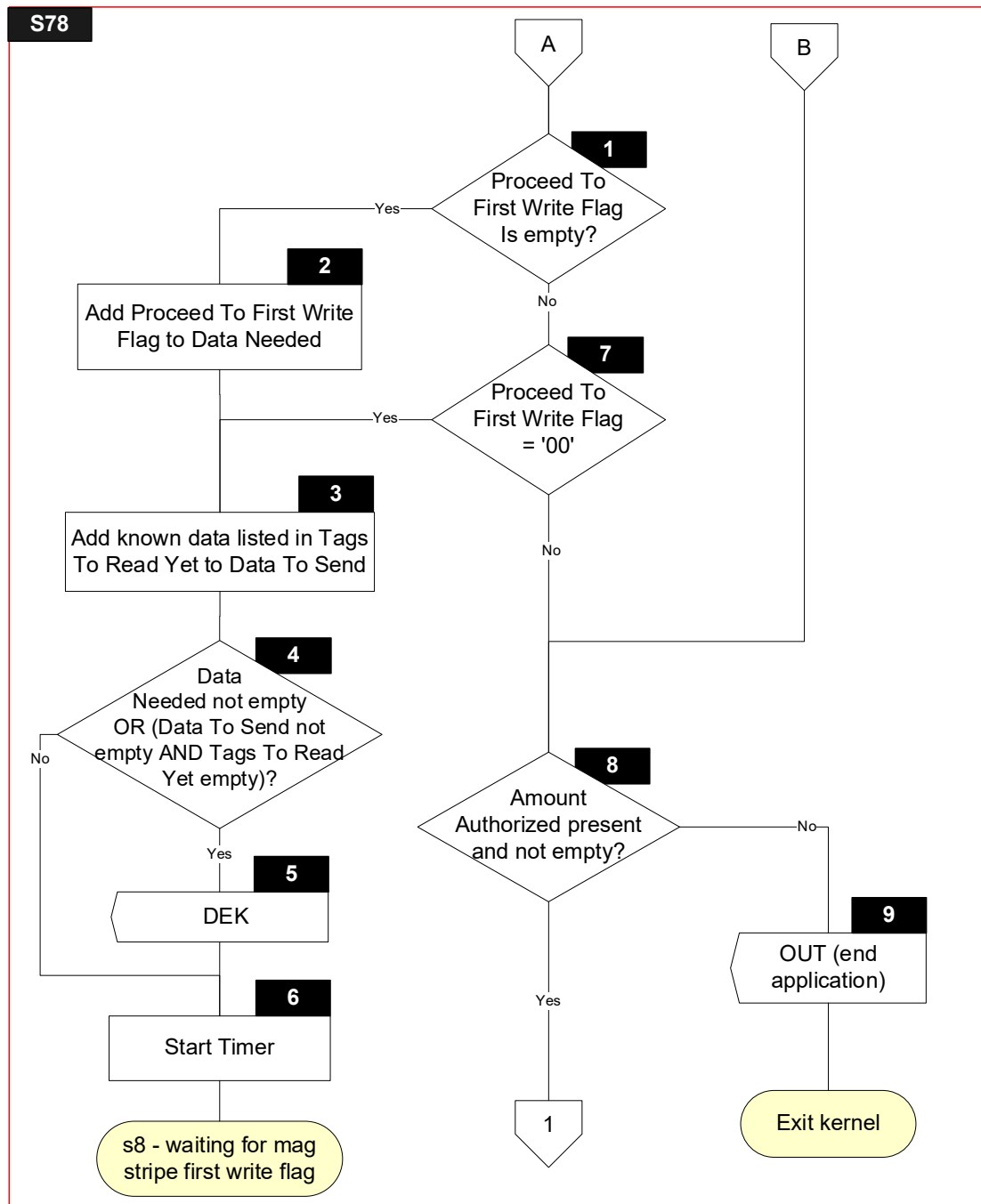
6.14.1 Local Variables

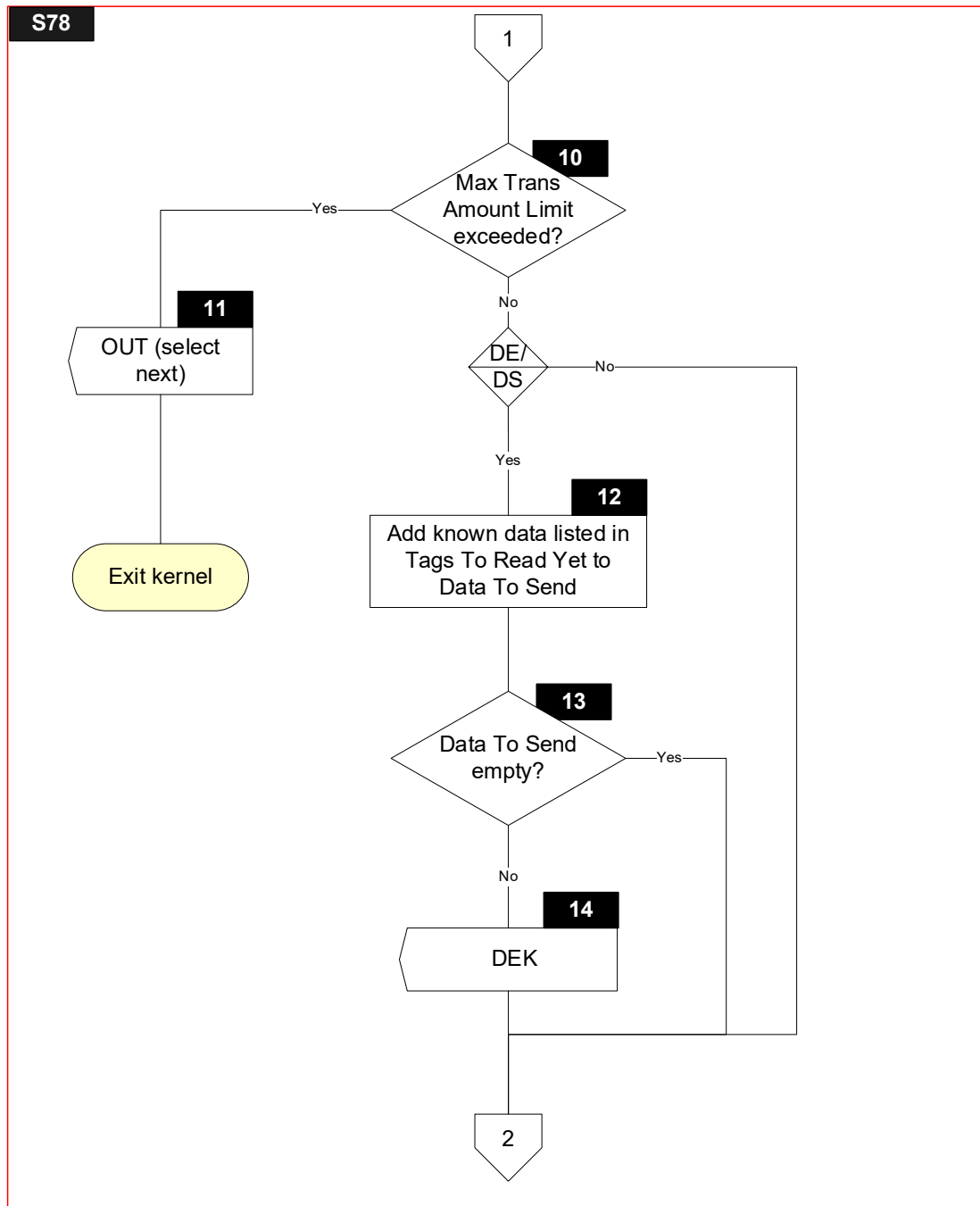
Local variables for common processing are defined in states 7 and 8.

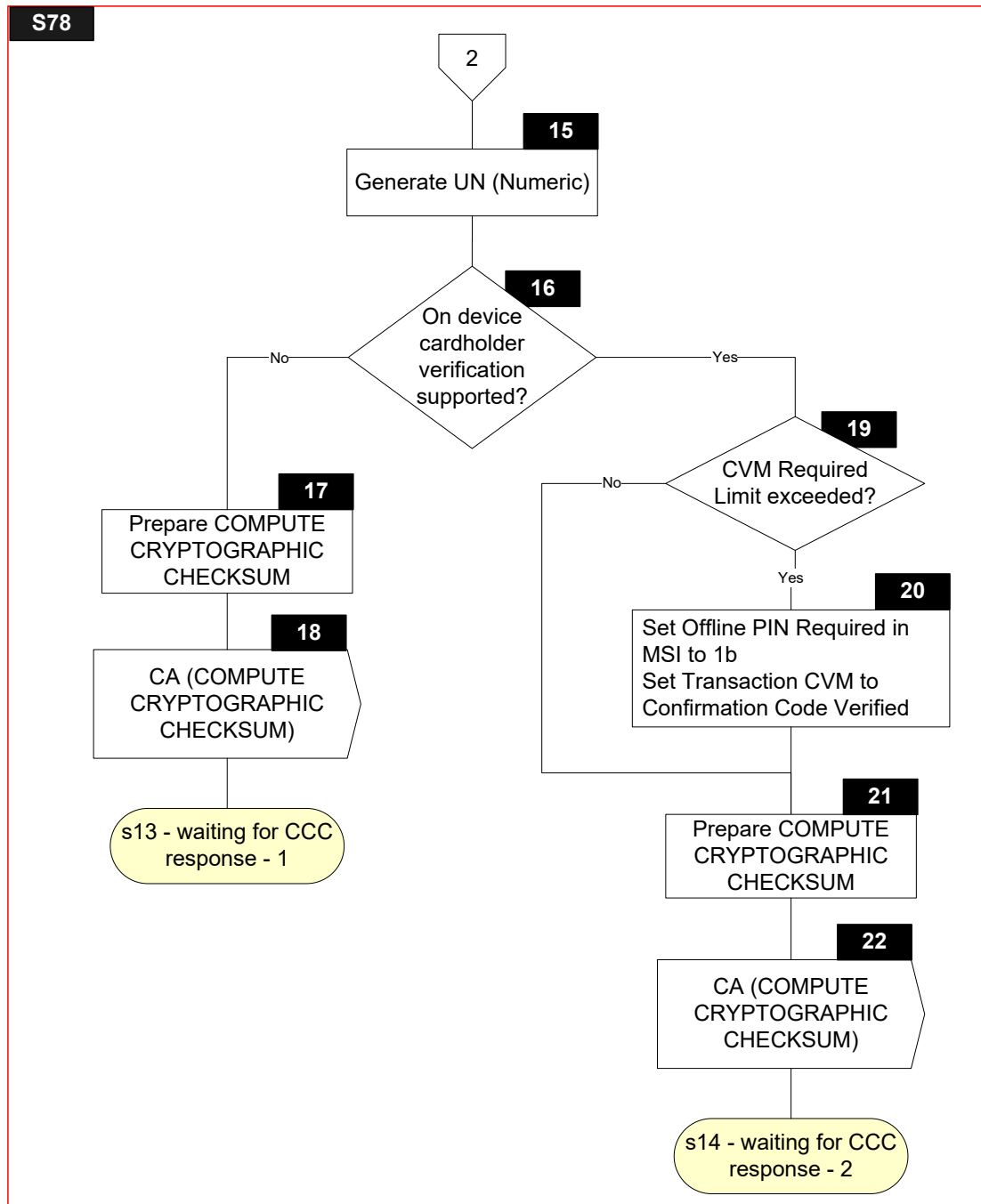
6.14.2 Flow Diagram

Figure 6.13 shows the flow diagram for common processing between states 7 and 8. Symbols in this diagram are labelled S78.X.

Figure 6.13—States 7 and 8 – Common Processing – Flow Diagram







6.14.3 Processing

Note that symbols S78.1, S78.2, S78.3, S78.4, S78.5, S78.6, S78.7, S78.12, S78.13 and S78.14 are only implemented for the DE/DS Implementation Option.

S78.1

```
IF      [IsEmpty(TagOf(Proceed To First Write Flag))]
THEN
    GOTO S78.2
ELSE
    GOTO S78.7
ENDIF
```

S78.2

AddToList(TagOf(*Proceed To First Write Flag*), *Data Needed*)

S78.3

```
FOR every T in Tags To Read Yet
{
    IF      [IsNotEmpty(T)]
    THEN
        AddToList(GetTLV(T), Data To Send)
        RemoveFromList(T, Tags To Read Yet)
    ENDIF
}
```

S78.4

```
IF      [IsNotEmptyList(Data Needed) OR
        (IsNotEmptyList(Data To Send) AND IsEmptyList(Tags To Read Yet))]
THEN
    GOTO S78.5
ELSE
    GOTO S78.6
ENDIF
```

S78.5

Send DEK(*Data To Send*, *Data Needed*) Signal
Initialize *Data To Send*
Initialize *Data Needed*

S78.6

Start Timer (*Time Out Value*)

S78.7

```
IF      [IsPresent(TagOf(Proceed To First Write Flag)) AND
        (Proceed To First Write Flag = '00')]
THEN
    GOTO S78.3
ELSE
    GOTO S78.8
ENDIF
```

S78.8

```
IF      [IsNotEmpty(TagOf(Amount, Authorized (Numeric)))]
THEN
    GOTO S78.10
ELSE
    GOTO S78.9
ENDIF
```

S78.9

```
'Status' in Outcome Parameter Set := END APPLICATION
'L3' in Error Indication := AMOUNT NOT PRESENT
CreateMSDiscretionaryData ()
Send OUT(GetTLV(TagOf(Outcome Parameter Set)),
        GetTLV(TagOf(Discretionary Data))) Signal
```

S78.10

```
IF      [Amount, Authorized (Numeric) > Reader Contactless Transaction Limit]
THEN
    GOTO S78.11
ELSE
    GOTO S78.12 or S78.15
ENDIF
```

S78.11

```
'Field Off Request' in Outcome Parameter Set := N/A
'Status' in Outcome Parameter Set := SELECT NEXT
'Start' in Outcome Parameter Set := C
'L2' in Error Indication := MAX LIMIT EXCEEDED
CreateMSDiscretionaryData ()
Send OUT(GetTLV(TagOf(Outcome Parameter Set)),
        GetTLV(TagOf(Discretionary Data))) Signal
```

S78.12

FOR every T in *Tags To Read Yet*

```
{
    IF      [IsPresent(T)]
    THEN
        AddToList(GetTLV(T), Data To Send)
    ELSE
        IF [IsKnown(T)]
        THEN
            Add an empty data object with tag T to Data To Send if the
            TLV Database does not include a data object with tag T:
            AddToList(T || '00', Data To Send)
        ENDIF
    ENDIF
    RemoveFromList(T, Tags To Read Yet)
}
```

S78.13

```
IF      [IsEmptyList(Data To Send)]
THEN
    GOTO S78.15
ELSE
    GOTO S78.14
ENDIF
```

S78.14

Send DEK(*Data To Send*) Signal
Initialize(*Data To Send*)

S78.15

Generate a 4 byte random value as described in section 8.1. Convert the random value to a 4 byte BCD encoded value and set the 8 – *nUN* most significant digits to zero.

Store this value in *Unpredictable Number (Numeric)*.

Note that it is possible to generate the value of the *Unpredictable Number (Numeric)* at other times in parallel with the processing of a CA Signal without changing the external behaviour of the Kernel. The *Unpredictable Number (Numeric)* could for example be generated after S3.81.

S78.16

IF ['On device cardholder verification is supported' in *Application Interchange Profile* is set AND
 'On device cardholder verification supported' in *Kernel Configuration* is set]
THEN
 GOTO S78.19
ELSE
 GOTO S78.17
ENDIF

S78.17

Prepare COMPUTE CRYPTOGRAPHIC CHECKSUM command as specified in section 5.2.2

S78.18

Send CA(COMPUTE CRYPTOGRAPHIC CHECKSUM) Signal

S78.19

IF [*Amount, Authorized (Numeric)* > *Reader CVM Required Limit*]
THEN
 GOTO S78.20
ELSE
 GOTO S78.21
ENDIF

S78.20

SET 'OD-CVM Required' in *Mobile Support Indicator*
'CVM' in *Outcome Parameter Set* := CONFIRMATION CODE VERIFIED

S78.21

Prepare COMPUTE CRYPTOGRAPHIC CHECKSUM command as specified in section 5.2.2

S78.22

Send CA(COMPUTE CRYPTOGRAPHIC CHECKSUM) Signal

6.15 State 9 – Waiting for Generate AC Response

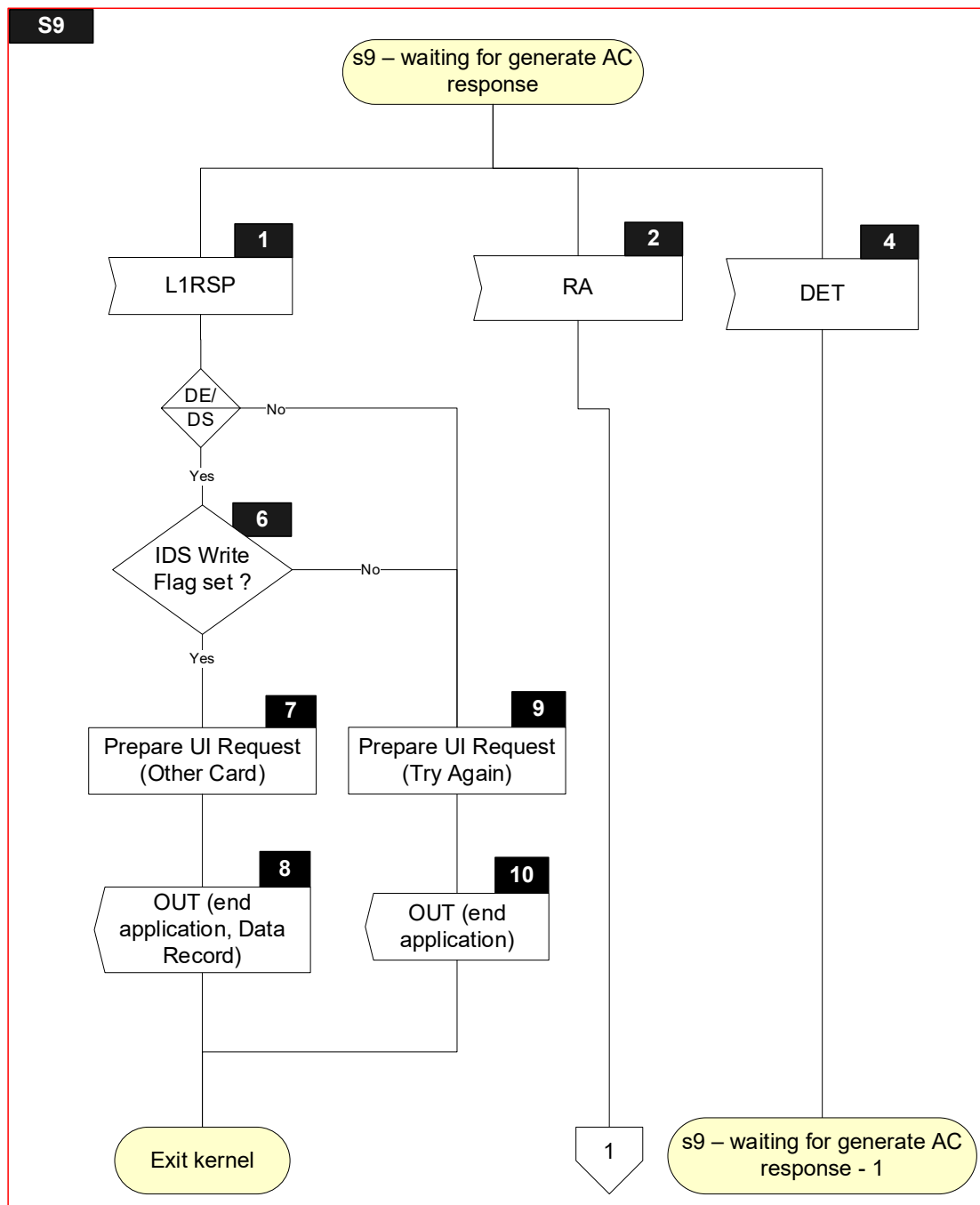
6.15.1 Local Variables

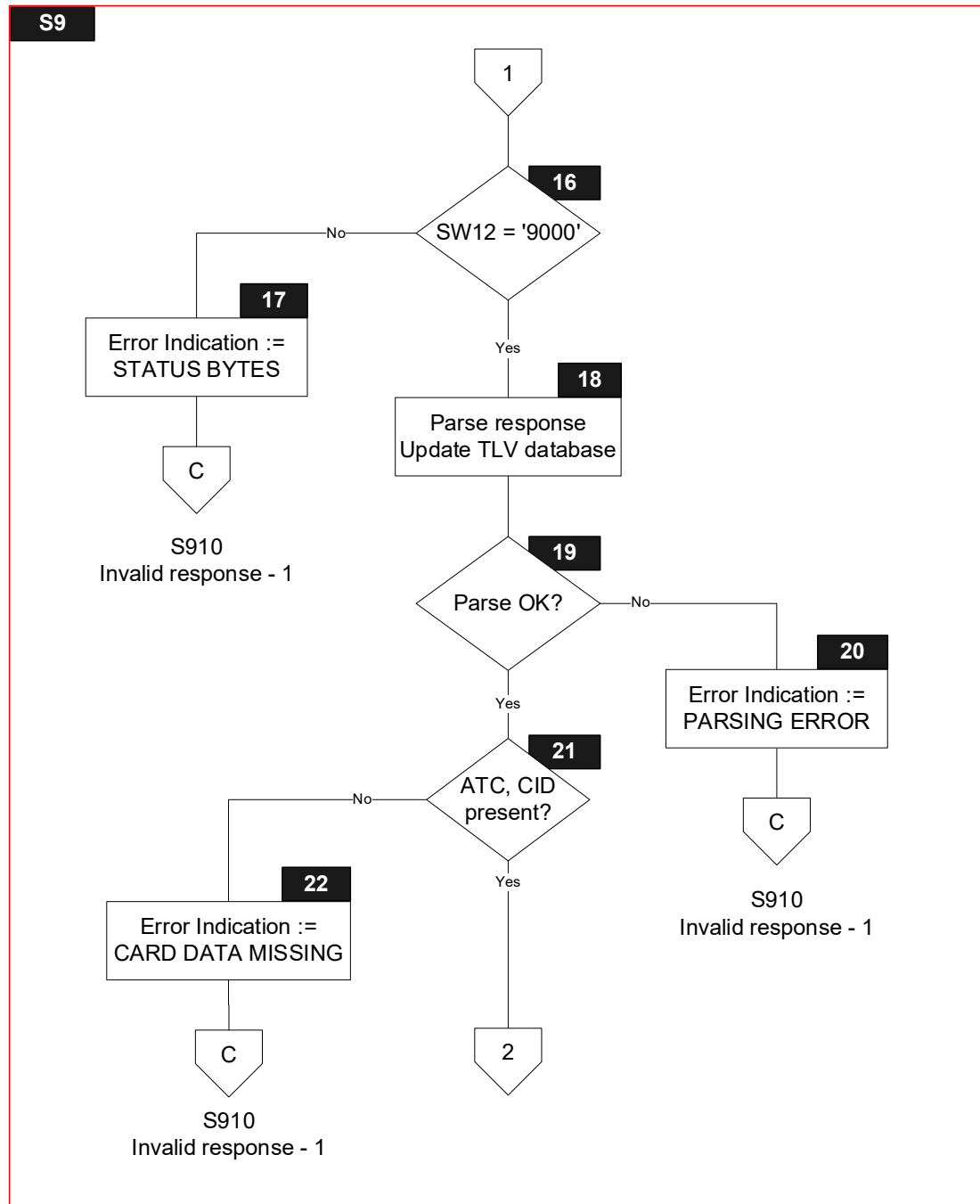
Name	Length	Format	Description
Return Code	1	b	Value returned with L1RSP Signal (TIME OUT ERROR, PROTOCOL ERROR, TRANSMISSION ERROR)
Parsing Result	1	b	Boolean used to store result of parsing a TLV string
SW12	2	b	Status bytes
Response Message Data Field	var. up to 256	b	TLV encoded string included in R-APDU of GENERATE AC
T	var.	b	Tag of TLV encoded string
L	var.	b	Length of TLV encoded string
V	var. up to 252	b	Value of TLV encoded string

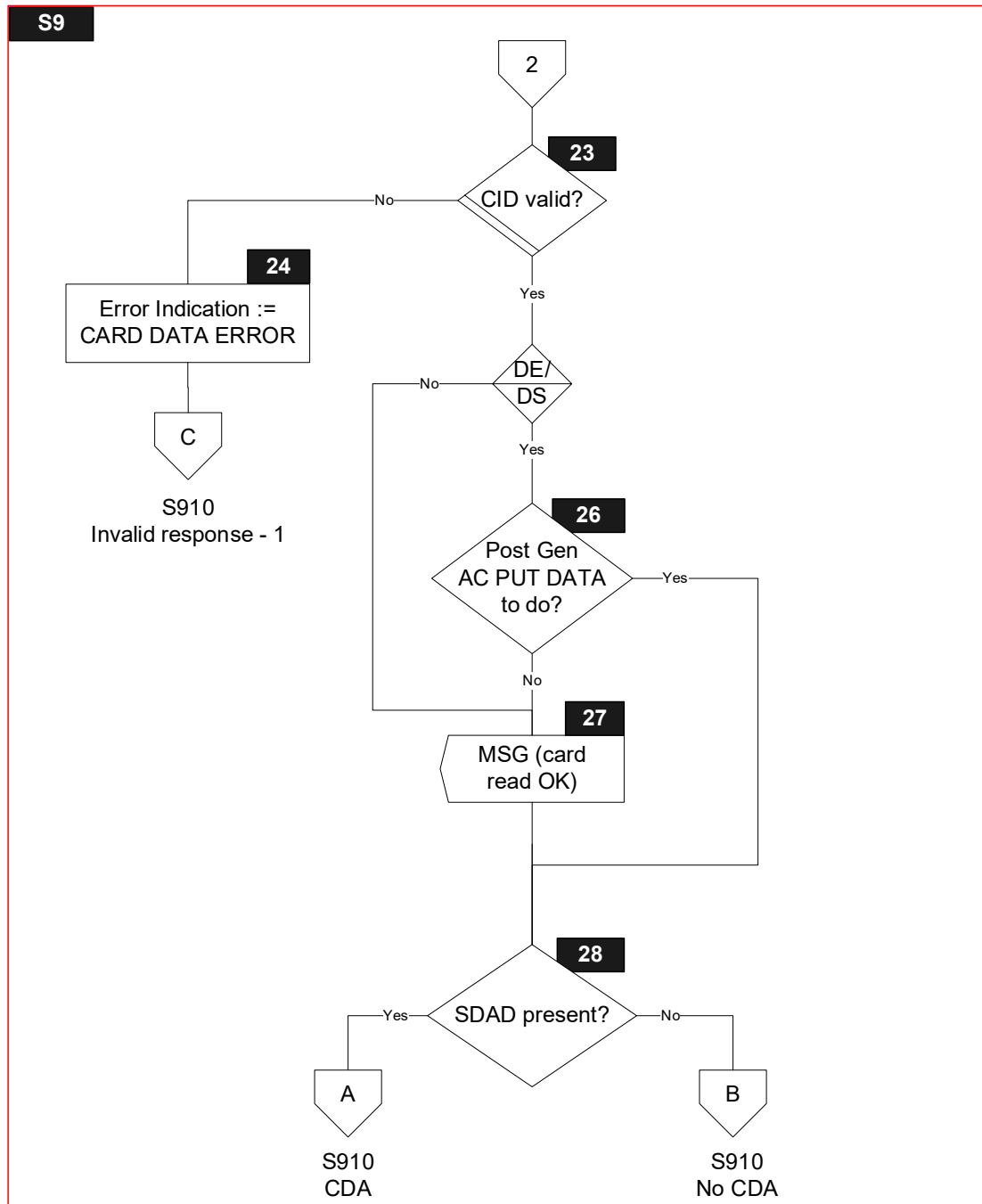
6.15.2 Flow Diagram

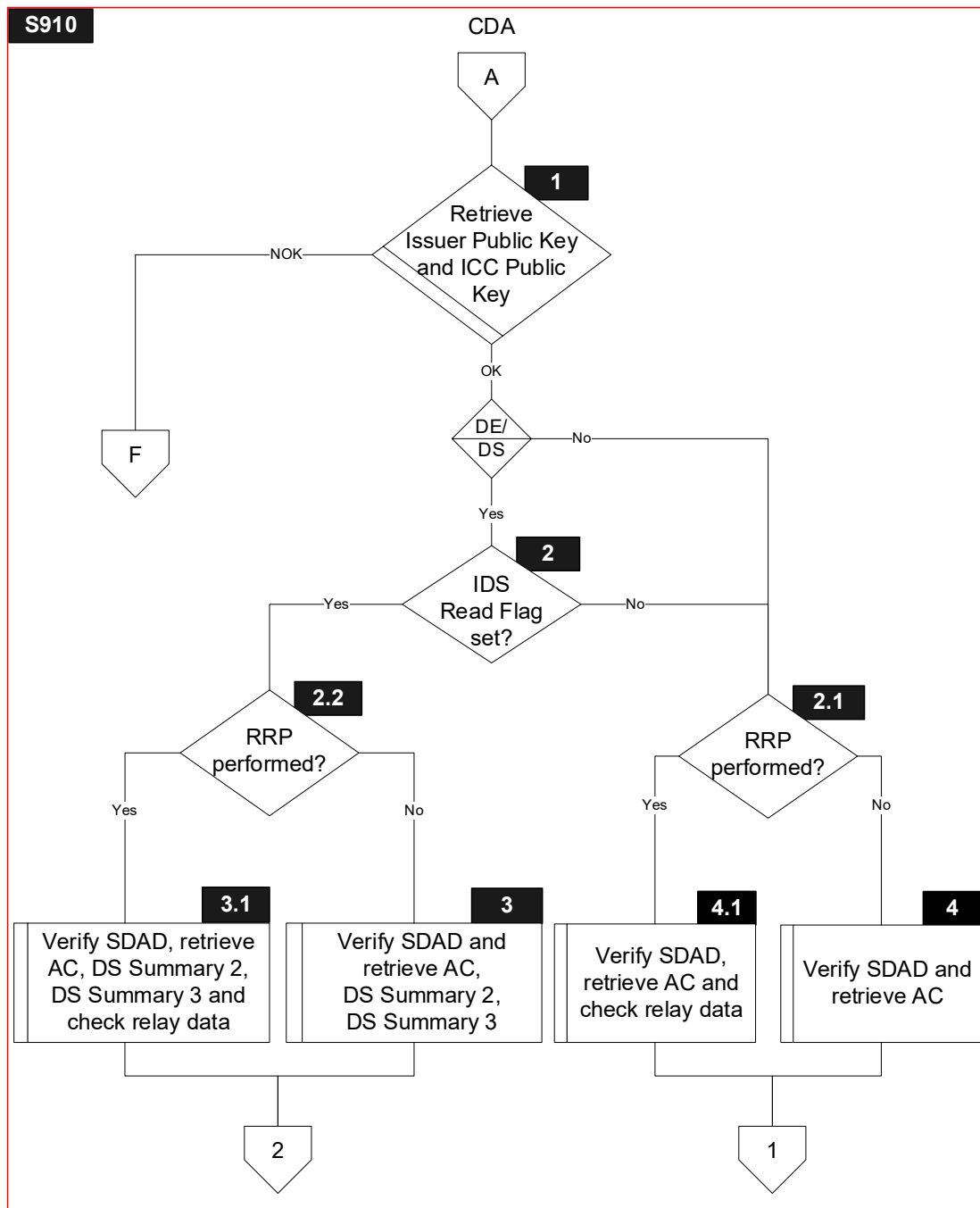
Figure 6.14 shows the flow diagram of `s9 – waiting for generate AC response`. Symbols in this diagram are labelled `S9.X` or `S910.X`.

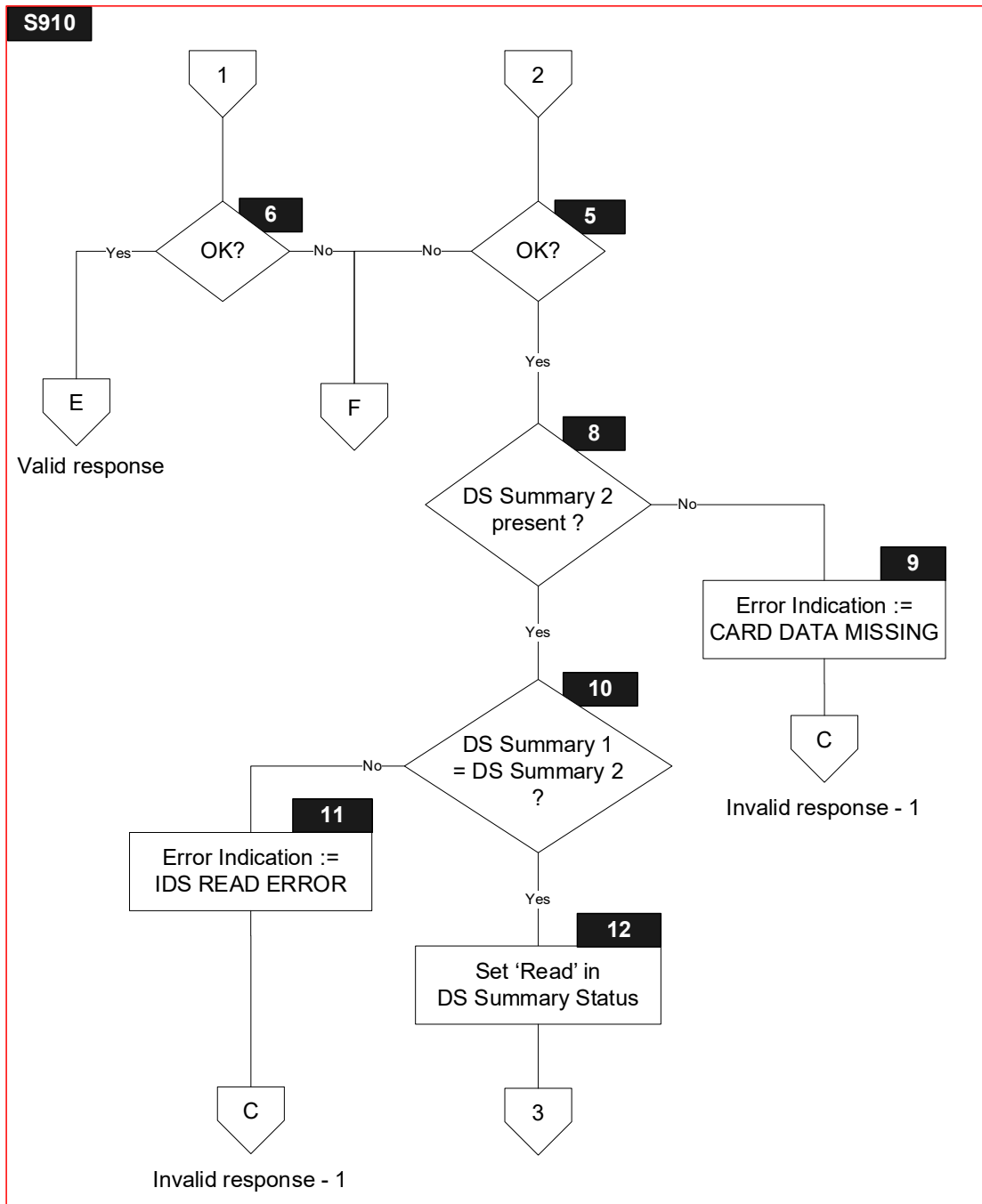
Figure 6.14—State 9 Flow Diagram

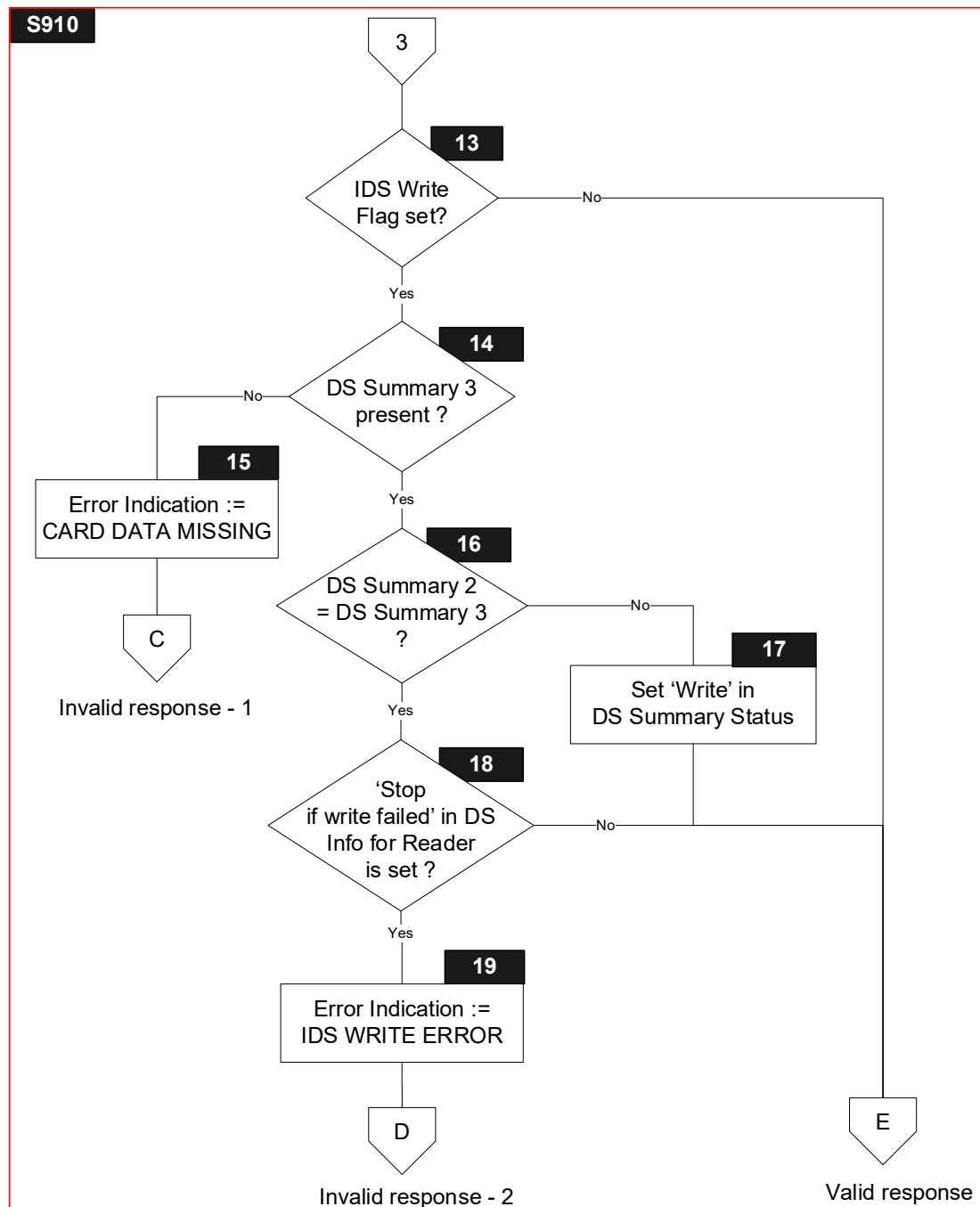




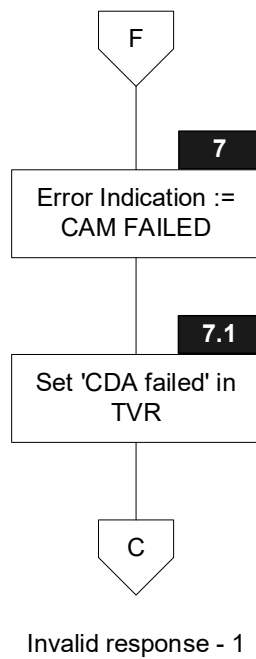


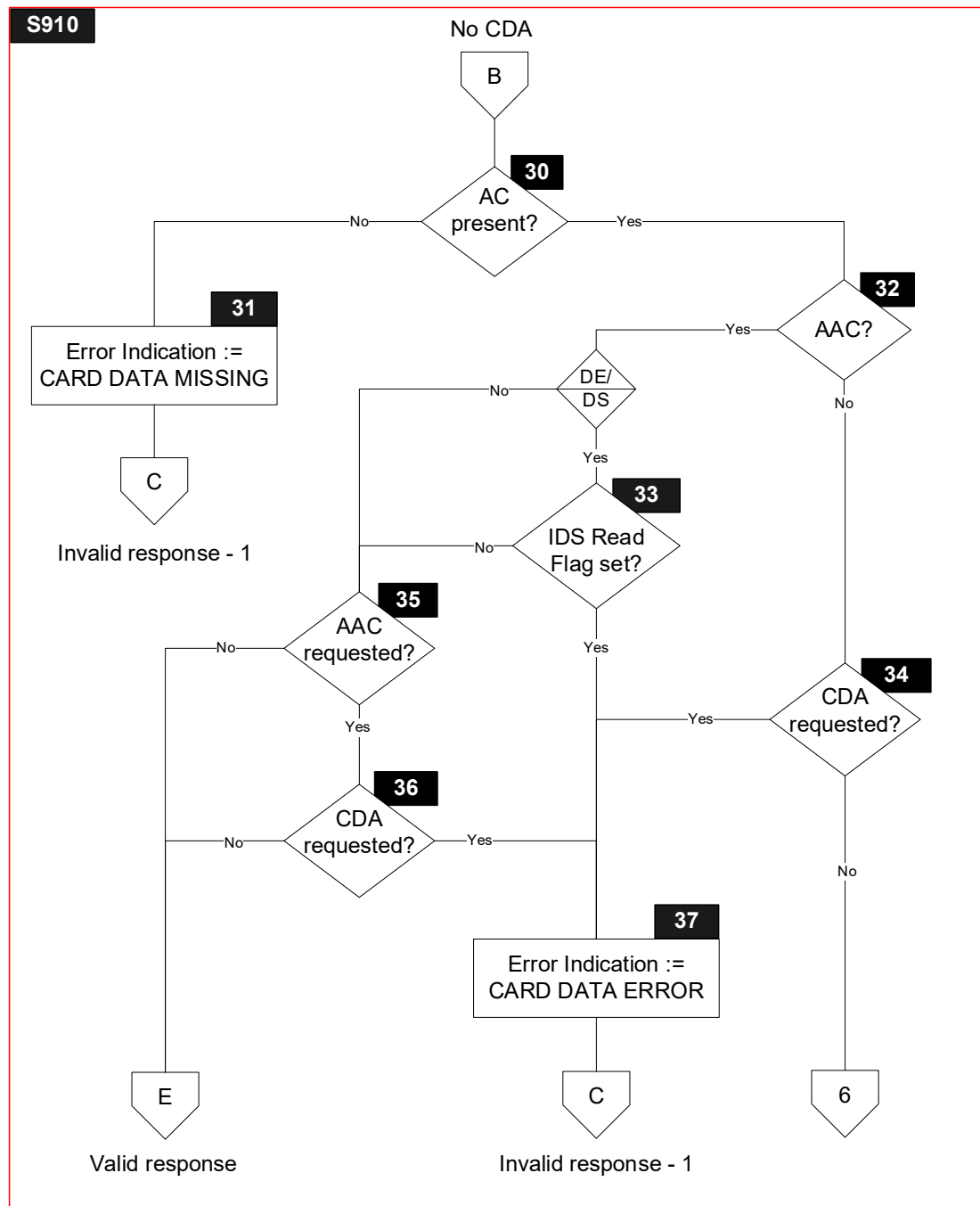


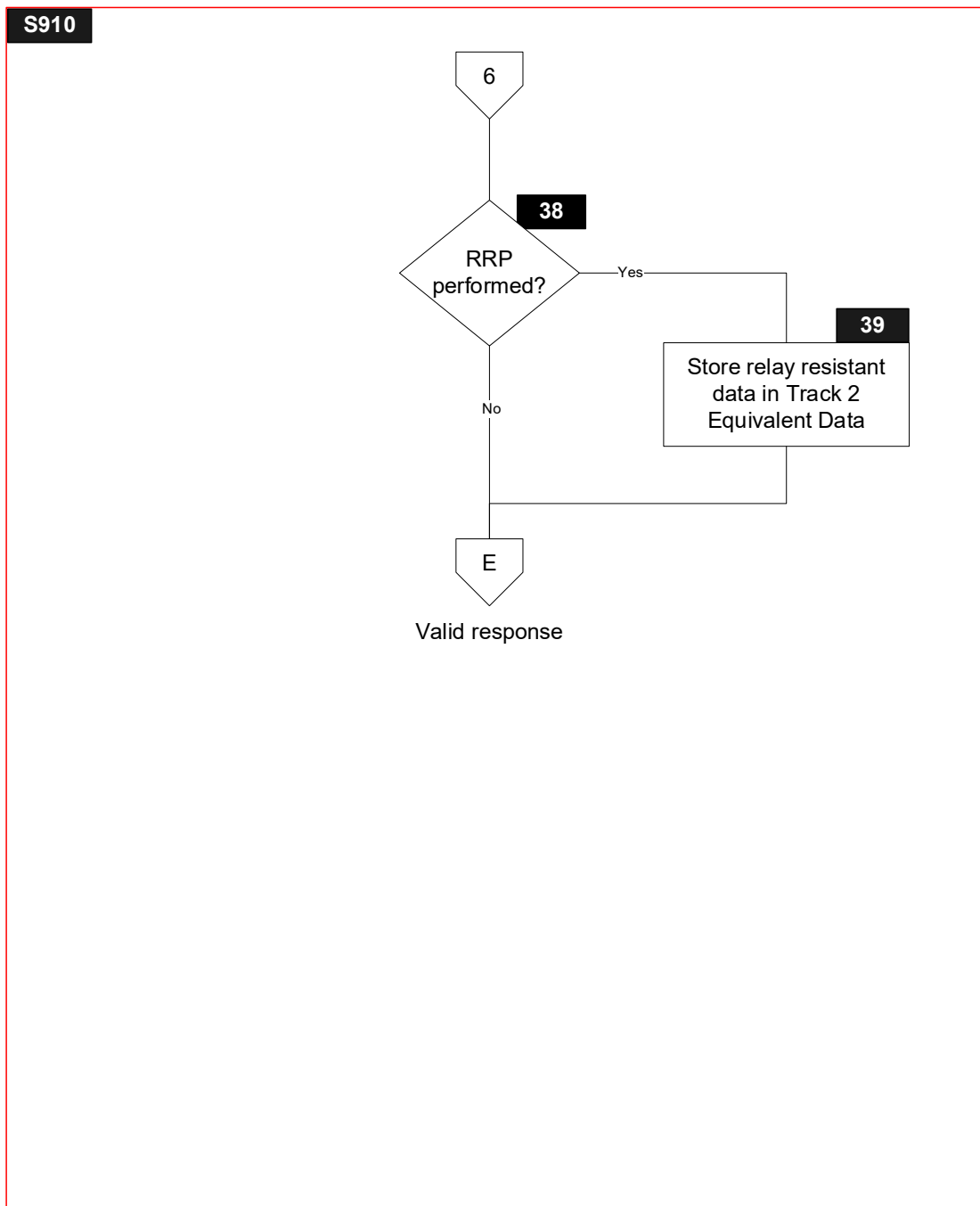




S910

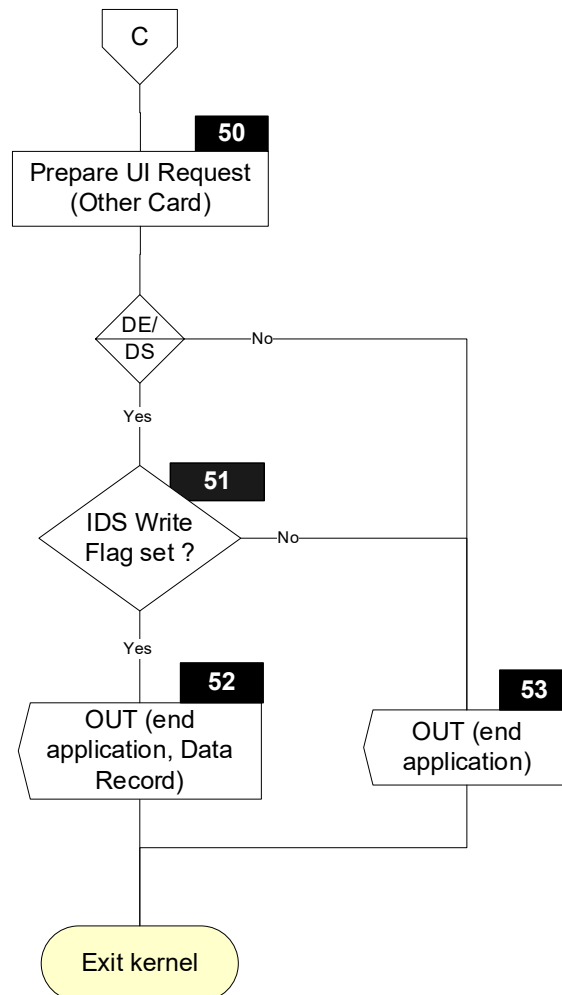






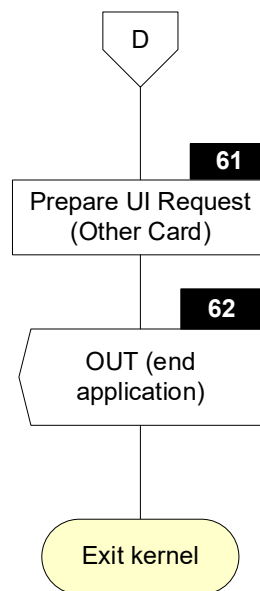
S910

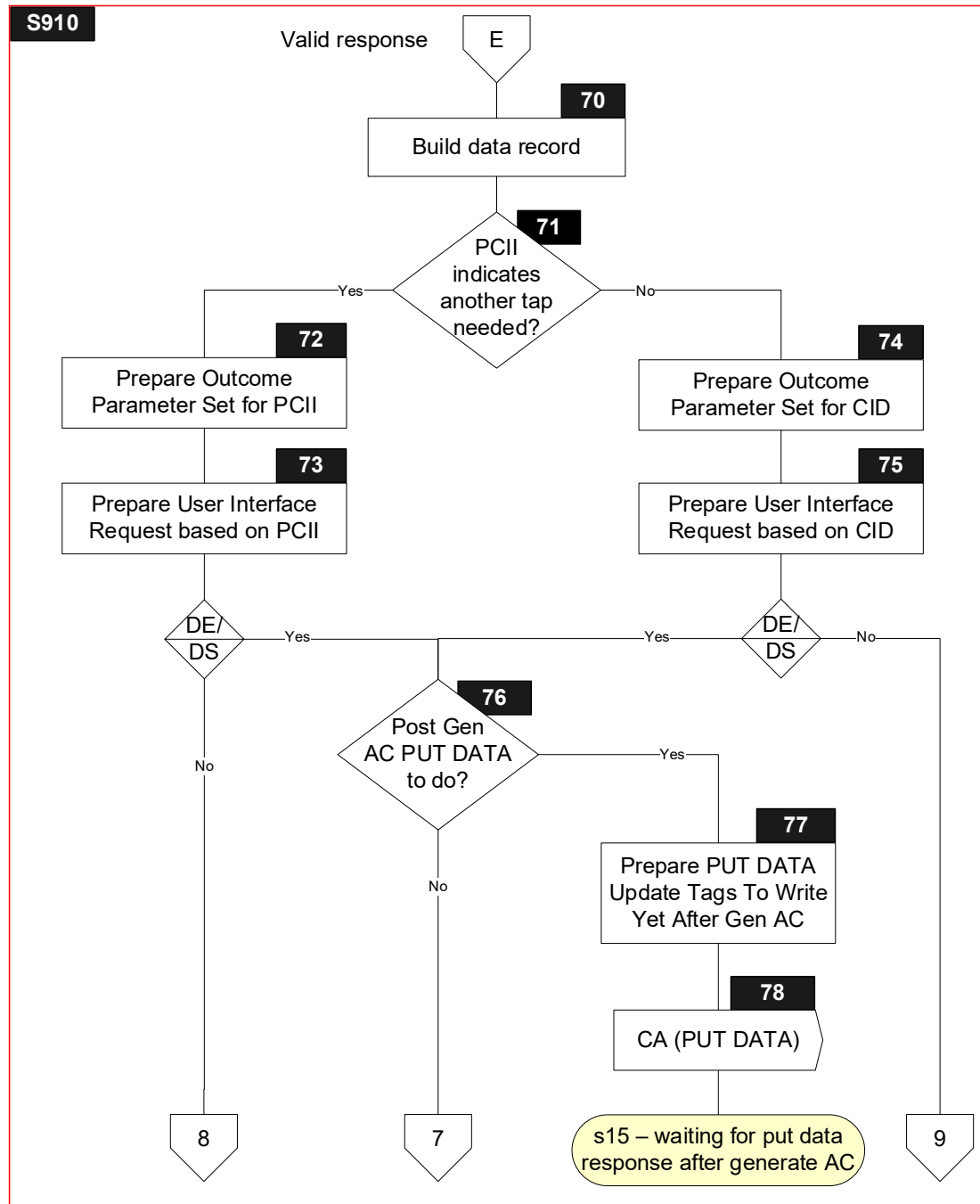
Invalid response - 1

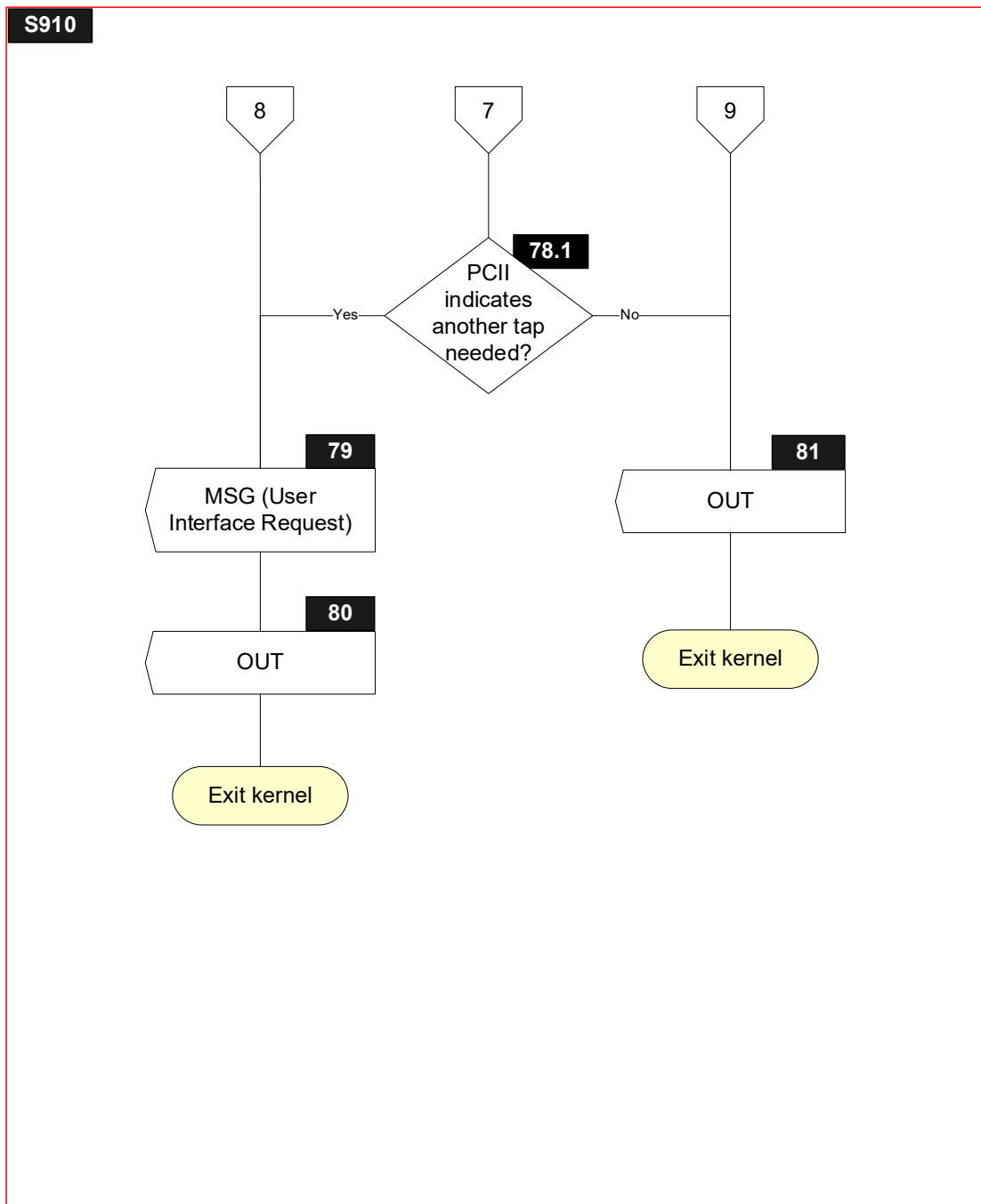


S910

Invalid response - 2







6.15.3 Processing

Note that symbols S9.6, S9.7, S9.8, S9.26, S910.2, S910.2.2, S910.3, S910.3.1, S910.5, S910.8, S910.9, S910.10, S910.11, S910.12, S910.13, S910.14, S910.15, S910.16, S910.17, S910.18, S910.19, S910.61, S910.62, S910.76, S910.77, S910.78 and S910.78.1 are only implemented for the DE/DS Implementation Option.

S9.1

Receive L1RSP Signal with Return Code

S9.2

Receive RA Signal with Response Message Data Field and SW12

S9.4

Receive DET Signal

S9.6

IF ['Write' in *IDS Status* is set]

THEN

GOTO S9.7

ELSE

GOTO S9.9

ENDIF

S9.7

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

S9.8

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

'L1' in *Error Indication* := Return Code

SET 'Data Record Present' in *Outcome Parameter Set*

CreateEMVDataRecord ()

CreateEMVDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Data Record*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*))) Signal

S9.9

'Message Identifier' in *User Interface Request Data* := TRY AGAIN

'Status' in *User Interface Request Data* := READY TO READ

'Hold Time' in *User Interface Request Data* := '000000'

S9.10

'Status' in *Outcome Parameter Set* := END APPLICATION

'Start' in *Outcome Parameter Set* := B

SET 'UI Request on Restart Present' in *Outcome Parameter Set*

'L1' in *Error Indication* := Return Code

'Msg On Error' in *Error Indication* := TRY AGAIN

CreateEMVDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*)))) Signal

S9.16

IF [SW12 = '9000']

THEN

GOTO S9.18

ELSE

GOTO S9.17

ENDIF

S9.17

'L2' in *Error Indication* := STATUS BYTES

'SW12' in *Error Indication* := SW12

S9.18

Parsing Result := FALSE

IF [(Length of Response Message Data Field > 0) AND
(Response Message Data Field[1] = '77')]

THEN

Parsing Result := ParseAndStoreCardResponse(Response Message Data Field)

ELSE

IF [(Length of Response Message Data Field > 0) AND
(Response Message Data Field[1] = '80')]

THEN

Parse the Response Message Data Field according to section 5.4.3 to retrieve the value field

IF [Response Message Data Field does not parse correctly OR
The length of the value field of the Response Message Data Field is less than 11 OR
The length of the value field of the Response Message Data Field is greater than 43 OR
IsEmpty(TagOf(*Cryptogram Information Data*)) OR
IsEmpty(TagOf(*Application Transaction Counter*)) OR
IsEmpty(TagOf(*Application Cryptogram*)) OR
(The length of the value field of the Response Message Data Field is greater than 11 AND
IsEmpty(TagOf(*Issuer Application Data*)))]

THEN

Parsing Result := FALSE

ELSE

Store the first byte of the value field of Response Message Data Field in the TLV Database for tag TagOf(*Cryptogram Information Data*).

Store from the second up to the third byte of the value field of Response Message Data Field in the TLV Database for tag TagOf(*Application Transaction Counter*).

Store from the fourth up to the eleventh byte of the value field of Response Message Data Field in the TLV Database for tag TagOf(*Application Cryptogram*).

If the length of the value field of the Response Message Data Field is greater than 11, then store from the twelfth up to the last byte of the value field of Response Message Data Field in the TLV Database for tag TagOf(*Issuer Application Data*).

Parsing Result := TRUE

ENDIF

ENDIF

ENDIF

S9.19

IF [Parsing Result]
THEN
 GOTO S9.21
ELSE
 GOTO S9.20
ENDIF

S9.20

'L2' in *Error Indication* := PARSING ERROR

S9.21

IF [IsEmpty(TagOf(*Application Transaction Counter*))
 AND IsEmpty(TagOf(*Cryptogram Information Data*))]
THEN
 GOTO S9.23
ELSE
 GOTO S9.22
ENDIF

S9.22

'L2' in *Error Indication* := CARD DATA MISSING

S9.23

IF [((*Cryptogram Information Data* AND 'C0' = '40') AND
 ('AC type' in *Reference Control Parameter* = TC))
 OR
 ((*Cryptogram Information Data* AND 'C0' = '80')
 AND
 (('AC type' in *Reference Control Parameter* = TC) OR
 ('AC type' in *Reference Control Parameter* = ARQC)))
 OR
 (*Cryptogram Information Data* AND 'C0' = '00')]
THEN
 GOTO S9.26 or S9.27
ELSE
 GOTO S9.24
ENDIF

S9.24

'L2' in *Error Indication* := CARD DATA ERROR

S9.26

```
IF      [IsEmptyList(Tags To Write Yet After Gen AC)]  
THEN  
    GOTO S9.28  
ELSE  
    GOTO S9.27  
ENDIF
```

S9.27

'Message Identifier' in *User Interface Request Data* := CLEAR DISPLAY
'Status' in *User Interface Request Data* := CARD READ SUCCESSFULLY
'Hold Time' in *User Interface Request Data* := '000000'
Send MSG(*User Interface Request Data*) Signal

S9.28

```
IF      [IsEmpty(TagOf(Signed Dynamic Application Data))]  
THEN  
    GOTO S910.1  
ELSE  
    GOTO S910.30  
ENDIF
```

CDA

S910.1

Retrieve with the *CA Public Key Index (Card)* the Certification Authority Public Key Modulus and Exponent and associated key related information, and the corresponding algorithm to be used from the CA Public Key Database (see section 4.4.2).

Retrieve the Issuer Public Key and ICC Public Key as described in section 6.3 and 6.4 of [EMV Book 2].

Check if the concatenation of the *CA Public Key Index (Card)* and the Certificate Serial Number recovered from the *Issuer Public Key Certificate* appears in the CRL. If this is the case, then ICC Public Key retrieval is not successful.

```
IF      [ICC Public Key retrieval was successful]  
THEN  
    GOTO S910.2 or S910.2.1  
ELSE  
    GOTO S910.7  
ENDIF
```

S910.2

```
IF      ['Read' in IDS Status is set]
THEN
    GOTO S910.2.2
ELSE
    GOTO S910.2.1
ENDIF
```

S910.2.1

```
IF      ['Relay resistance performed' in Terminal Verification Results =
        RRP PERFORMED]
THEN
    GOTO S910.4.1
ELSE
    GOTO S910.4
ENDIF
```

S910.2.2

```
IF      ['Relay resistance performed' in Terminal Verification Results =
        RRP PERFORMED]
THEN
    GOTO S910.3.1
ELSE
    GOTO S910.3
ENDIF
```

S910.3

Verify *Signed Dynamic Application Data* as in section 6.6 of [EMV Book 2]. Use *PDOL Values* for the values of the data elements specified by, and in the order they appear in the PDOL.

If the length of ICC Dynamic Data is less than 30 + Length of ICC Dynamic Number bytes, then CDA fails.

Retrieve from the ICC Dynamic Data (see Table 6.5) the *ICC Dynamic Number*, *Application Cryptogram*, *DS Summary 2* and *DS Summary 3* and store in the TLV Database.

If the ICC Dynamic Data does not include *DS Summary 3* (i.e. there are less than 16 bytes after Hash Result (if 'Data Storage Version Number' in *Application Capabilities Information* = VERSION 1) or less than 32 bytes after Hash Result (if 'Data Storage Version Number' in *Application Capabilities Information* = VERSION 2)), then do not store *DS Summary 3*. This is not a reason to fail CDA.

If the ICC Dynamic Data also does not include *DS Summary 2* (i.e. there are less than 8 bytes after Hash Result (if 'Data Storage Version Number' in *Application Capabilities Information* = VERSION 1) or less than 16 bytes after Hash Result (if 'Data Storage Version Number' in *Application Capabilities Information* = VERSION 2)), then do not store *DS Summary 2*. This is not a reason to fail CDA.

Table 6.5—ICC Dynamic Data (IDS)

Value	Length
Length of ICC Dynamic Number	1
ICC Dynamic Number	2-8
Cryptogram Information Data	1
Application Cryptogram	8
Hash Result	20
DS Summary 2	8 or 16
DS Summary 3	8 or 16

S910.3.1

Verify *Signed Dynamic Application Data* as in section 6.6 of [EMV Book 2]. Use *PDOL Values* for the values of the data elements specified by, and in the order they appear in the PDOL.

If the length of ICC Dynamic Data is less than 60 + Length of ICC Dynamic Number bytes (if 'Data Storage Version Number' in *Application Capabilities Information* = VERSION 1) or less than 76 + Length of ICC Dynamic Number bytes (if 'Data Storage Version Number' in *Application Capabilities Information* = VERSION 2)), then CDA fails.

Retrieve from the ICC Dynamic Data (see Table 6.6) the *ICC Dynamic Number*, *Application Cryptogram*, *DS Summary 2* and *DS Summary 3* and store in the TLV Database.

Check if the relay resistance related data objects in the ICC Dynamic Data match the corresponding data objects in the TLV Database as follows:

Terminal Relay Resistance Entropy = Terminal Relay Resistance Entropy (CDA) and
Device Relay Resistance Entropy = Device Relay Resistance Entropy (CDA) and
Min Time For Processing Relay Resistance APDU = Min Time For Processing Relay Resistance APDU (CDA) and

Max Time For Processing Relay Resistance APDU = Max Time For Processing Relay Resistance APDU (CDA) and

Device Estimated Transmission Time For Relay Resistance R-APDU = Device Estimated Transmission Time For Relay Resistance R-APDU (CDA).

If this is not the case, then CDA fails.

Table 6.6—ICC Dynamic Data (IDS + RRP)

Value	Length
Length of ICC Dynamic Number	1
ICC Dynamic Number	2-8
Cryptogram Information Data	1
Application Cryptogram	8
Hash Result	20
DS Summary 2	8 or 16
DS Summary 3	8 or 16
Terminal Relay Resistance Entropy (CDA)	4
Device Relay Resistance Entropy (CDA)	4
Min Time For Processing Relay Resistance APDU (CDA)	2
Max Time For Processing Relay Resistance APDU (CDA)	2
Device Estimated Transmission Time For Relay Resistance R-APDU (CDA)	2

S910.4

Verify *Signed Dynamic Application Data* as in section 6.6 of [EMV Book 2]. Use *PDOL Values* for the values of the data elements specified by, and in the order they appear in the PDOL.

If the length of ICC Dynamic Data is less than 30 + Length of ICC Dynamic Number bytes, then CDA fails.

Retrieve from the ICC Dynamic Data (see Table 6.7) the *ICC Dynamic Number* and *Application Cryptogram* and store in the TLV Database.

Table 6.7—ICC Dynamic Data (No IDS)

Value	Length
Length of ICC Dynamic Number	1
ICC Dynamic Number	2-8
Cryptogram Information Data	1
Application Cryptogram	8
Hash Result	20

S910.4.1

Verify *Signed Dynamic Application Data* as in section 6.6 of [EMV Book 2]. Use *PDOL Values* for the values of the data elements specified by, and in the order they appear in the PDOL.

If the length of ICC Dynamic Data is less than 44 + Length of ICC Dynamic Number bytes, then CDA fails.

Retrieve from the ICC Dynamic Data (see Table 6.8) the *ICC Dynamic Number* and *Application Cryptogram* and store in the TLV Database.

Check if the relay resistance related data objects in the ICC Dynamic Data match the corresponding data objects in the TLV Database as follows:

Terminal Relay Resistance Entropy = Terminal Relay Resistance Entropy (CDA) and

Device Relay Resistance Entropy = Device Relay Resistance Entropy (CDA) and

Min Time For Processing Relay Resistance APDU = Min Time For Processing Relay Resistance APDU (CDA) and

Max Time For Processing Relay Resistance APDU = Max Time For Processing Relay Resistance APDU (CDA) and

Device Estimated Transmission Time For Relay Resistance R-APDU = Device Estimated Transmission Time For Relay Resistance R-APDU (CDA).

If this is not the case, then CDA fails.

Table 6.8—ICC Dynamic Data (RRP)

Value	Length
Length of ICC Dynamic Number	1
ICC Dynamic Number	2-8
Cryptogram Information Data	1
Application Cryptogram	8
Hash Result	20
Terminal Relay Resistance Entropy (CDA)	4
Device Relay Resistance Entropy (CDA)	4
Min Time For Processing Relay Resistance APDU (CDA)	2
Max Time For Processing Relay Resistance APDU (CDA)	2
Device Estimated Transmission Time For Relay Resistance R-APDU (CDA)	2

S910.5

IF [Signed Dynamic Application Data verification is OK]
THEN
 GOTO S910.8
ELSE
 GOTO S910.7
ENDIF

S910.6

IF [Signed Dynamic Application Data verification is OK]
THEN
 GOTO S910.70
ELSE
 GOTO S910.7
ENDIF

S910.7

'L2' in *Error Indication* := CAM FAILED

S910.7.1

SET 'CDA Failed' in *Terminal Verification Results*

S910.8

```
IF      [IsNotEmpty(TagOf(DS Summary 2))]  
THEN  
    GOTO S910.10  
ELSE  
    GOTO S910.9  
ENDIF
```

S910.9

'L2' in *Error Indication* := CARD DATA MISSING

S910.10

```
IF      [DS Summary 1 = DS Summary 2]  
THEN  
    GOTO S910.12  
ELSE  
    GOTO S910.11  
ENDIF
```

S910.11

'L2' in *Error Indication* := IDS READ ERROR

S910.12

SET 'Successful Read' in *DS Summary Status*

S910.13

```
IF      ['Write' in IDS Status is set]  
THEN  
    GOTO S910.14  
ELSE  
    GOTO S910.70  
ENDIF
```

S910.14

```
IF      [IsPresent(TagOf(DS Summary 3))]  
THEN  
    GOTO S910.16  
ELSE  
    GOTO S910.15  
ENDIF
```

S910.15

'L2' in *Error Indication* := CARD DATA MISSING

S910.16

```
IF      [DS Summary 2 = DS Summary 3]
THEN
    GOTO S910.18
ELSE
    GOTO S910.17
ENDIF
```

S910.17

SET 'Successful Write' in *DS Summary Status*

S910.18

```
IF      ['Stop if write failed' in DS ODS Info For Reader is set]
THEN
    GOTO S910.19
ELSE
    GOTO S910.70
ENDIF
```

S910.19

'L2' in *Error Indication* := IDS WRITE ERROR

No CDA

S910.30

```
IF      [IsNotEmpty(TagOf(Application Cryptogram))]  
THEN  
    GOTO S910.32  
ELSE  
    GOTO S910.31  
ENDIF
```

S910.31

'L2' in *Error Indication* := CARD DATA MISSING

S910.32

```
IF      [(Cryptogram Information Data AND 'C0') = '00']  
THEN  
    GOTO S910.33 or S910.35  
ELSE  
    GOTO S910.34  
ENDIF
```

S910.33

```
IF      ['Read' in IDS Status is set]  
THEN  
    GOTO S910.37  
ELSE  
    GOTO S910.35  
ENDIF
```

S910.34

```
IF      ['CDA signature requested' in Reference Control Parameter is set]  
THEN  
    GOTO S910.37  
ELSE  
    GOTO S910.38  
ENDIF
```

S910.35

```
IF      ['AC type' in Reference Control Parameter = AAC]  
THEN  
    GOTO S910.36  
ELSE  
    GOTO S910.70  
ENDIF
```

S910.36

```
IF      ['CDA signature requested' in Reference Control Parameter is set]
THEN
    GOTO S910.37
ELSE
    GOTO S910.70
ENDIF
```

S910.37

'L2' in *Error Indication* := CARD DATA ERROR

S910.38

```
IF      ['Relay resistance performed' in Terminal Verification Results =
        RRP PERFORMED]
THEN
    GOTO S910.39
ELSE
    GOTO S910.70
ENDIF
```

S910.39

```
IF      [IsEmpty(TagOf(Track 2 Equivalent Data))]
THEN
    IF   [Number of digits in 'Primary Account Number' in Track 2 Equivalent
        Data ≤ 16]
    THEN
        Replace 'Discretionary Data' in Track 2 Equivalent Data with
        '000000000000' (13 hexadecimal zeroes). Pad with 'F' if needed to
        ensure whole bytes.
    ELSE
        Replace 'Discretionary Data' in Track 2 Equivalent Data with
        '0000000000' (10 hexadecimal zeroes). Pad with 'F' if needed to
        ensure whole bytes.
    ENDIF
    IF   [IsEmpty(TagOf(CA Public Key Index (Card))) AND
        CA Public Key Index (Card) < '0A']
    THEN
        Replace the most significant digit of the 'Discretionary Data' in Track 2
        Equivalent Data with a digit representing CA Public Key Index (Card).
    ENDIF
    Replace the second most significant digit of the 'Discretionary Data' in Track 2
    Equivalent Data with a digit representing RRP Counter.
```

Convert the two least significant bytes of the *Device Relay Resistance Entropy* from 2 byte binary to 5 digit decimal by considering the two bytes as an integer in the range 0 to 65535. Replace the 5 digits of 'Discretionary Data' in *Track 2 Equivalent Data* that follow the *RRP Counter* digit with that value.

IF [Number of digits in 'Primary Account Number' in *Track 2 Equivalent Data* \leq 16]

THEN

Convert the third least significant byte of *Device Relay Resistance Entropy* from binary to 3 digit decimal in the range 0 to 255. Replace the next 3 digits of 'Discretionary Data' in *Track 2 Equivalent Data* with that value.

ENDIF

Divide the *Measured Relay Resistance Processing Time* by 10 using the div operator to give a count in milliseconds. If the value exceeds '03E7' (999), then set the value to '03E7'. Convert this value from 2 byte binary to 3 digit decimal by considering the 2 bytes as an integer. Replace the 3 least significant digits of 'Discretionary Data' in *Track 2 Equivalent Data* with this 3 digit decimal value.

ENDIF

Invalid Response – 1

S910.50

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

'Hold Time' in *User Interface Request Data* := Message Hold Time

S910.51

IF ['Write' in *IDS Status* is set]

THEN

GOTO S910.52

ELSE

GOTO S910.53

ENDIF

S910.52

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

SET 'Data Record Present' in *Outcome Parameter Set*

CreateEMVDataRecord ()

CreateEMVDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Data Record*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*)))) Signal

S910.53

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

CreateEMVDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)), GetTLV(TagOf(*Discretionary Data*)), GetTLV(TagOf(*User Interface Request Data*)))) Signal

Invalid Response – 2

S910.61

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

'Hold Time' in *User Interface Request Data* := Message Hold Time

S910.62

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := ERROR – OTHER CARD

CreateEMVDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*))) Signal

Valid Response

S910.70

SET 'Data Record Present' in *Outcome Parameter Set*
CreateEMVDataRecord ()

S910.71

IF [IsEmpty(TagOf(*POS Cardholder Interaction Information*)) AND
(*POS Cardholder Interaction Information* AND '00030F' ≠ '000000')]
THEN
GOTO S910.72
ELSE
GOTO S910.74
ENDIF

S910.72

'Status' in *Outcome Parameter Set* := END APPLICATION
'Start' in *Outcome Parameter Set* := B

S910.73

FOR every entry in the *Phone Message Table*
{
IF [(PCII Mask AND *POS Cardholder Interaction Information*) =
PCII Value]
THEN
'Hold Time' in *User Interface Request Data* := *Message Hold Time*
'Message Identifier' in *User Interface Request Data* := *Message Identifier*
'Status' in *User Interface Request Data* := *Status*
EXIT loop
ENDIF
}

S910.74

```
IF      [(Cryptogram Information Data AND 'C0') = '40']
THEN
    'Status' in Outcome Parameter Set := APPROVED
ELSE
    IF [(Cryptogram Information Data AND 'C0') = '80']
    THEN
        'Status' in Outcome Parameter Set := ONLINE REQUEST
    ELSE
        Check if Transaction Type indicates a cash transaction (cash
        withdrawal or cash disbursement) or a purchase transaction (purchase
        or purchase with cashback).
        IF      [Transaction Type = '01' OR Transaction Type = '17' OR
                Transaction Type = '00' OR Transaction Type = '09']
        THEN
            IF      [(IsEmpty(TagOf(Third Party Data))) AND
                    ('Unique Identifier' in Third Party Data AND '8000' =
                     '0000') AND
                    ('Device Type' in Third Party Data ≠ '3030'))
            OR
            ('IC with contacts' in Terminal Capabilities is not set)]
            THEN
                'Status' in Outcome Parameter Set := DECLINED
            ELSE
                'Status' in Outcome Parameter Set := TRY ANOTHER
                INTERFACE
            ENDIF
        ELSE
            'Status' in Outcome Parameter Set := END APPLICATION
        ENDIF
    ENDIF
ENDIF
```

S910.75

```
'Status' in User Interface Request Data := NOT READY
IF      [(Cryptogram Information Data AND 'C0') = '40']
THEN
    'Hold Time' in User Interface Request Data := Message Hold Time
    IF      ['CVM' in Outcome Parameter Set = OBTAIN SIGNATURE]
    THEN
        'Message Identifier' in User Interface Request Data := APPROVED –
        SIGN
    ELSE
        'Message Identifier' in User Interface Request Data := APPROVED
    ENDIF
ELSE
    IF      [(Cryptogram Information Data AND 'C0') = '80']
    THEN
        'Hold Time' in User Interface Request Data := '000000'
        'Message Identifier' in User Interface Request Data :=
        AUTHORISING – PLEASE WAIT
    ELSE
        Check if Transaction Type indicates a cash transaction (cash
        withdrawal or cash disbursement) or a purchase transaction (purchase
        or purchase with cashback).
        IF      [Transaction Type = '01' OR Transaction Type = '17' OR
        Transaction Type = '00' OR Transaction Type = '09']
        THEN
            'Hold Time' in User Interface Request Data := Message Hold
            Time
```

```
IF      [(IsEmpty(TagOf(Third Party Data)) AND
        ('Unique Identifier' in Third Party Data AND '8000' =
        '0000') AND
        ('Device Type' in Third Party Data ≠ '3030'))
        OR
        ('IC with contacts' in Terminal Capabilities is not set) ]
THEN
    'Message Identifier' in User Interface Request Data :=
    DECLINED
ELSE
    'Message Identifier' in User Interface Request Data :=
    INSERT CARD
ENDIF
ELSE
    'Hold Time' in User Interface Request Data := '000000'
    'Message Identifier' in User Interface Request Data :=
    CLEAR DISPLAY
ENDIF
ENDIF
ENDIF
```

S910.76

```
IF      [IsEmptyList(Tags To Write Yet After Gen AC)]
THEN
    GOTO S910.77
ELSE
    GOTO S910.78.1
ENDIF
```

S910.77

TLV = GetAndRemoveFromList(*Tags To Write Yet After Gen AC*)

Prepare the PUT DATA command with TLV as defined in section 5.7

S910.78

Send CA(PUT DATA command) Signal

S910.78.1

IF [IsNotEmpty(TagOf(*POS Cardholder Interaction Information*)) AND
(*POS Cardholder Interaction Information* AND '00030F' ≠ '000000')]
THEN
 GOTO S910.79
ELSE
 GOTO S910.81
ENDIF

S910.79

Send MSG(*User Interface Request Data*) Signal

S910.80

CreateEMVDiscretionaryData ()
SET 'UI Request on Restart Present' in *Outcome Parameter Set*
'Status' in *User Interface Request Data* := READY TO READ
'Hold Time' in *User Interface Request Data* := '000000'
Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Data Record*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*))) Signal

S910.81

CreateEMVDiscretionaryData ()
SET 'UI Request on Outcome Present' in *Outcome Parameter Set*
Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Data Record*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*))) Signal

6.16 State 12 – Waiting for Put Data Response Before Generate AC

State 12 is a state specific to SDS processing and is only implemented for the DE/DS Implementation Option.

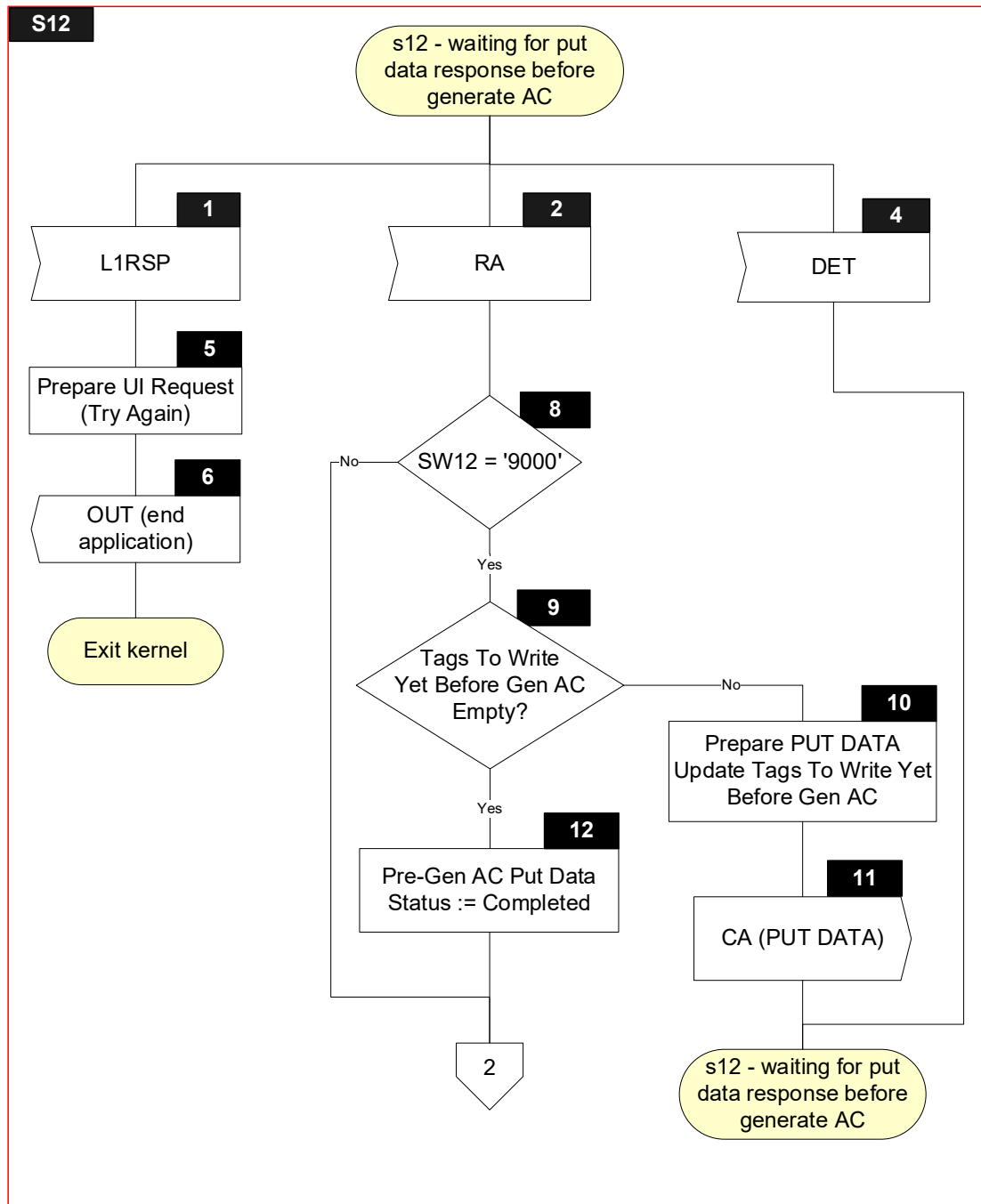
6.16.1 Local Variables

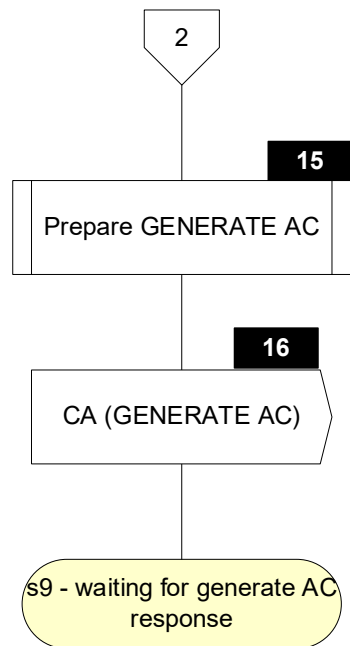
Name	Length	Format	Description
Return Code	1	b	Value returned with L1RSP Signal (TIME OUT ERROR, PROTOCOL ERROR, TRANSMISSION ERROR)
SW12	2	b	Status bytes
T	var.	b	Tag of TLV encoded string
L	var.	b	Length of TLV encoded string
V	var. up to 252	b	Value of TLV encoded string

6.16.2 Flow Diagram

Figure 6.15 shows the flow diagram of `s12 - waiting for put data response before generate AC`. Symbols in this diagram are labelled `S12.X`.

Figure 6.15—State 12 Flow Diagram



S12

6.16.3 Processing

S12.1

Receive L1RSP Signal with Return Code

S12.2

Receive RA Signal with SW12

S12.4

Receive DET Signal

S12.5

'Message Identifier' in *User Interface Request Data* := TRY AGAIN

'Status' in *User Interface Request Data* := READY TO READ

'Hold Time' in *User Interface Request Data* := '000000'

S12.6

'Status' in *Outcome Parameter Set* := END APPLICATION

'Start' in *Outcome Parameter Set* := B

SET 'UI Request on Restart Present' in *Outcome Parameter Set*

'L1' in *Error Indication* := Return Code

'Msg On Error' in *Error Indication* := TRY AGAIN

CreateEMVDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*)))) Signal

S12.7

'Status' in *Outcome Parameter Set* := END APPLICATION

'L3' in *Error Indication* := STOP

CreateEMVDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)))) Signal

S12.8

IF [SW12 = '9000']

THEN

GOTO S12.9

ELSE

GOTO S12.15

ENDIF

S12.9

IF [IsEmptyList(*Tags To Write Yet Before Gen AC*)]

THEN

GOTO S12.12

ELSE

GOTO S12.10

ENDIF

S12.10

TLV := GetAndRemoveFromList(*Tags To Write Yet Before Gen AC*)

Prepare PUT DATA command for TLV as specified in section 5.7

S12.11

Send CA(PUT DATA command) Signal

S12.12

SET 'Completed' in *Pre-Gen AC Put Data Status*

S12.15

Prepare GENERATE AC command as specified in section 7.2

S12.16

Send CA(GENERATE AC) Signal

6.17 State 13 – Waiting for CCC Response – 1

State 13 is a state specific to Mag-stripe Mode and is only implemented for the MAG Implementation Option.

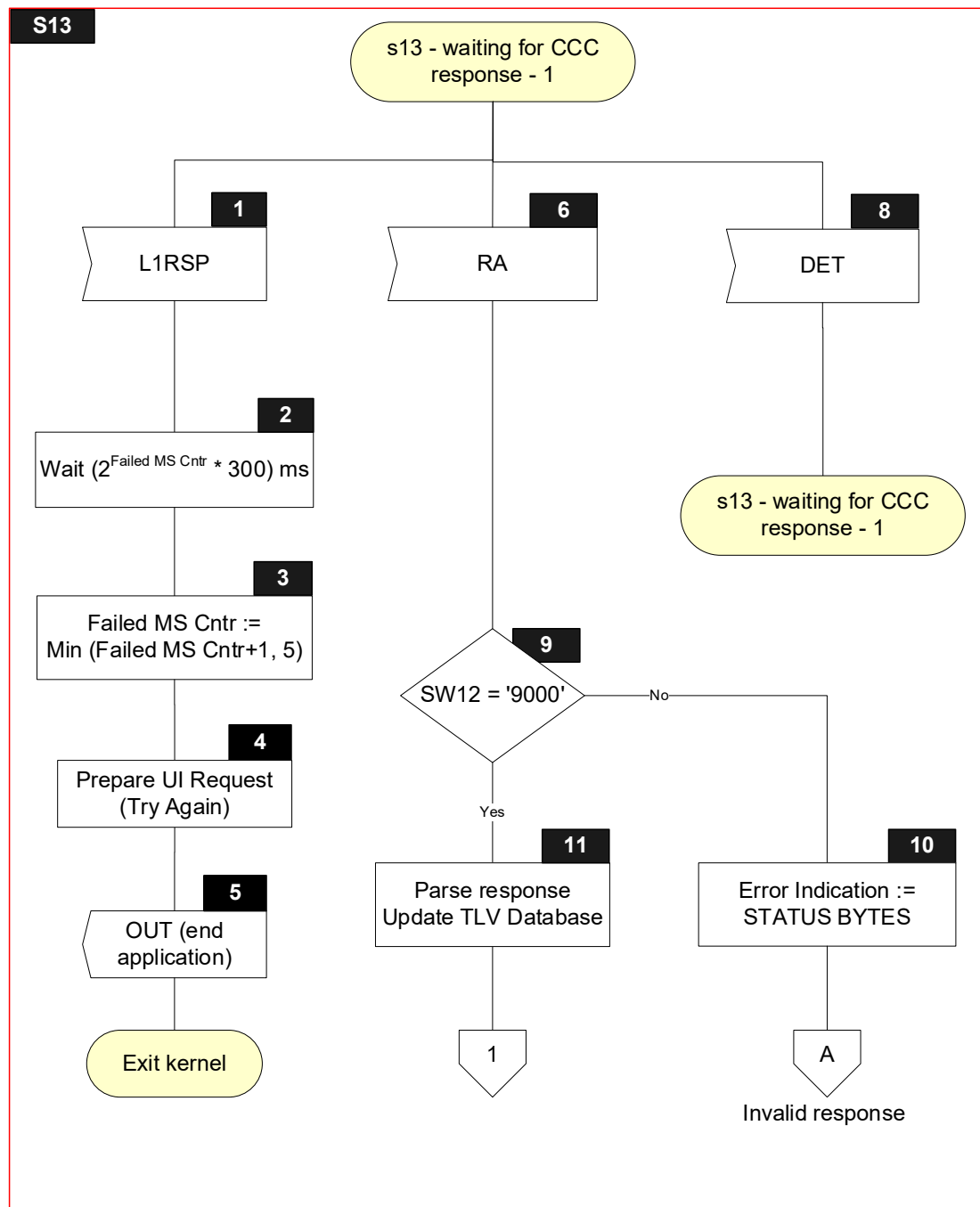
6.17.1 Local Variables

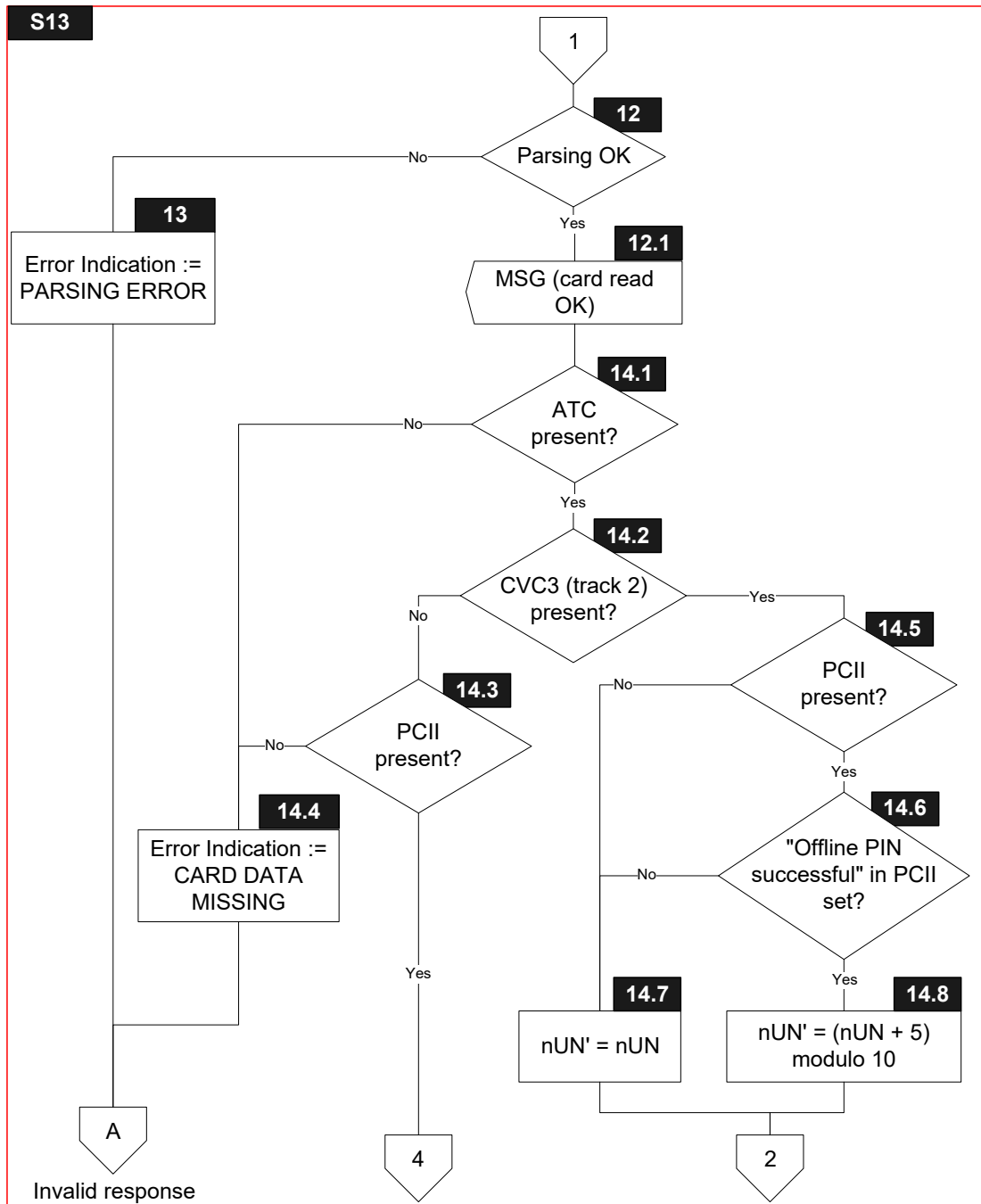
Name	Length	Format	Description
Return Code	1	b	Value returned with L1RSP Signal (TIME OUT ERROR, PROTOCOL ERROR, TRANSMISSION ERROR)
Parsing Result	1	b	Boolean used to store result of parsing a TLV string
SW12	2	b	Status bytes
nUN'	1	n	nUN' is used to store the value to be copied in the last digit of the 'Discretionary Data' in <i>Track 1 Data</i> and <i>Track 2 Data</i>
Response Message Data Field	var. up to 256	b	TLV encoded string included in R-APDU of COMPUTE CRYPTOGRAPHIC CHECKSUM
q	1	n	Number of CVC3 digits to be copied in the 'Discretionary Data' in <i>Track 1 Data</i> and <i>Track 2 Data</i>
t	1	n	Number of ATC digits to be copied in the 'Discretionary Data' in <i>Track 1 Data</i> and <i>Track 2 Data</i>

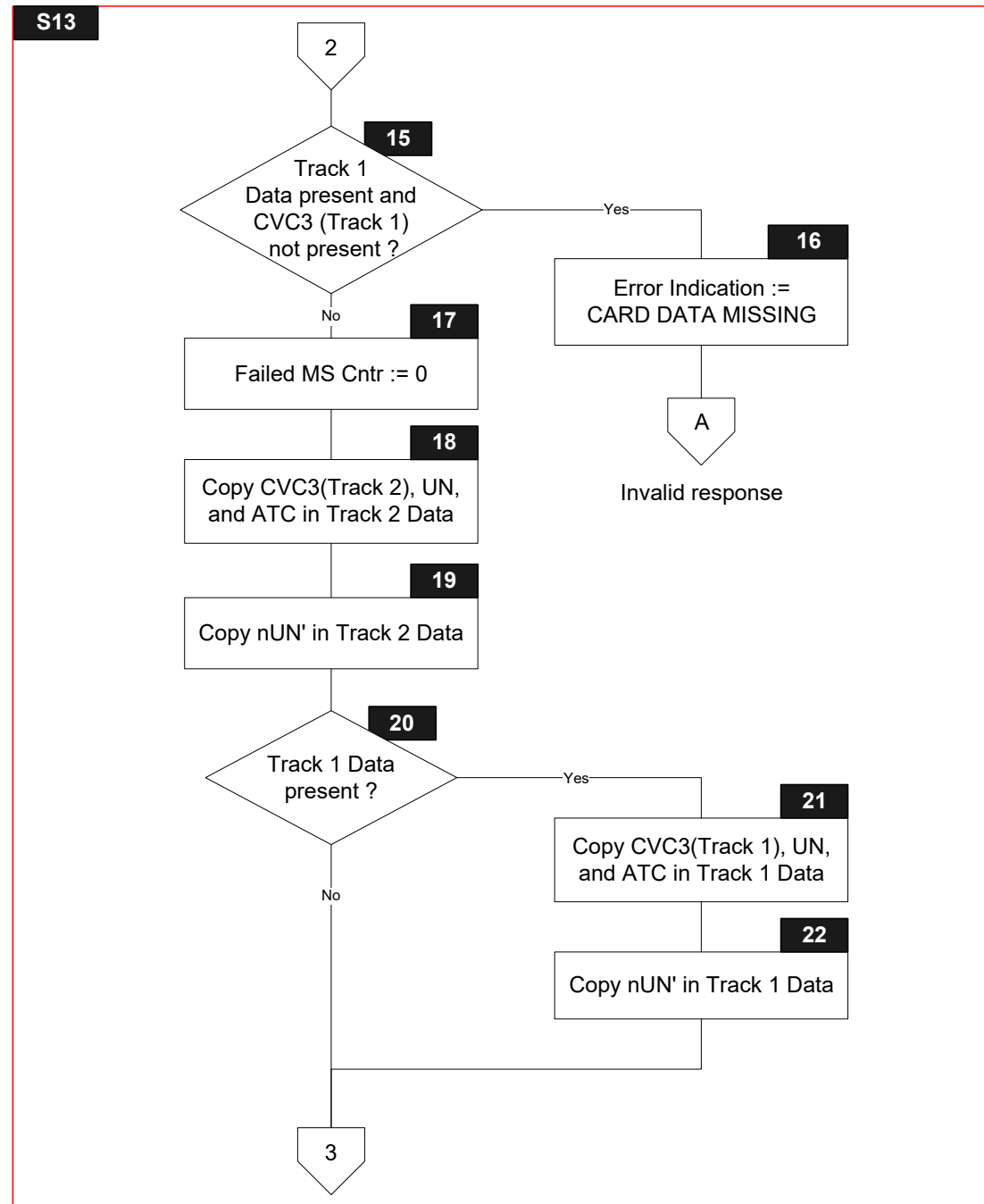
6.17.2 Flow Diagram

Figure 6.16 shows the flow diagram of s13 – waiting for CCC response – 1. Symbols in this diagram are labelled S13.X.

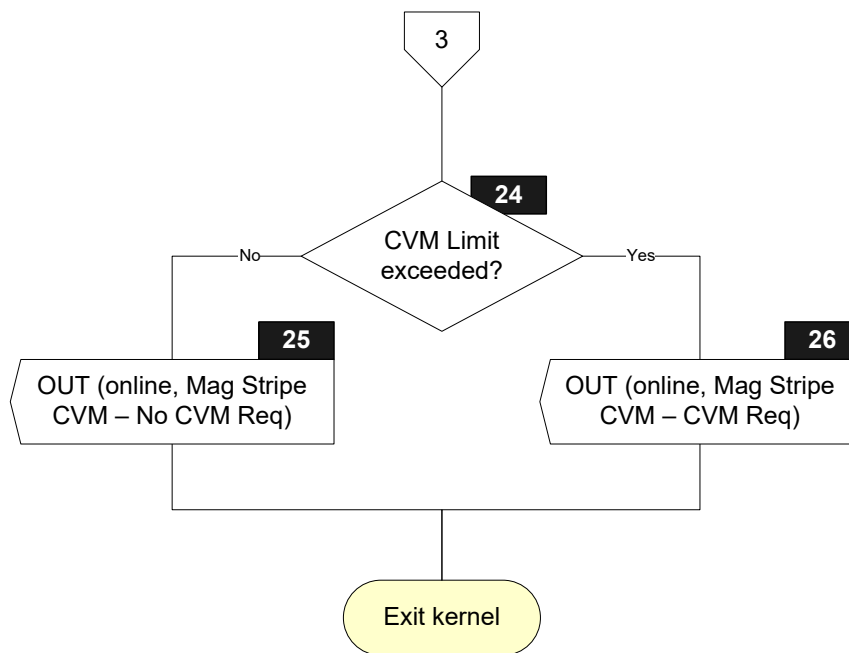
Figure 6.16—State 13 Flow Diagram



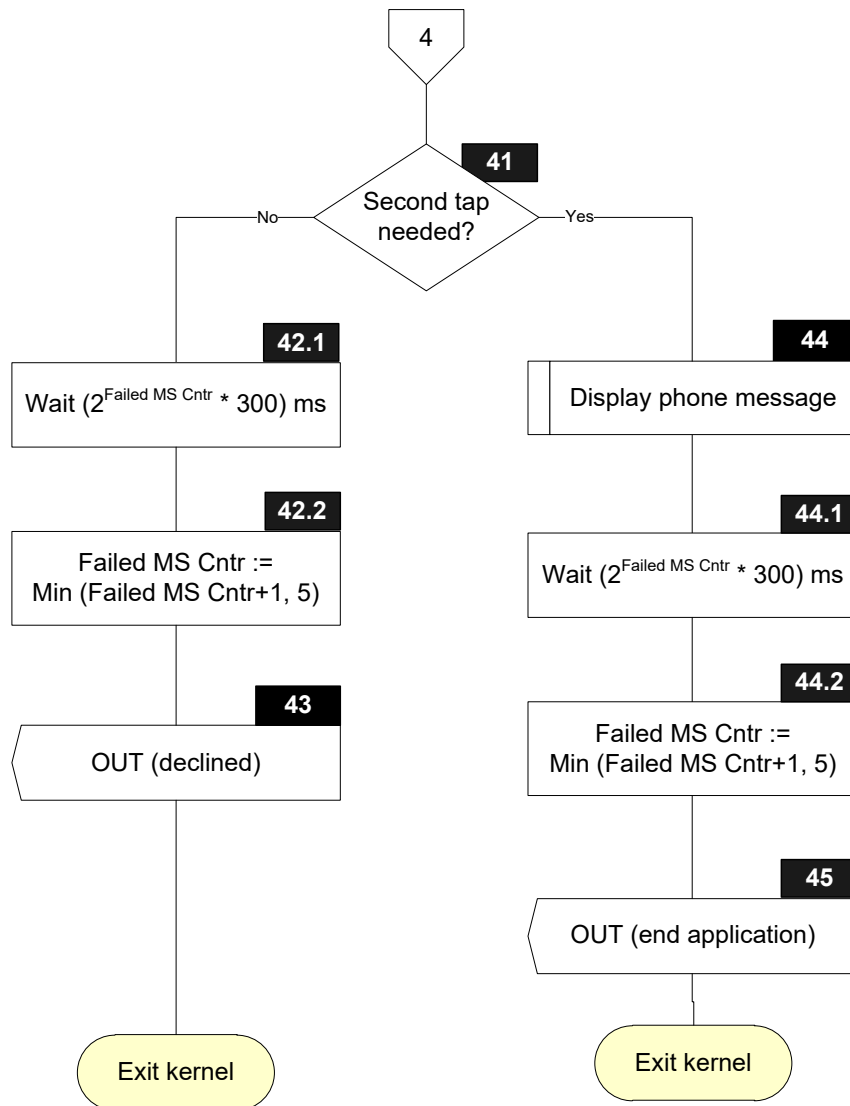




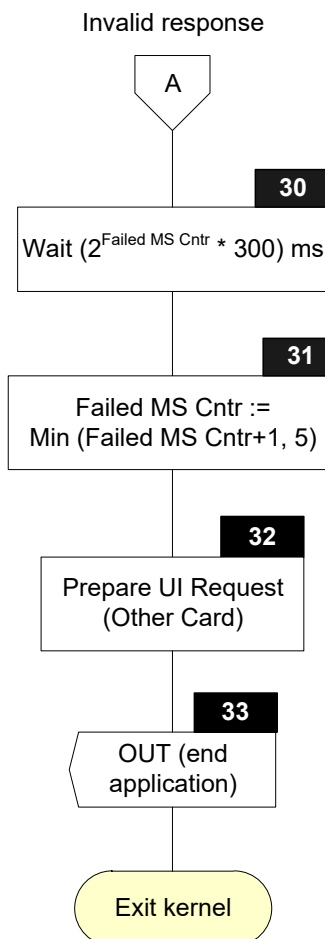
S13



S13



S13



6.17.3 Processing

S13.1

Receive L1RSP Signal with Return Code

S13.2

Wait for $(2^{\text{Failed MS Cntr}} * 300)$ ms

Note that *Failed MS Cntr* is stored in the scratch pad provided to the Kernel at instantiation. Dependent on the implementation, it may be that *Failed MS Cntr* does not exist the first time the Kernel is executed. In this case, *Failed MS Cntr* must be created and initialized to zero.

S13.3

Failed MS Cntr := min(*Failed MS Cntr* + 1, 5)

S13.4

'Message Identifier' in *User Interface Request Data* := TRY AGAIN

'Status' in *User Interface Request Data* := READY TO READ

'Hold Time' in *User Interface Request Data* := '000000'

S13.5

'Status' in *Outcome Parameter Set* := END APPLICATION

'Start' in *Outcome Parameter Set* := B

SET 'UI Request on Restart Present' in *Outcome Parameter Set*

'L1' in *Error Indication* := Return Code

'Msg On Error' in *Error Indication* := TRY AGAIN

CreateMSDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*))) Signal

S13.6

Receive RA Signal with Response Message Data Field and SW12

S13.8

Receive DET Signal

S13.9

IF [SW12 = '9000']

THEN

GOTO S13.11

ELSE

GOTO S13.10

ENDIF

S13.10

'L2' in *Error Indication* := STATUS BYTES

'SW12' in *Error Indication* := SW12

S13.11

IF [(Length of Response Message Data Field > 0) AND
(Response Message Data Field[1] = '77')]

THEN

Parsing Result := ParseAndStoreCardResponse(Response Message Data Field)

ELSE

Parsing Result := FALSE

ENDIF

S13.12

IF [Parsing Result]

THEN

GOTO S13.12.1

ELSE

GOTO S13.13

ENDIF

S13.12.1

'Message Identifier' in *User Interface Request Data* := CLEAR DISPLAY

'Status' in *User Interface Request Data* := CARD READ SUCCESSFULLY

'Hold Time' in *User Interface Request Data* := '000000'

Send MSG(*User Interface Request Data*) Signal

S13.13

'L2' in *Error Indication* := PARSING ERROR

S13.14.1

IF [IsEmpty(TagOf(*Application Transaction Counter*))]

THEN

GOTO S13.14.2

ELSE

GOTO S13.14.4

ENDIF

S13.14.2

```
IF      [IsNotEmpty(TagOf(CVC3 (Track2)))]  
THEN  
    GOTO S13.14.5  
ELSE  
    GOTO S13.14.3  
ENDIF
```

S13.14.3

```
IF      [IsNotEmpty(TagOf(POS Cardholder Interaction Information))]  
THEN  
    GOTO S13.41  
ELSE  
    GOTO S13.14.4  
ENDIF
```

S13.14.4

'L2' in *Error Indication* := CARD DATA MISSING

S13.14.5

```
IF      [IsNotEmpty(TagOf(POS Cardholder Interaction Information))]  
THEN  
    GOTO S13.14.6  
ELSE  
    GOTO S13.14.7  
ENDIF
```

S13.14.6

```
IF      ['OD-CVM verification successful' in POS Cardholder Interaction  
        Information is set]  
THEN  
    GOTO S13.14.8  
ELSE  
    GOTO S13.14.7  
ENDIF
```

S13.14.7

nUN' := nUN

S13.14.8

nUN' := (nUN + 5) modulo 10

S13.15

```
IF      [IsNotEmpty(TagOf(Track 1 Data))
        AND
        (IsNotPresent(TagOf(CVC3 (Track1))) OR
         IsEmpty(TagOf(CVC3 (Track1))))]
THEN
        GOTO S13.16
ELSE
        GOTO S13.17
ENDIF
```

S13.16

'L2' in *Error Indication* := CARD DATA MISSING

S13.17

Failed MS Cntr := 0

S13.18

q := Number of non-zero bits in *PCVC3(Track2)*
t := *NATC(Track2)*

Convert the binary encoded *CVC3 (Track2)* to the BCD encoding of the corresponding number expressed in base 10. Copy the q least significant digits of the BCD encoded *CVC3 (Track2)* in the eligible positions of the 'Discretionary Data' in *Track 2 Data*. The eligible positions are indicated by the q non-zero bits in *PCVC3(Track2)*.

Replace the *nUN* least significant eligible positions of the 'Discretionary Data' in *Track 2 Data* by the *nUN* least significant digits of *Unpredictable Number (Numeric)*. The eligible positions in the 'Discretionary Data' in *Track 2 Data* are indicated by the *nUN* least significant non-zero bits in *PUNATC(Track2)*.

If *t* ≠ 0, convert the *Application Transaction Counter* to the BCD encoding of the corresponding number expressed in base 10. Replace the t most significant eligible positions of the 'Discretionary Data' in *Track 2 Data* by the t least significant digits of the BCD encoded *Application Transaction Counter*. The eligible positions in the 'Discretionary Data' in *Track 2 Data* are indicated by the t most significant non-zero bits in *PUNATC(Track2)*.

S13.19

Copy *nUN* into the least significant digit of the 'Discretionary Data' in *Track 2 Data*

S13.20

```
IF      [IsNotEmpty(TagOf(Track 1 Data))]
THEN
    GOTO S13.21
ELSE
    GOTO S13.24
ENDIF
```

S13.21

q := Number of non-zero bits in *PCVC3(Track1)*
t := *NATC(Track1)*

Convert the binary encoded *CVC3 (Track1)* to the BCD encoding of the corresponding number expressed in base 10. Convert the q least significant digits of the BCD encoded *CVC3 (Track1)* into ASCII format and copy the q ASCII encoded *CVC3 (Track1)* characters into the eligible positions of the 'Discretionary Data' in *Track 1 Data*. The eligible positions are indicated by the q non-zero bits in *PCVC3(Track1)*.

Convert the BCD encoded *Unpredictable Number (Numeric)* into ASCII format and replace the *nUN* least significant eligible positions of the 'Discretionary Data' in *Track 1 Data* by the *nUN* least significant characters of the ASCII encoded *Unpredictable Number (Numeric)*. The eligible positions in the 'Discretionary Data' in *Track 1 Data* are indicated by the *nUN* least significant non-zero bits in *PUNATC(Track1)*.

If $t \neq 0$, convert the *Application Transaction Counter* to the BCD encoding of the corresponding number expressed in base 10. Convert the t least significant digits of the BCD encoded *Application Transaction Counter* into ASCII format. Replace the t most significant eligible positions of the 'Discretionary Data' in *Track 1 Data* by the t ASCII encoded *Application Transaction Counter* characters. The eligible positions in the 'Discretionary Data' in *Track 1 Data* are indicated by the t most significant non-zero bits in *PUNATC(Track1)*.

S13.22

Convert *nUN'* into the ASCII format

Copy the ASCII encoded *nUN'* character into the least significant position of the 'Discretionary Data' in *Track 1 Data*

S13.24

IF [Amount, Authorized (Numeric) > Reader CVM Required Limit]
THEN
 GOTO S13.26
ELSE
 GOTO S13.25
ENDIF

S13.25

'Status' in Outcome Parameter Set := ONLINE REQUEST
'CVM' in Outcome Parameter Set := 'CVM' in Mag-stripe CVM Capability – No CVM Required
IF ['CVM' in Mag-stripe CVM Capability – No CVM Required = OBTAIN SIGNATURE]
THEN
 'Receipt' in Outcome Parameter Set := YES
ENDIF
SET 'Data Record Present' in Outcome Parameter Set
CreateMSDataRecord ()
CreateMSDiscretionaryData ()
Send OUT(GetTLV(TagOf(Outcome Parameter Set)),
 GetTLV(TagOf(Data Record)),
 GetTLV(TagOf(Discretionary Data))) Signal

S13.26

'Status' in Outcome Parameter Set := ONLINE REQUEST
'CVM' in Outcome Parameter Set := 'CVM' in Mag-stripe CVM Capability – CVM Required
'Receipt' in Outcome Parameter Set := YES
SET 'Data Record Present' in Outcome Parameter Set
CreateMSDataRecord ()
CreateMSDiscretionaryData ()
Send OUT(GetTLV(TagOf(Outcome Parameter Set)),
 GetTLV(TagOf(Data Record)),
 GetTLV(TagOf(Discretionary Data))) Signal

S13.41

IF [POS Cardholder Interaction Information AND '00030F' ≠ '000000']
THEN
 GOTO S13.44
ELSE
 GOTO S13.42.1
ENDIF

S13.42.1

Wait for $(2^{\text{Failed MS Cntr}} * 300)$ ms

Note that *Failed MS Cntr* is stored in the scratch pad provided to the Kernel at instantiation. Dependent on the implementation, it may be that *Failed MS Cntr* does not exist the first time the Kernel is executed. In this case, *Failed MS Cntr* must be created and initialized to zero.

S13.42.2

Failed MS Cntr := min(*Failed MS Cntr* + 1, 5)

S13.43

'Hold Time' in *User Interface Request Data* := Message Hold Time

'Message Identifier' in *User Interface Request Data* := DECLINED

'Status' in *User Interface Request Data* := NOT READY

'Status' in *Outcome Parameter Set* := DECLINED

SET 'Data Record Present' in *Outcome Parameter Set*

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

CreateMSDiscretionaryData ()

CreateMSDataRecord ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Data Record*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*))) Signal

S13.44

FOR every entry in the *Phone Message Table*

{

IF [(PCII Mask AND
POS Cardholder Interaction Information) = PCII Value]

THEN

'Message Identifier' in *User Interface Request Data* := Message Identifier

'Status' in *User Interface Request Data* := Status

Send MSG(*User Interface Request Data*) Signal

EXIT loop

ENDIF

}

S13.44.1

Wait for $(2^{\text{Failed MS Cntr}} * 300)$ ms

Note that *Failed MS Cntr* is stored in the scratch pad provided to the Kernel at instantiation. Dependent on the implementation, it may be that *Failed MS Cntr* does not exist the first time the Kernel is executed. In this case, *Failed MS Cntr* must be created and initialized to zero.

S13.44.2

Failed MS Cntr := min(*Failed MS Cntr* + 1, 5)

S13.45

'Hold Time' in *User Interface Request Data* := '000000'

'Status' in *User Interface Request Data* := READY TO READ

SET 'UI Request on Restart Present' in *Outcome Parameter Set*

'Status' in *Outcome Parameter Set* := END APPLICATION

'Start' in *Outcome Parameter Set* := B

SET 'Data Record Present' in *Outcome Parameter Set*

CreateMSDataRecord ()

CreateMSDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Data Record*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*))) Signal

Invalid Response

S13.30

Wait for $(2^{\text{Failed MS Cntr}} * 300)$ ms

Note that *Failed MS Cntr* is stored in the scratch pad provided to the Kernel at instantiation. Dependent on the implementation, it may be that *Failed MS Cntr* does not exist the first time the Kernel is executed. In this case, *Failed MS Cntr* must be created and initialized to zero.

S13.31

Failed MS Cntr := min(*Failed MS Cntr* + 1, 5)

S13.32

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

'Hold Time' in *User Interface Request Data* := Message Hold Time

S13.33

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := 'Message Identifier' in *User Interface Request Data*

CreateMSDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*))) Signal

6.18 State 14 – Waiting for CCC Response – 2

State 14 is a state specific to mag-stripe mode and is only implemented for the MAG Implementation Option.

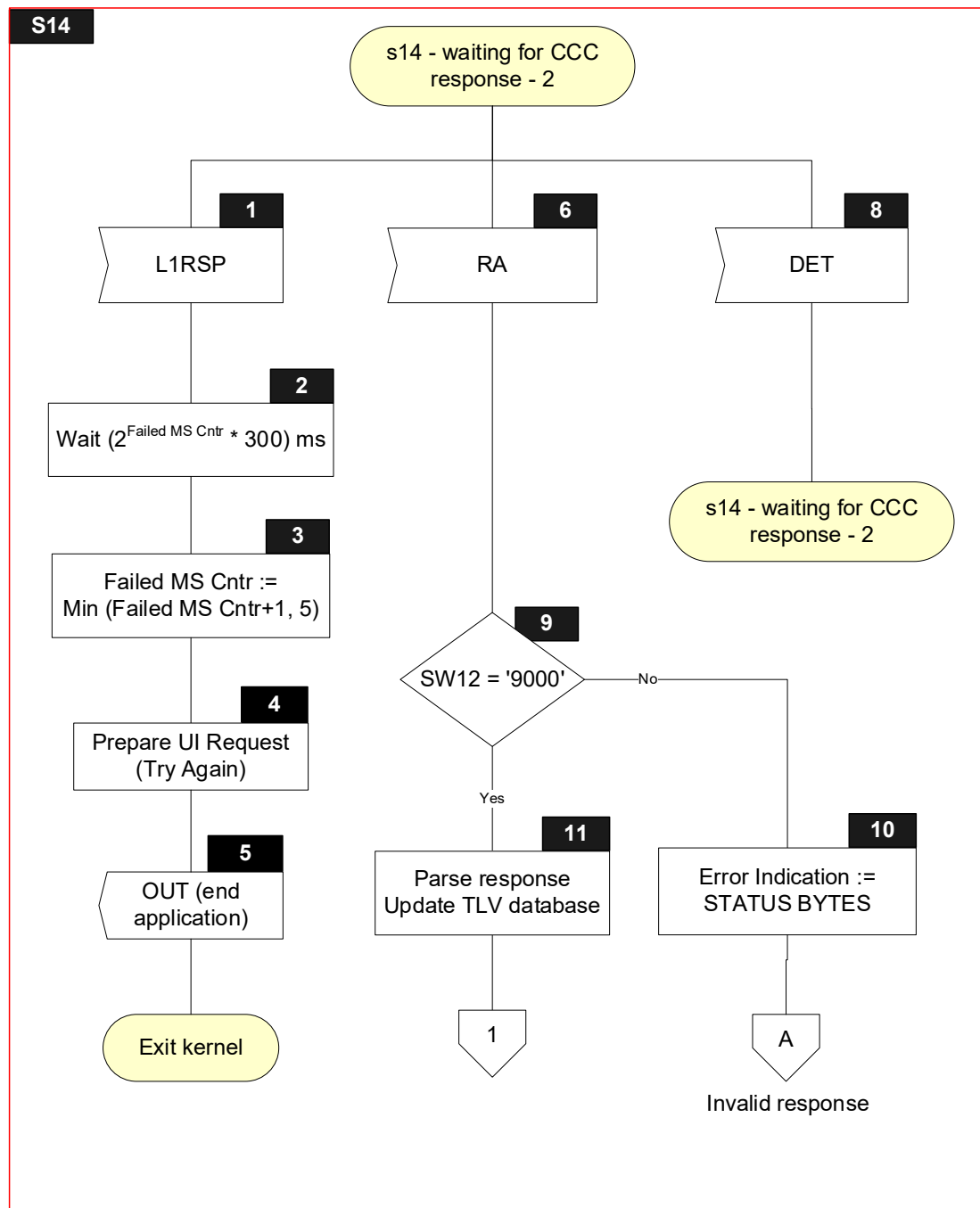
6.18.1 Local Variables

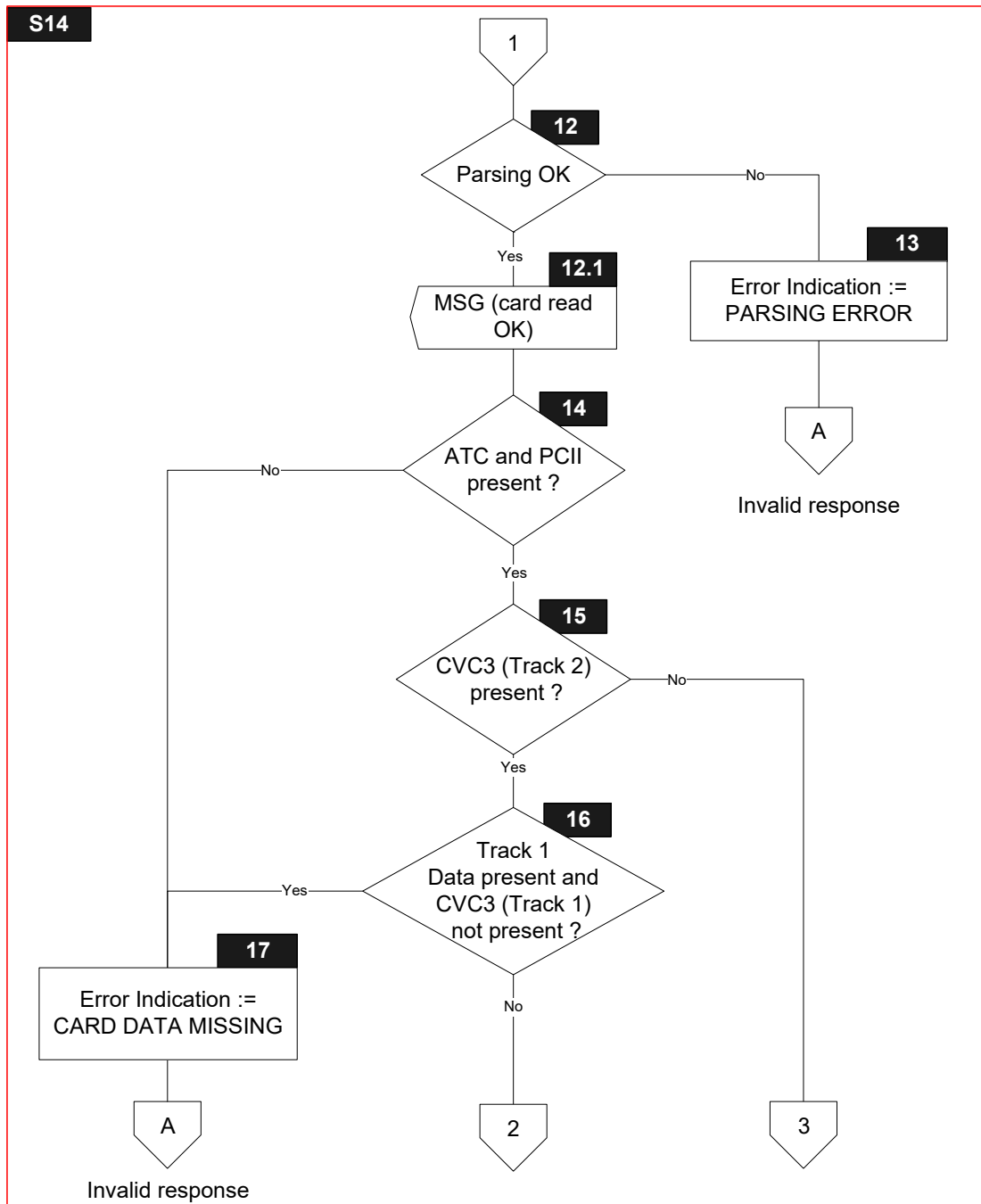
Name	Length	Format	Description
Return Code	1	b	Value returned with L1RSP Signal (TIME OUT ERROR, PROTOCOL ERROR, TRANSMISSION ERROR)
Parsing Result	1	b	Boolean used to store result of parsing a TLV string
SW12	2	b	Status bytes
Response Message Data Field	var. up to 256	b	TLV encoded string included in R-APDU of COMPUTE CRYPTOGRAPHIC CHECKSUM
nUN'	1	n	nUN' is used to store the value to be copied in the last digit of the 'Discretionary Data' in <i>Track 1 Data</i> and <i>Track 2 Data</i>
q	1	n	Number of CVC3 digits to be copied in the 'Discretionary Data' in <i>Track 1 Data</i> and <i>Track 2 Data</i>
t	1	n	Number of ATC digits to be copied in the 'Discretionary Data' in <i>Track 1 Data</i> and <i>Track 2 Data</i>

6.18.2 Flow Diagram

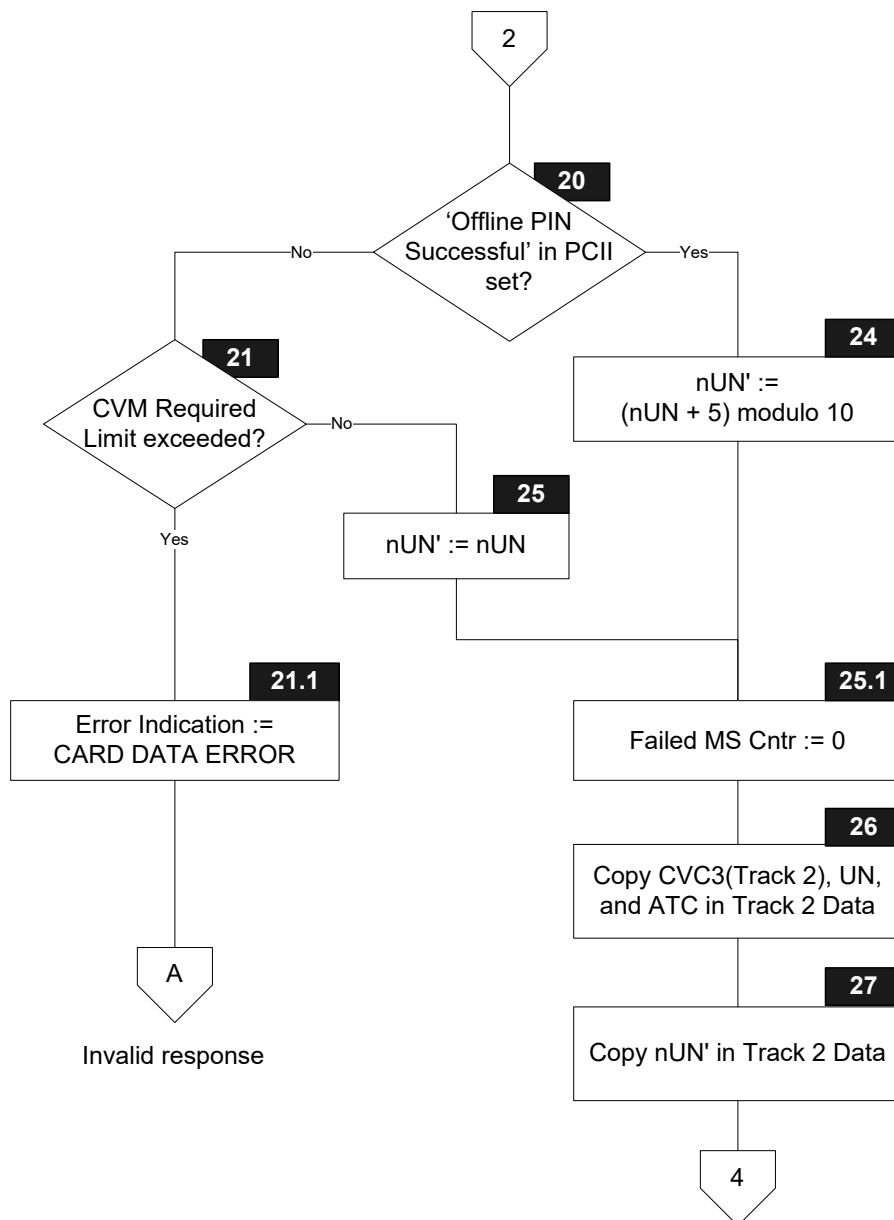
Figure 6.17 shows the flow diagram of s14 – waiting for CCC response – 2. Symbols in this diagram are labelled S14.X.

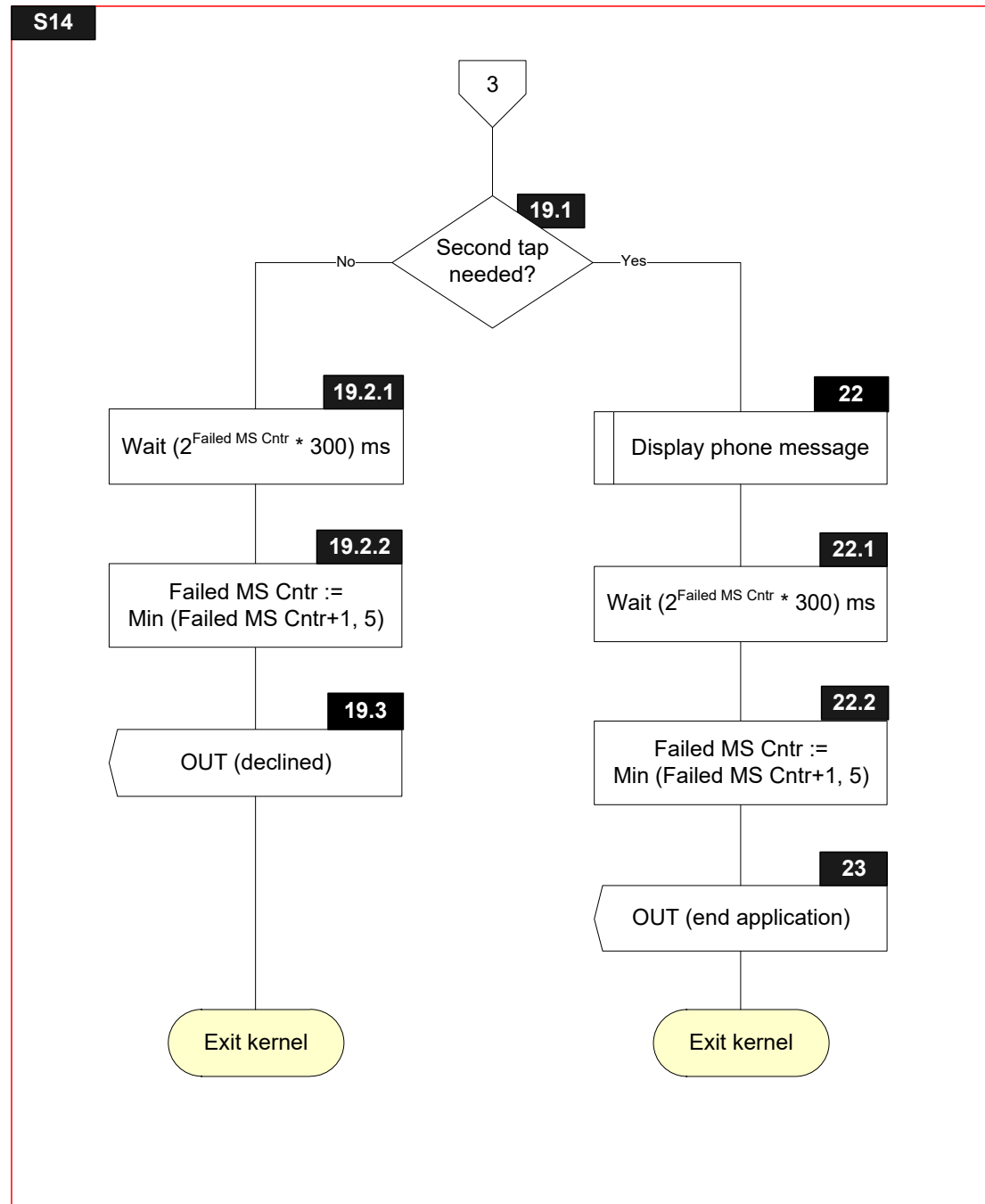
Figure 6.17—State 14 Flow Diagram



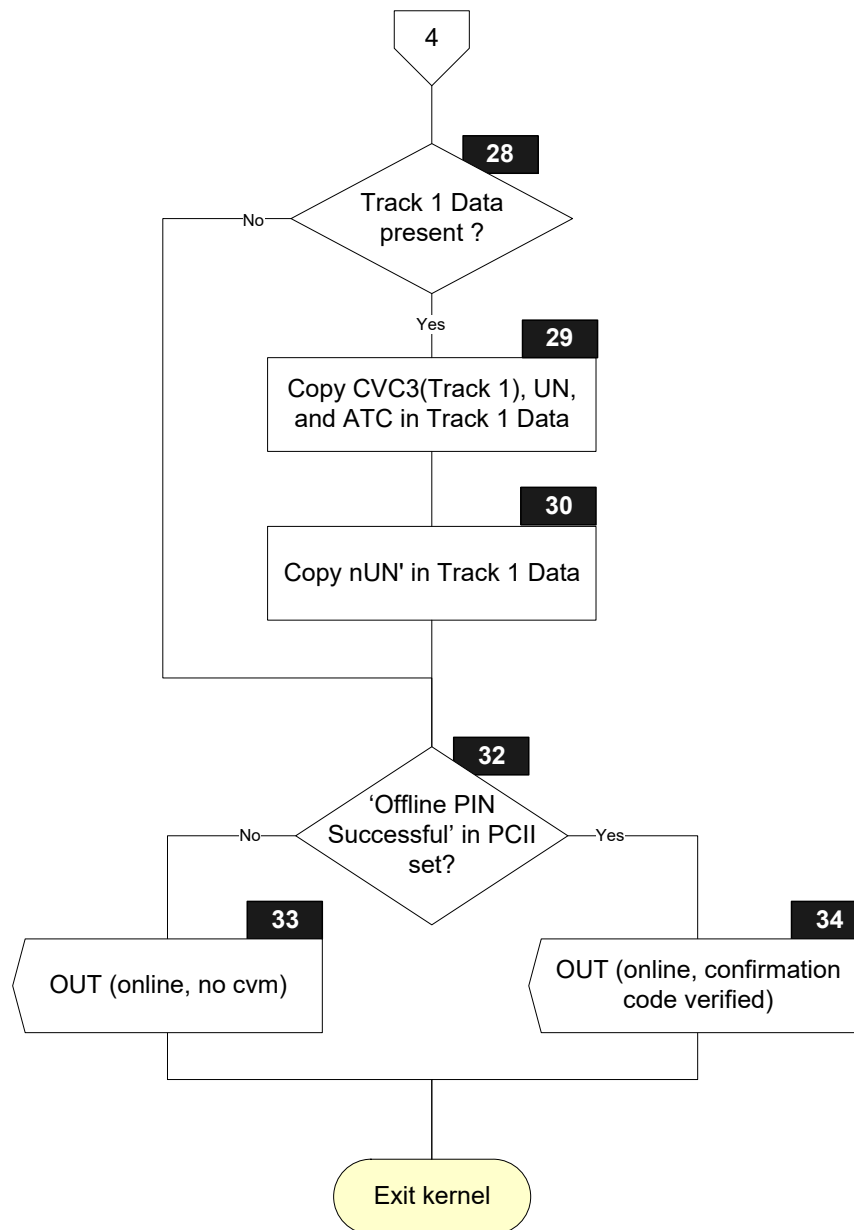


S14

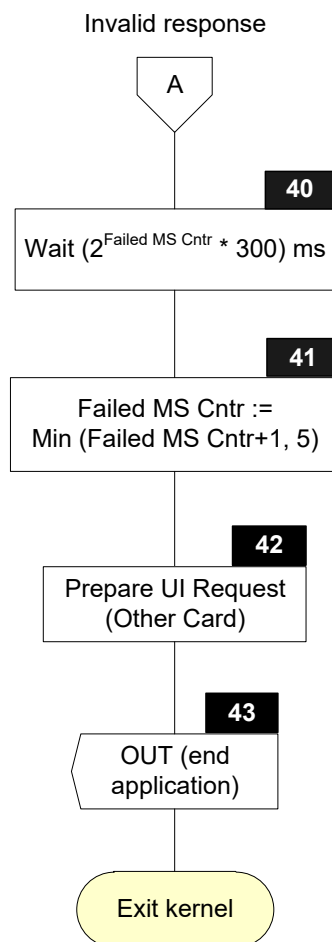




S14



S14



6.18.3 Processing

S14.1

Receive L1RSP Signal with Return Code

S14.2

Wait for $(2^{\text{Failed MS Cntr}} * 300)$ ms

Note that *Failed MS Cntr* is stored in the scratch pad provided to the Kernel at instantiation. Dependent on the implementation, it may be that *Failed MS Cntr* does not exist the first time the Kernel is executed. In this case, *Failed MS Cntr* must be created and initialized to zero.

S14.3

Failed MS Cntr := min(*Failed MS Cntr* + 1, 5)

S14.4

'Message Identifier' in *User Interface Request Data* := TRY AGAIN

'Status' in *User Interface Request Data* := READY TO READ

'Hold Time' in *User Interface Request Data* := '000000'

S14.5

'Status' in *Outcome Parameter Set* := END APPLICATION

'Start' in *Outcome Parameter Set* := B

SET 'UI Request on Restart Present' in *Outcome Parameter Set*

'L1' in *Error Indication* := Return Code

'Msg On Error' in *Error Indication* := TRY AGAIN

CreateMSDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*))) Signal

S14.6

Receive RA Signal with Response Message Data Field and SW12

S14.8

Receive DET Signal

S14.9

IF [SW12 = '9000']

THEN

GOTO S14.11

ELSE

GOTO S14.10

ENDIF

S14.10

'L2' in *Error Indication* := STATUS BYTES

'SW12' in *Error Indication* := SW12

S14.11

IF [(Length of Response Message Data Field > 0) AND
(Response Message Data Field[1] = '77')]

THEN

Parsing Result := ParseAndStoreCardResponse(Response Message Data Field)

ELSE

Parsing Result := FALSE

ENDIF

S14.12

IF [Parsing Result]

THEN

GOTO S14.12.1

ELSE

GOTO S14.13

ENDIF

S14.12.1

'Message Identifier' in *User Interface Request Data* := CLEAR DISPLAY

'Status' in *User Interface Request Data* := CARD READ SUCCESSFULLY

'Hold Time' in *User Interface Request Data* := '000000'

Send MSG(*User Interface Request Data*) Signal

S14.13

'L2' in *Error Indication* := PARSING ERROR

S14.14

IF [IsEmpty(TagOf(*Application Transaction Counter*)) AND
IsEmpty(TagOf(*POS Cardholder Interaction Information*))]

THEN

GOTO S14.15

ELSE

GOTO S14.17

ENDIF

S14.15

```
IF      [IsNotEmpty(TagOf(CVC3 (Track2)))]  
THEN  
    GOTO S14.16  
ELSE  
    GOTO S14.19.1  
ENDIF
```

S14.16

```
IF      [IsNotEmpty(TagOf(Track 1 Data))  
        AND  
        (IsNotPresent(TagOf(CVC3 (Track1))) OR  
         IsEmpty(TagOf(CVC3 (Track1))))]  
THEN  
    GOTO S14.17  
ELSE  
    GOTO S14.20  
ENDIF
```

S14.17

'L2' in *Error Indication* := CARD DATA MISSING

S14.19.1

```
IF [POS Cardholder Interaction Information AND '00030F' ≠ '000000']  
THEN  
    GOTO S14.22  
ELSE  
    GOTO S14.19.2.1  
ENDIF
```

S14.19.2.1

Wait for ($2^{\text{Failed MS Cntr}} * 300$) ms

Note that *Failed MS Cntr* is stored in the scratch pad provided to the Kernel at instantiation. Dependent on the implementation, it may be that *Failed MS Cntr* does not exist the first time the Kernel is executed. In this case, *Failed MS Cntr* must be created and initialized to zero.

S14.19.2.2

Failed MS Cntr := min(*Failed MS Cntr* + 1, 5)

S14.19.3

'Hold Time' in *User Interface Request Data* := Message Hold Time
'Message Identifier' in *User Interface Request Data* := DECLINED
'Status' in *User Interface Request Data* := NOT READY

'Status' in *Outcome Parameter Set* := DECLINED
SET 'Data Record Present' in *Outcome Parameter Set*
SET 'UI Request on Outcome Present' in *Outcome Parameter Set*
CreateMSDiscretionaryData ()
CreateMSDataRecord ()
Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Data Record*)),
 GetTLV(TagOf(*Discretionary Data*)) ,
 GetTLV(TagOf(*User Interface Request Data*)))) Signal

S14.20

IF ['OD-CVM verification successful' in *POS Cardholder Interaction Information* is set]
THEN
 GOTO S14.24
ELSE
 GOTO S14.21
ENDIF

S14.21

IF [*Amount, Authorized (Numeric) > Reader CVM Required Limit*]
THEN
 GOTO S14.21.1
ELSE
 GOTO S14.25
ENDIF

S14.21.1

'L2' in *Error Indication* := CARD DATA ERROR

S14.22

FOR every entry in the *Phone Message Table*
{
 IF [(PCII Mask AND
 POS Cardholder Interaction Information) = PCII Value]
 THEN
 'Message Identifier' in *User Interface Request Data* := Message Identifier
 'Status' in *User Interface Request Data* := Status
 Send MSG(*User Interface Request Data*) Signal
 EXIT loop
 ENDIF
}

S14.22.1

Wait for $(2^{\text{Failed MS Cntr}} * 300)$ ms

Note that *Failed MS Cntr* is stored in the scratch pad provided to the Kernel at instantiation. Dependent on the implementation, it may be that *Failed MS Cntr* does not exist the first time the Kernel is executed. In this case, *Failed MS Cntr* must be created and initialized to zero.

S14.22.2

Failed MS Cntr := min(*Failed MS Cntr* + 1, 5)

S14.23

'Hold Time' in *User Interface Request Data* := '000000'

'Status' in *User Interface Request Data* := READY TO READ

SET 'UI Request on Restart Present' in *Outcome Parameter Set*

'Status' in *Outcome Parameter Set* := END APPLICATION

'Start' in *Outcome Parameter Set* := B

SET 'Data Record Present' in *Outcome Parameter Set*

CreateMSDataRecord ()

CreateMSDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Data Record*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*))) Signal

S14.24

nUN' := (nUN + 5) modulo 10

S14.25

nUN' := nUN

S14.25.1

Failed MS Cntr := 0

S14.26

q := Number of non-zero bits in *PCVC3(Track2)*

t := *NATC(Track2)*

Convert the binary encoded *CVC3 (Track2)* to the BCD encoding of the corresponding number expressed in base 10. Copy the q least significant digits of the BCD encoded *CVC3 (Track2)* in the eligible positions of the 'Discretionary Data' in *Track 2 Data*. The eligible positions are indicated by the q non-zero bits in *PCVC3(Track2)*.

Replace the *nUN* least significant eligible positions of the 'Discretionary Data' in *Track 2 Data* by the *nUN* least significant digits of *Unpredictable Number (Numeric)*. The eligible positions in the 'Discretionary Data' in *Track 2 Data* are indicated by the *nUN* least significant non-zero bits in *PUNATC(Track2)*.

If $t \neq 0$, convert the *Application Transaction Counter* to the BCD encoding of the corresponding number expressed in base 10. Replace the t most significant eligible positions of the 'Discretionary Data' in *Track 2 Data* by the t least significant digits of the BCD encoded *Application Transaction Counter*. The eligible positions in the 'Discretionary Data' in *Track 2 Data* are indicated by the t most significant non-zero bits in *PUNATC(Track2)*.

S14.27

Copy *nUN* into the least significant digit of the 'Discretionary Data' in *Track 2 Data*

S14.28

IF [IsEmpty(TagOf(*Track 1 Data*))]

THEN

GOTO S14.29

ELSE

GOTO S14.32

ENDIF

S14.29

q := Number of non-zero bits in *PCVC3(Track1)*

t := *NATC(Track1)*

Convert the binary encoded *CVC3 (Track1)* to the BCD encoding of the corresponding number expressed in base 10. Convert the q least significant digits of the BCD encoded *CVC3 (Track1)* into the ASCII format and copy the q ASCII encoded *CVC3 (Track1)* characters into the eligible positions of the 'Discretionary Data' in *Track 1 Data*. The eligible positions are indicated by the q non-zero bits in *PCVC3(Track1)*.

Convert the BCD encoded *Unpredictable Number (Numeric)* into the ASCII format and replace the nUN least significant eligible positions of the 'Discretionary Data' in *Track 1 Data* by the nUN least significant characters of the ASCII encoded *Unpredictable Number (Numeric)*. The eligible positions in the 'Discretionary Data' in *Track 1 Data* are indicated by the nUN least significant non-zero bits in *PUNATC(Track1)*.

If t ≠ 0, convert the *Application Transaction Counter* to the BCD encoding of the corresponding number expressed in base 10. Convert the t least significant digits of the *Application Transaction Counter* into the ASCII format. Replace the t most significant eligible positions of the 'Discretionary Data' in *Track 1 Data* by the t ASCII encoded *Application Transaction Counter* characters. The eligible positions in the 'Discretionary Data' in *Track 1 Data* are indicated by the t most significant non-zero bits in *PUNATC(Track1)*.

S14.30

Convert nUN' into the ASCII format

Copy the ASCII encoded nUN' character into the least significant position of the 'Discretionary Data' in *Track 1 Data*

S14.32

IF ['OD-CVM verification successful' in *POS Cardholder Interaction Information* is set]

THEN

GOTO S14.34

ELSE

GOTO S14.33

ENDIF

S14.33

'Status' in *Outcome Parameter Set* := ONLINE REQUEST

'CVM' in *Outcome Parameter Set* := NO CVM

SET 'Data Record Present' in *Outcome Parameter Set*

CreateMSDataRecord ()

CreateMSDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Data Record*)),
 GetTLV(TagOf(*Discretionary Data*))) Signal

S14.34

'Status' in *Outcome Parameter Set* := ONLINE REQUEST

'CVM' in *Outcome Parameter Set* := CONFIRMATION CODE VERIFIED

IF [Amount, Authorized (Numeric) > Reader CVM Required Limit]

THEN

 'Receipt' in *Outcome Parameter Set* := YES

ENDIF

SET 'Data Record Present' in *Outcome Parameter Set*

CreateMSDataRecord ()

CreateMSDiscretionaryData ()

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Data Record*)),
 GetTLV(TagOf(*Discretionary Data*))) Signal

Invalid Response

S14.40

Wait for $(2^{\text{Failed MS Cntr}} * 300)$ ms

Note that *Failed MS Cntr* is stored in the scratch pad provided to the Kernel at instantiation. Dependent on the implementation, it may be that *Failed MS Cntr* does not exist the first time the Kernel is executed. In this case, *Failed MS Cntr* must be created and initialized to zero.

S14.41

Failed MS Cntr := min(*Failed MS Cntr* + 1, 5)

S14.42

'Message Identifier' in *User Interface Request Data* := ERROR – OTHER CARD

'Status' in *User Interface Request Data* := NOT READY

'Hold Time' in *User Interface Request Data* := Message Hold Time

S14.43

'Status' in *Outcome Parameter Set* := END APPLICATION

'Msg On Error' in *Error Indication* := 'Message Identifier' in *User Interface Request Data*

CreateMSDiscretionaryData ()

SET 'UI Request on Outcome Present' in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
GetTLV(TagOf(*Discretionary Data*)),
GetTLV(TagOf(*User Interface Request Data*))) Signal

6.19 State 15 – Waiting for Put Data Response After Generate AC

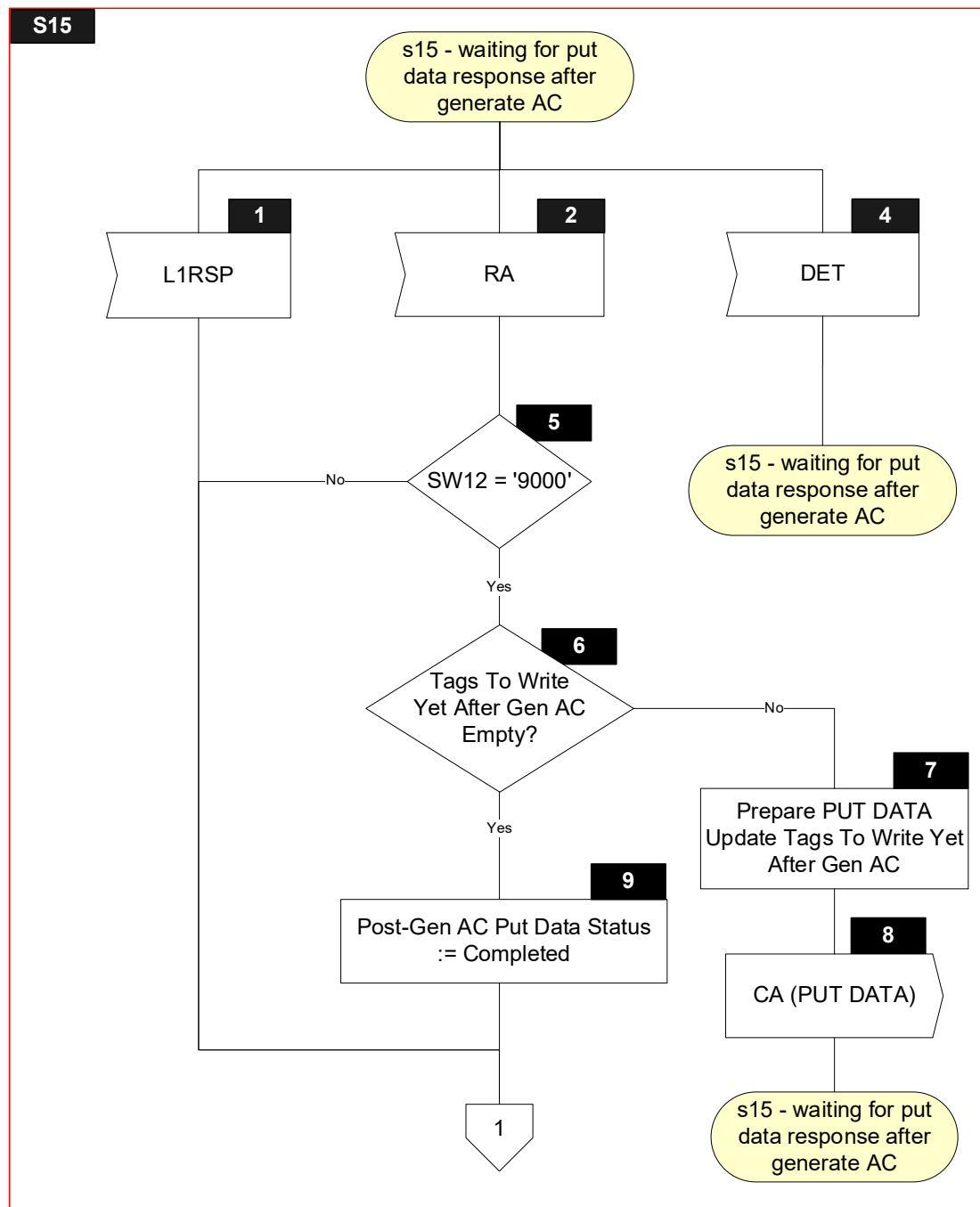
State 15 is a state specific to SDS processing and is only implemented for the DE/DS Implementation Option.

6.19.1 Local Variables

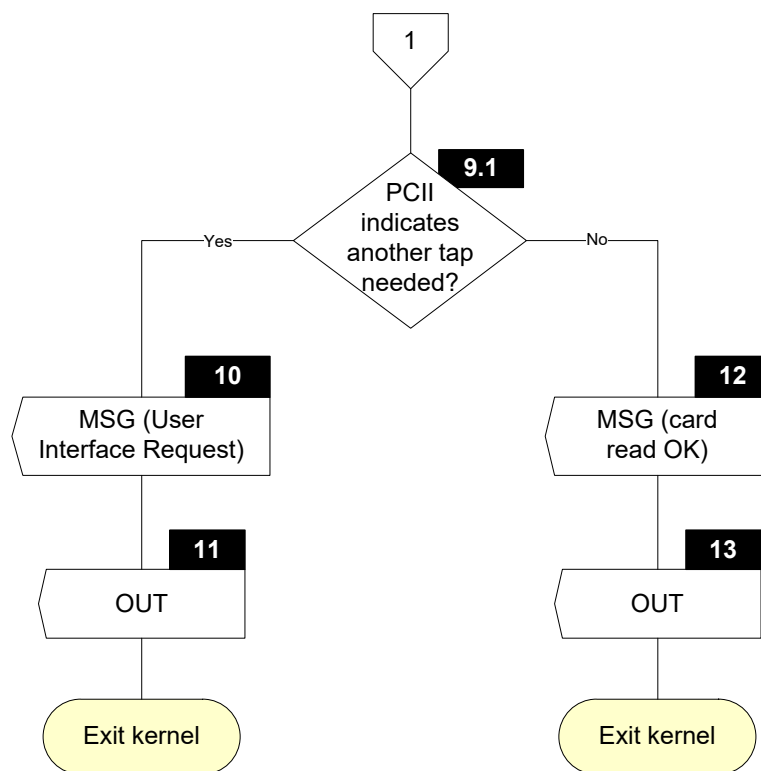
Name	Length	Format	Description
SW12	2	b	Status bytes
T	var.	b	Tag of TLV encoded string
L	var.	b	Length of TLV encoded string
V	var. up to 252	b	Value of TLV encoded string
Tmp UI Request Data	22	b	Local User Interface Request Data to clear display and change status to CARD READ SUCCESSFULLY

6.19.2 Flow Diagram

Figure 6.14 shows the flow diagram of s15 – waiting for put data response after generate AC. Symbols in this diagram are labelled S15.X.

Figure 6.18—State 15 Flow Diagram

S15



6.19.3 Processing

S15.1

Receive L1RSP Signal

S15.2

Receive RA Signal with SW12

S15.4

Receive DET Signal

S15.5

IF [SW12 = '9000']

THEN

GOTO S15.6

ELSE

GOTO S15.10

ENDIF

S15.6

IF [IsEmptyList(*Tags To Write Yet After Gen AC*)]

THEN

GOTO S15.9

ELSE

GOTO S15.7

ENDIF

S15.7

TLV := GetAndRemoveFromList(*Tags To Write Yet After Gen AC*)

Prepare PUT DATA command for TLV as specified in section 5.7

S15.8

Send CA(PUT DATA command) Signal

S15.9

SET 'Completed' in *Post-Gen AC Put Data Status*

S15.9.1

IF [IsEmpty(TagOf(*POS Cardholder Interaction Information*)) AND
(*POS Cardholder Interaction Information* AND '00030F' ≠ '000000')]

THEN

GOTO S15.10

ELSE

GOTO S15.12

ENDIF

S15.10

'Status' in *User Interface Request Data* := CARD READ SUCCESSFULLY
Send MSG(*User Interface Request Data*) Signal

S15.11

CreateEMVDiscretionaryData ()
SET 'UI Request on Restart Present' in *Outcome Parameter Set*
'Status' in *User Interface Request Data* := READY TO READ
'Hold Time' in *User Interface Request Data* := '000000'
Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Data Record*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*)))) Signal

S15.12

Tmp UI Request Data := '0000 ... 00'
'Message Identifier' in Tmp UI Request Data:= CLEAR DISPLAY
'Status' in Tmp UI Request Data:= CARD READ SUCCESSFULLY
'Hold Time' in Tmp UI Request Data:= '000000'
Send MSG(Tmp UI Request Data) Signal

S15.13

CreateEMVDiscretionaryData ()
SET 'UI Request on Outcome Present' in *Outcome Parameter Set*
Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Data Record*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*)))) Signal

7 Procedures

7.1 Procedure – CVM Selection

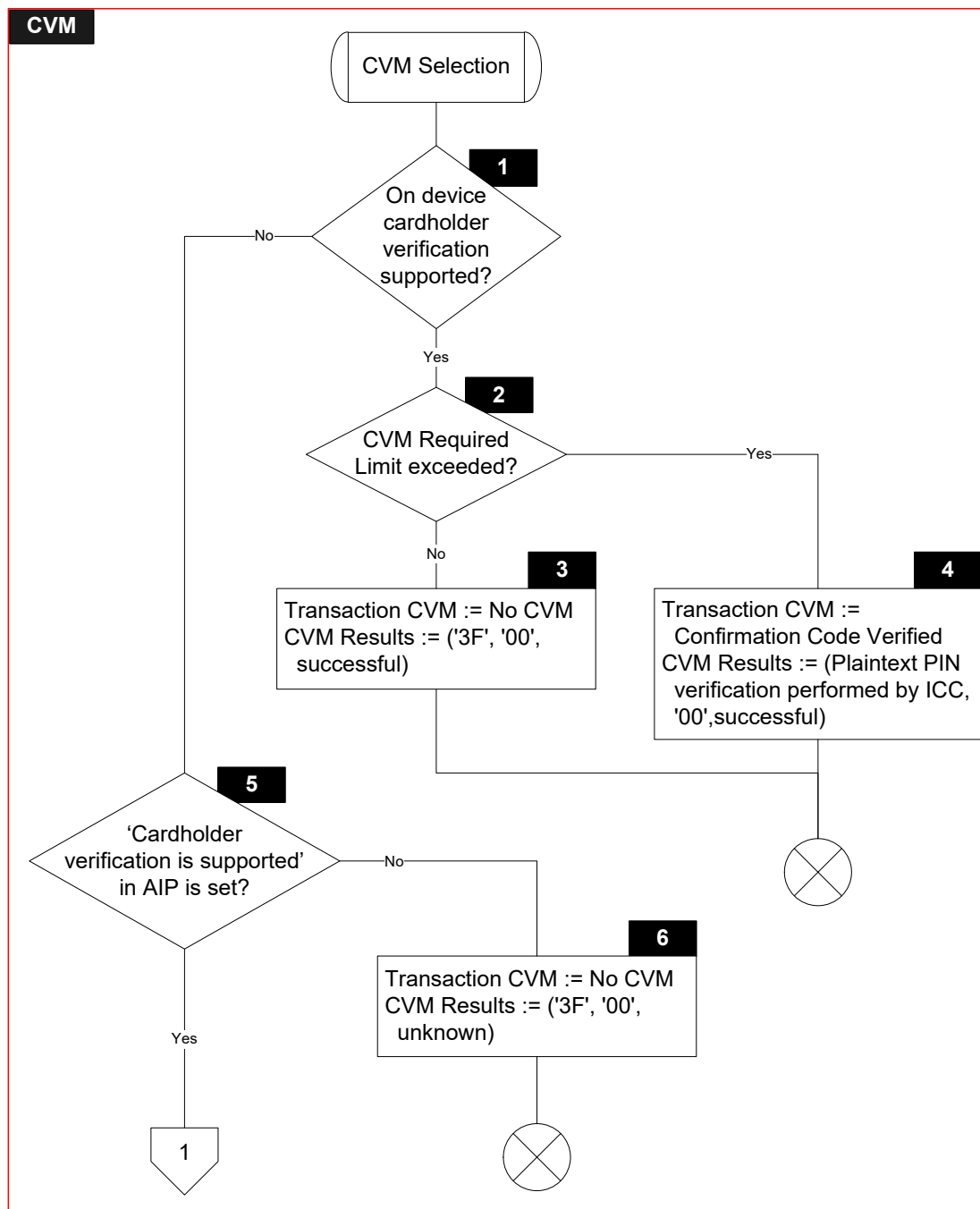
7.1.1 Local Variables

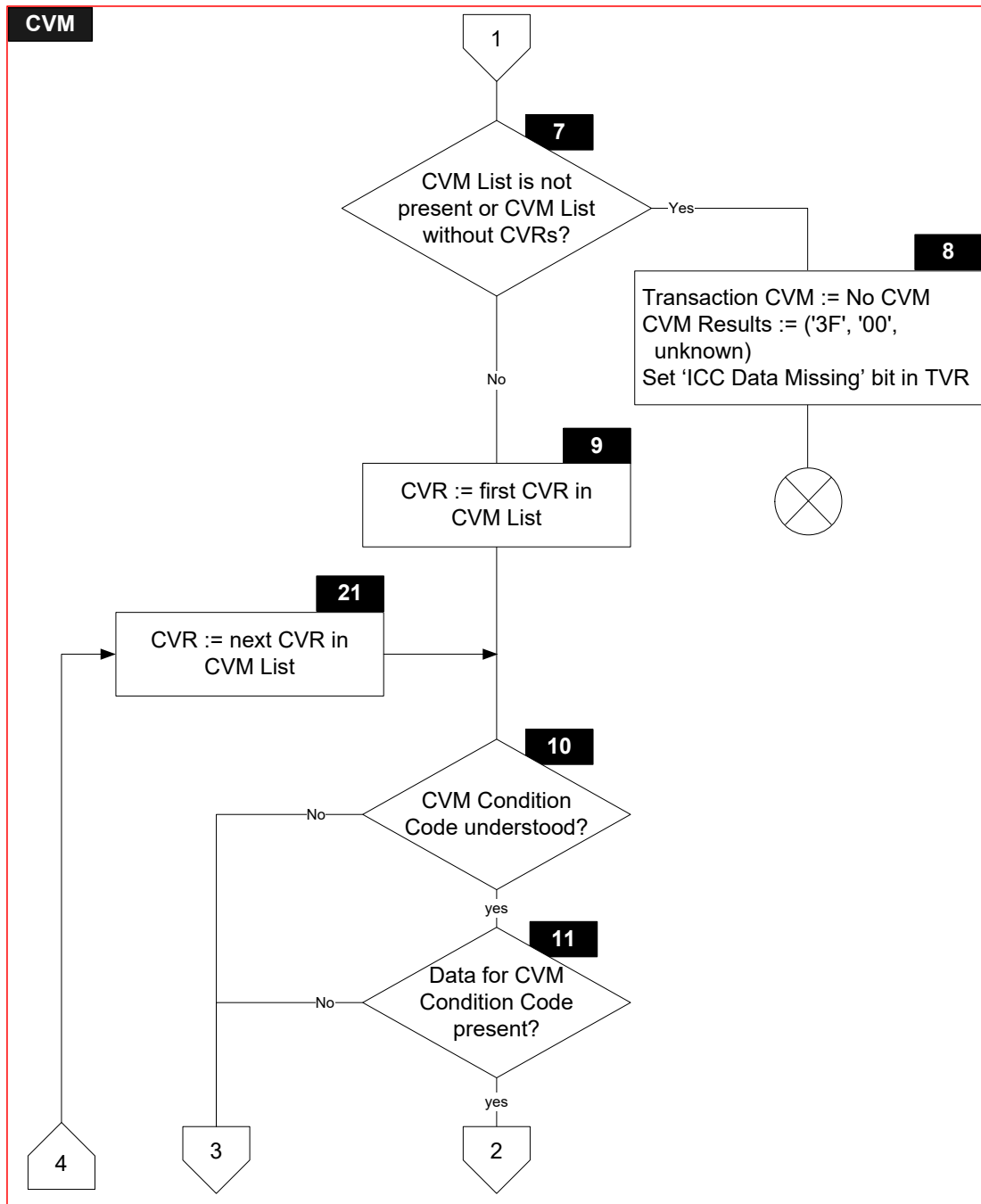
Name	Length	Format	Description
CVR	2	b	Cardholder Verification Rule
CVM Condition Code	1	b	Second byte of a CVR
CVM Code	1	b	First byte of a CVR

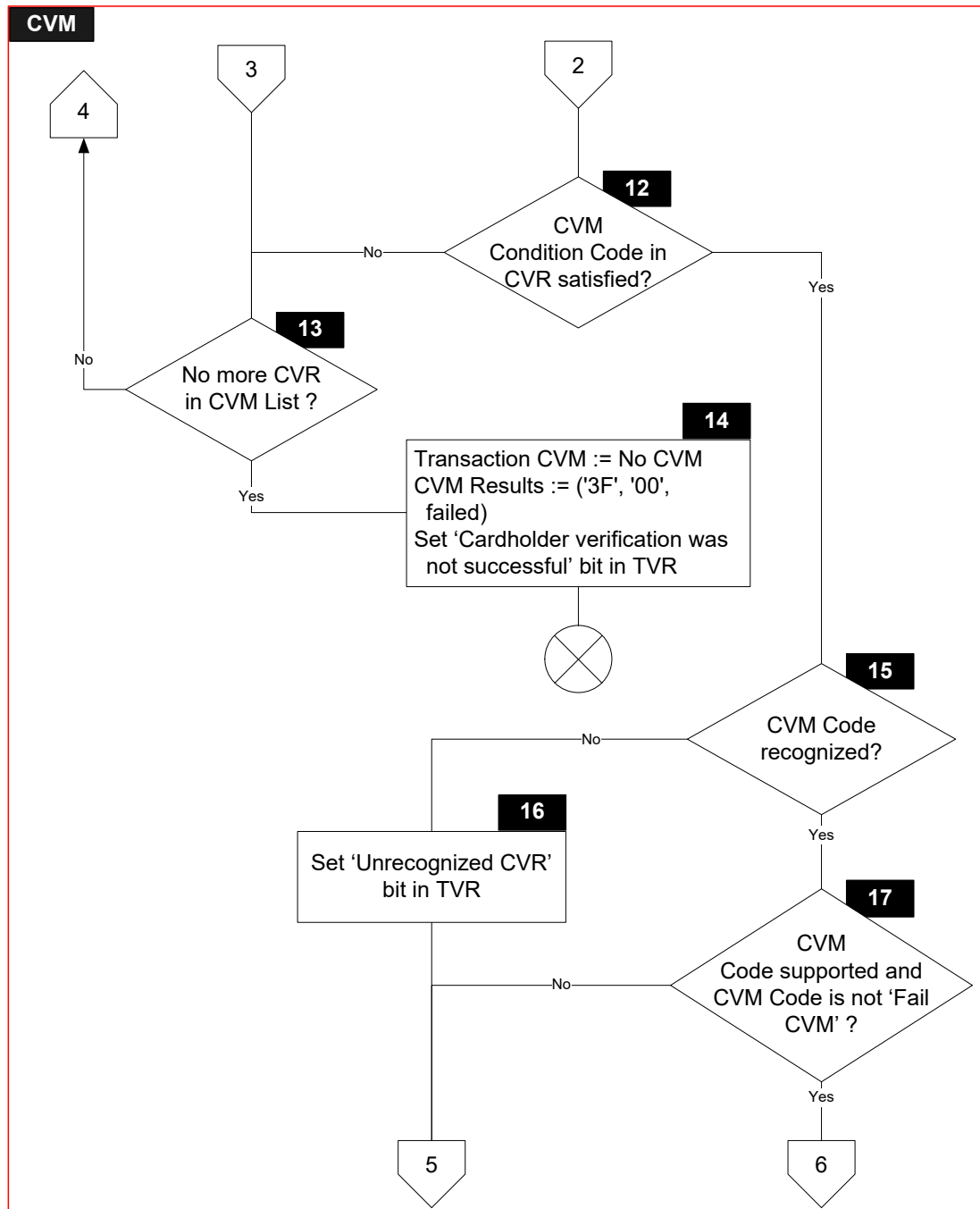
7.1.2 Flow Diagram

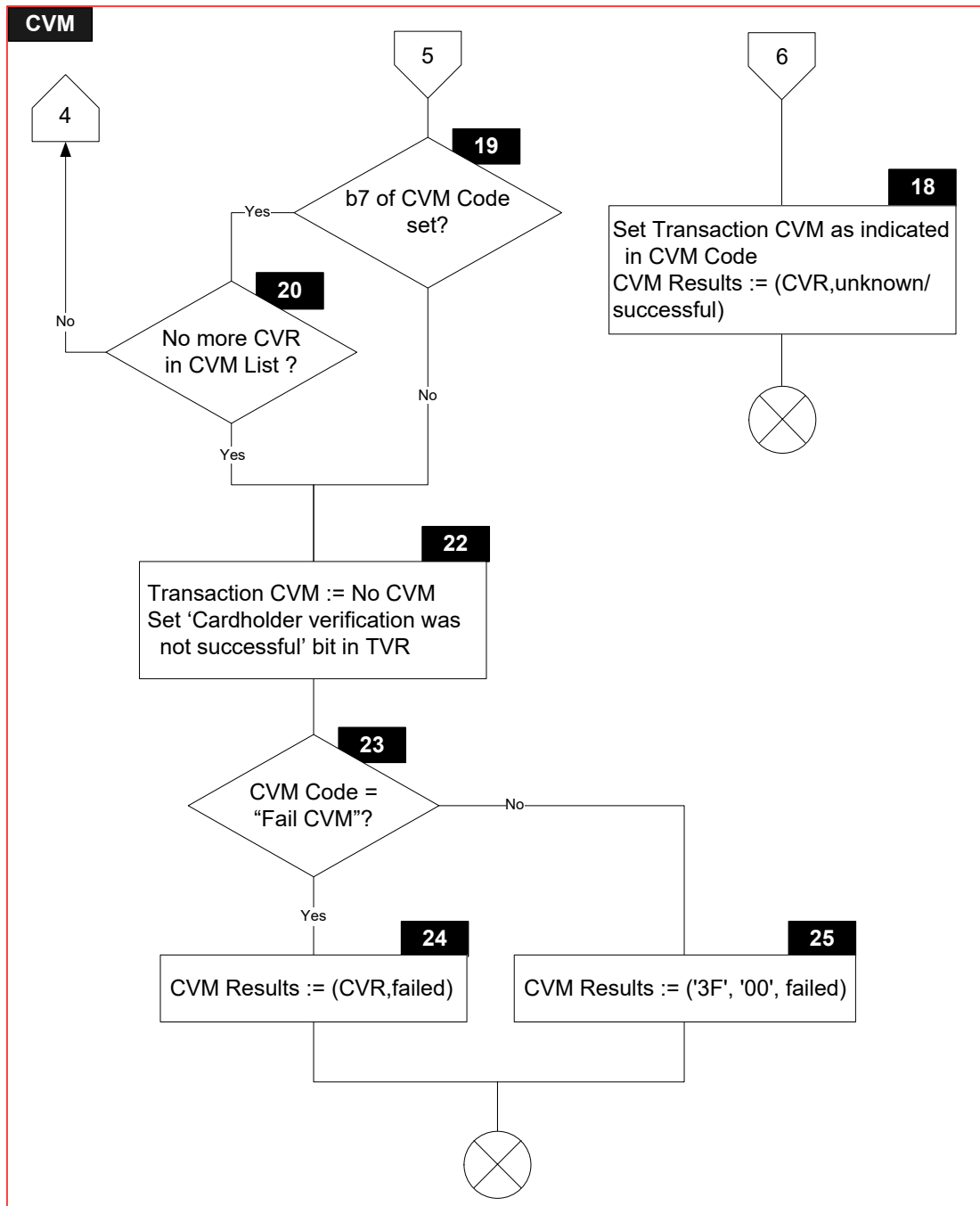
Figure 7.1 shows the flow diagram of the CVM Selection procedure. Symbols in this diagram are labelled CVM.X.

Figure 7.1—CVM Selection Flow Diagram









7.1.3 Processing

CVM.1

```
IF      ['On device cardholder verification is supported' in Application Interchange Profile is set AND  
        'On device cardholder verification supported' in Kernel Configuration is set]  
THEN  
    GOTO CVM.2  
ELSE  
    GOTO CVM.5  
ENDIF
```

CVM.2

```
IF      [Amount, Authorized (Numeric) > Reader CVM Required Limit]  
THEN  
    GOTO CVM.4  
ELSE  
    GOTO CVM.3  
ENDIF
```

CVM.3

```
'CVM' in Outcome Parameter Set := NO CVM  
'CVM Performed' in CVM Results := '3F' (No CVM performed)  
'CVM Condition' in CVM Results := '00'  
'CVM Result' in CVM Results := '02' (successful)
```

CVM.4

```
'CVM' in Outcome Parameter Set := CONFIRMATION CODE VERIFIED  
'CVM Performed' in CVM Results := '01' (on-device cardholder verification performed)  
'CVM Condition' in CVM Results := '00'  
'CVM Result' in CVM Results := '02' (successful)
```

CVM.5

```
IF      ['Cardholder verification is supported' in Application Interchange Profile is set]  
THEN  
    GOTO CVM.7  
ELSE  
    GOTO CVM.6  
ENDIF
```

CVM.6

'CVM' in *Outcome Parameter Set* := NO CVM

'CVM Performed' in *CVM Results* := '3F' (No CVM performed)

'CVM Condition' in *CVM Results* := '00'

'CVM Result' in *CVM Results* := '00' (unknown)

CVM.7

IF [IsNotPresent(TagOf(*CVM List*)) OR IsEmpty(TagOf(*CVM List*))]

THEN

GOTO CVM.8

ELSE

GOTO CVM.9

ENDIF

CVM.8

'CVM' in *Outcome Parameter Set* := NO CVM

'CVM Performed' in *CVM Results* := '3F' (No CVM performed)

'CVM Condition' in *CVM Results* := '00'

'CVM Result' in *CVM Results* := '00' (unknown)

SET 'ICC data missing' in *Terminal Verification Results*

CVM.9

CVR := first CV Rule in *CVM List*

CVM Code := CVR[1]

CVM Condition Code := CVR[2]

CVM.10

IF [CVM Condition Code is understood (i.e. the CVM Condition Code is included in Table 40 of Annex C.3 of [EMV Book 3])]

THEN

GOTO CVM.11

ELSE

GOTO CVM.13

ENDIF

Note that the Kernel may also understand proprietary CVM condition codes not defined in Annex C.3 of [EMV Book 3].

CVM.11

IF [Data required by the conditions expressed by the CVM Condition Code are present in the TLV Database]
THEN
 GOTO CVM.12
ELSE
 GOTO CVM.13
ENDIF

CVM.12

IF [Conditions expressed by the CVM Condition Code are satisfied]
THEN
 GOTO CVM.15
ELSE
 GOTO CVM.13
ENDIF

CVM.13

IF [CVR is last CV Rule in CVM List]
THEN
 GOTO CVM.14
ELSE
 GOTO CVM.21
ENDIF

CVM.14

'CVM' in *Outcome Parameter Set* := NO CVM
'CVM Performed' in *CVM Results* := '3F' (No CVM performed)
'CVM Condition' in *CVM Results* := '00'
'CVM Result' in *CVM Results* := '01' (failed)
SET 'Cardholder verification was not successful' in *Terminal Verification Results*

CVM.15

IF [CVM Code is recognized (i.e. the CVM Code is included in Table 39 of Annex C.3 of [EMV Book 3])]
THEN
 GOTO CVM.17
ELSE
 GOTO CVM.16
ENDIF

Note that the Kernel may also recognize proprietary CVM codes not defined in Annex C.3 of [EMV Book 3].

CVM.16

SET 'Unrecognised CVM' in *Terminal Verification Results*

CVM.17

Verify if the CVM Code is supported:

- For CVM Codes defined in Annex C.3 of [EMV Book 3], support must be indicated in *Terminal Capabilities*.
- For CVM Codes not defined in Annex C.3 of [EMV Book 3], support may be known explicitly.
- For combination CVMs, both CVM codes must be supported.
- Fail CVM processing ('00' or '40') must always be supported.

IF [CVM Code is supported AND ((CVM Code AND '3F') ≠ '00')]

THEN

GOTO CVM.18

ELSE

GOTO CVM.19

ENDIF

CVM.18

```
IF      [(CVM Code AND '3F')= '02']
THEN
    'CVM' in Outcome Parameter Set := ONLINE PIN
    'CVM Result' in CVM Results := '00' (unknown)
    SET 'Online PIN entered' in Terminal Verification Results
ELSE
    IF      [(CVM Code AND '3F') = '1E']
    THEN
        'CVM' in Outcome Parameter Set := OBTAIN SIGNATURE
        'CVM Result' in CVM Results := '00' (unknown)
        'Receipt' in Outcome Parameter Set := YES
    ELSE
        IF      [(CVM Code AND '3F') = '1F']
        THEN
            'CVM' in Outcome Parameter Set := NO CVM
            'CVM Result' in CVM Results := '02' (successful)
        ELSE
            Set 'CVM' in Outcome Parameter Set to proprietary value
            'CVM Result' in CVM Results := '00' or '02'
        ENDIF
    ENDIF
ENDIF

'CVM Performed' in CVM Results := CVM Code
'CVM Condition' in CVM Results := CVM Condition Code
```

CVM.19

```
IF      [CVM Code[7] is set (i.e. apply succeeding CV Rule if this CVM is
        unsuccessful)]
THEN
    GOTO CVM.20
ELSE
    GOTO CVM.22
ENDIF
```

CVM.20

```
IF      [CVR is last CV Rule in CVM List]
THEN
    GOTO CVM.22
ELSE
    GOTO CVM.21
ENDIF
```

CVM.21

```
CVR := next CV Rule in CVM List
CVM Code := CVR[1]
CVM Condition Code := CVR[2]
```

CVM.22

```
'CVM' in Outcome Parameter Set := NO CVM
SET 'Cardholder verification was not successful' in Terminal Verification Results
```

CVM.23

```
IF      [(CVM Code AND '3F') = '00']
THEN
    GOTO CVM.24
ELSE
    GOTO CVM.25
ENDIF
```

CVM.24

```
'CVM Performed' in CVM Results := CVM Code
'CVM Condition' in CVM Results := 'CVM Condition Code'
'CVM Result' in CVM Results := '01' (failed)
```

CVM.25

```
'CVM Performed' in CVM Results := '3F'
'CVM Condition' in CVM Results := '00'
'CVM Result' in CVM Results := '01' (failed)
```

7.2 Procedure – Prepare Generate AC Command

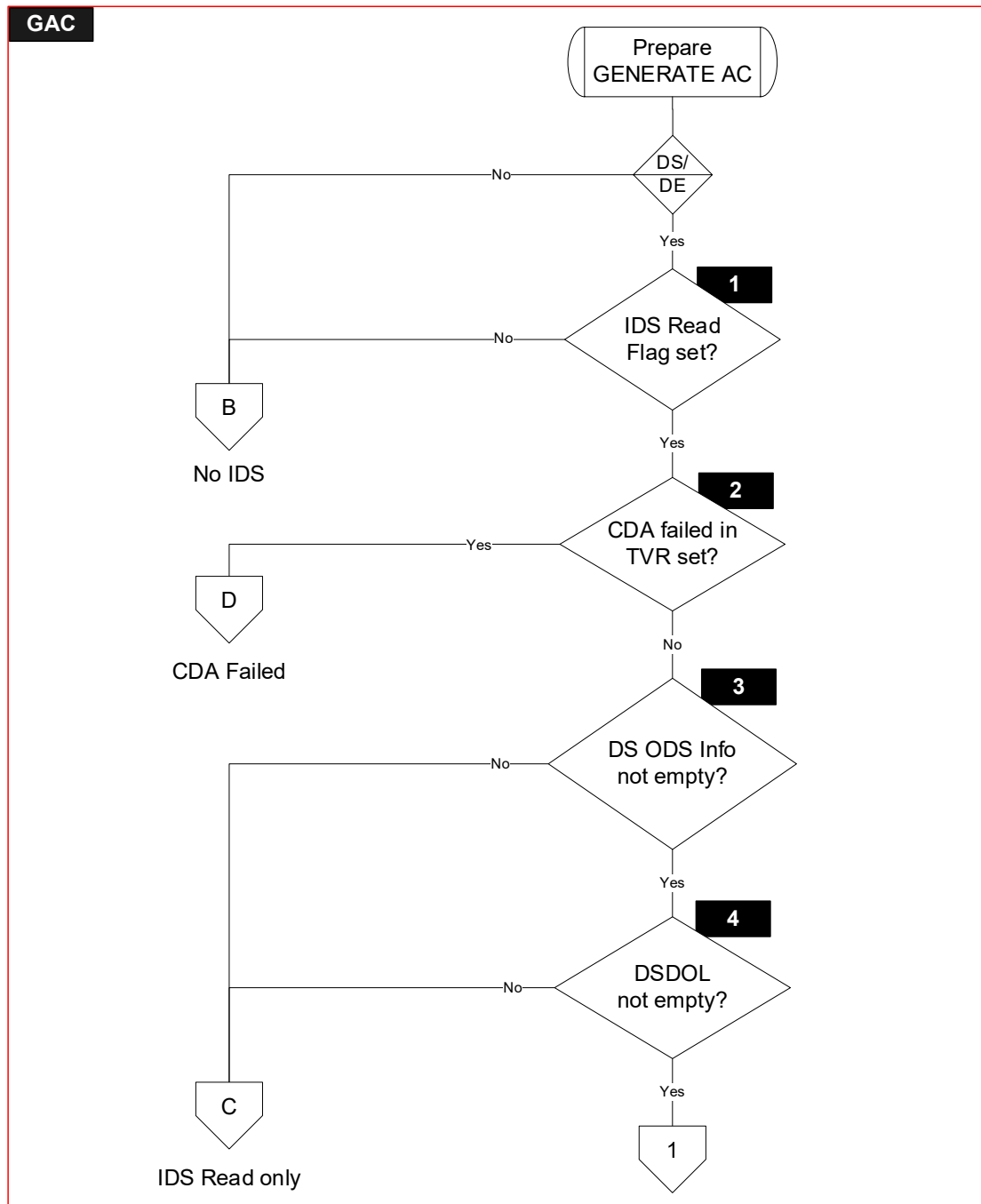
7.2.1 Local Variables

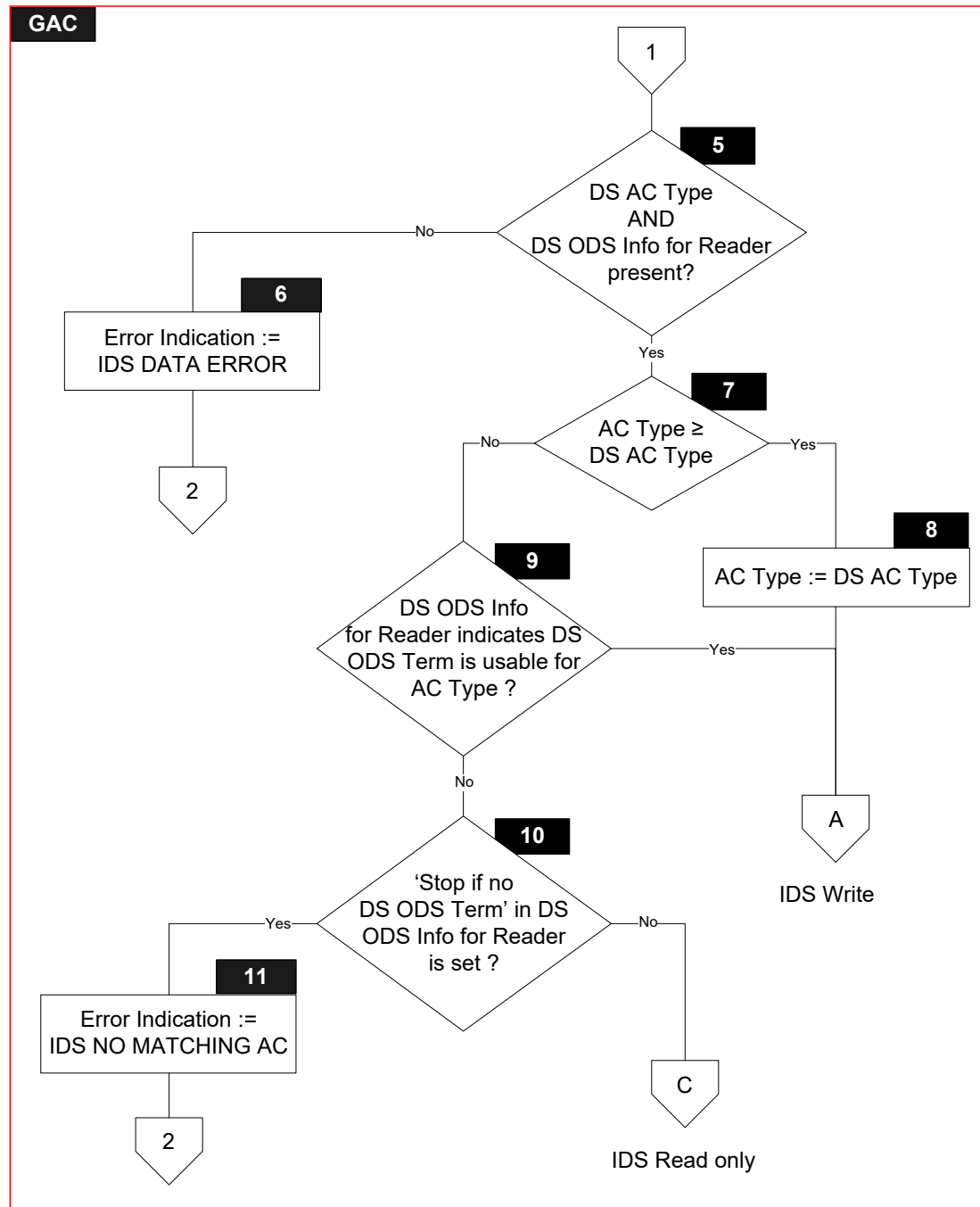
None

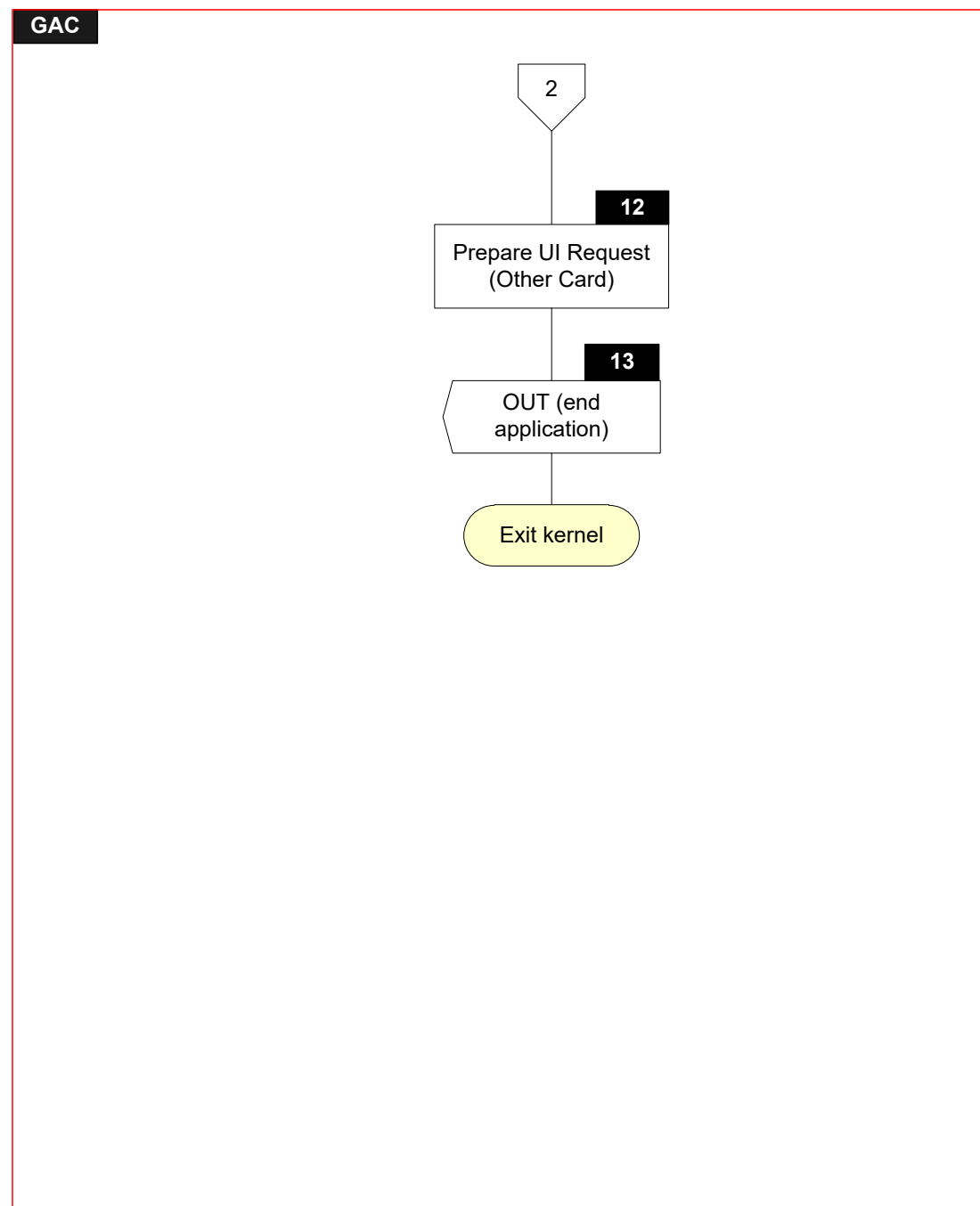
7.2.2 Flow Diagram

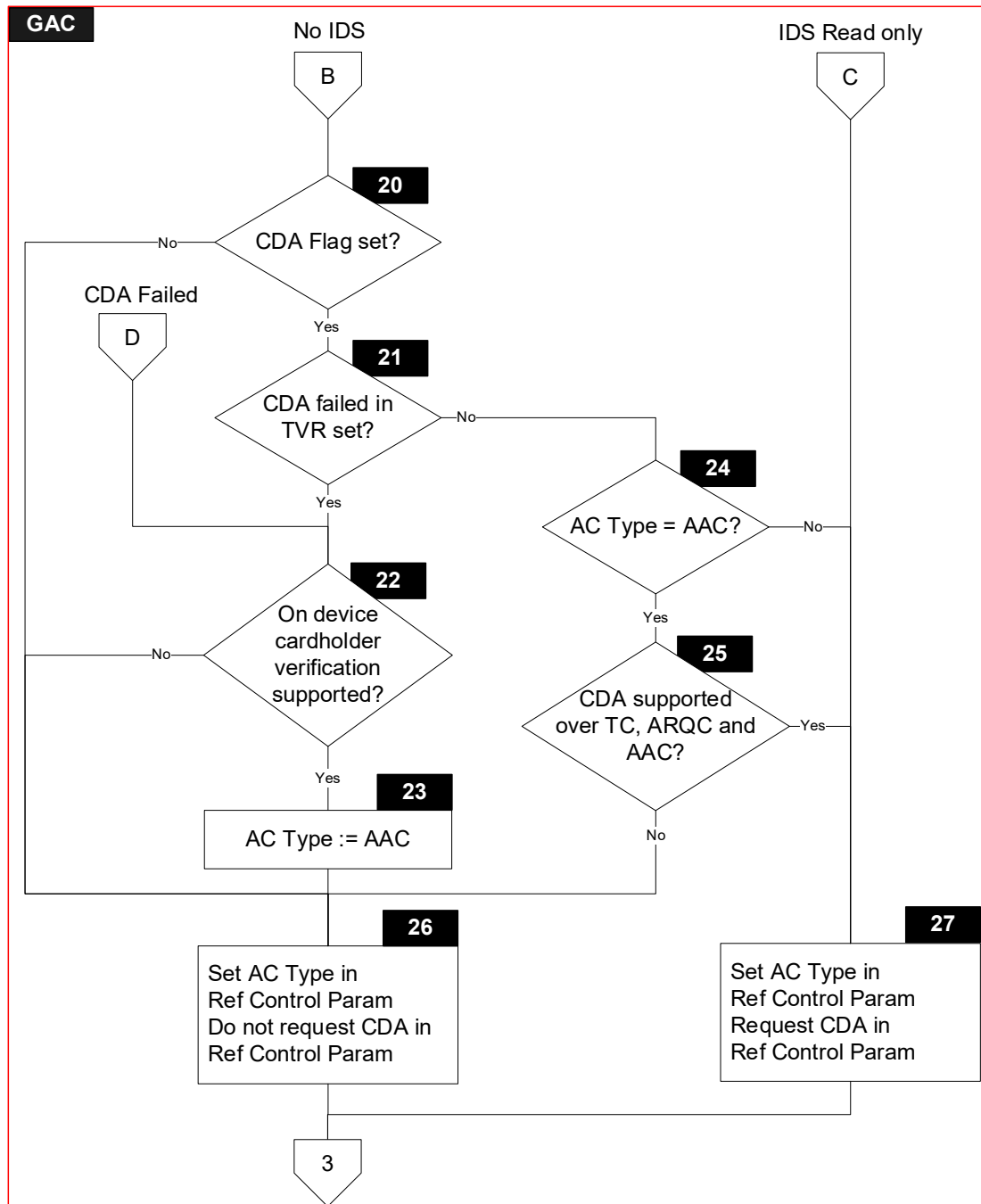
Figure 7.2 shows the flow diagram of the Prepare Generate AC Command procedure. Symbols in this diagram are labelled GAC.X.

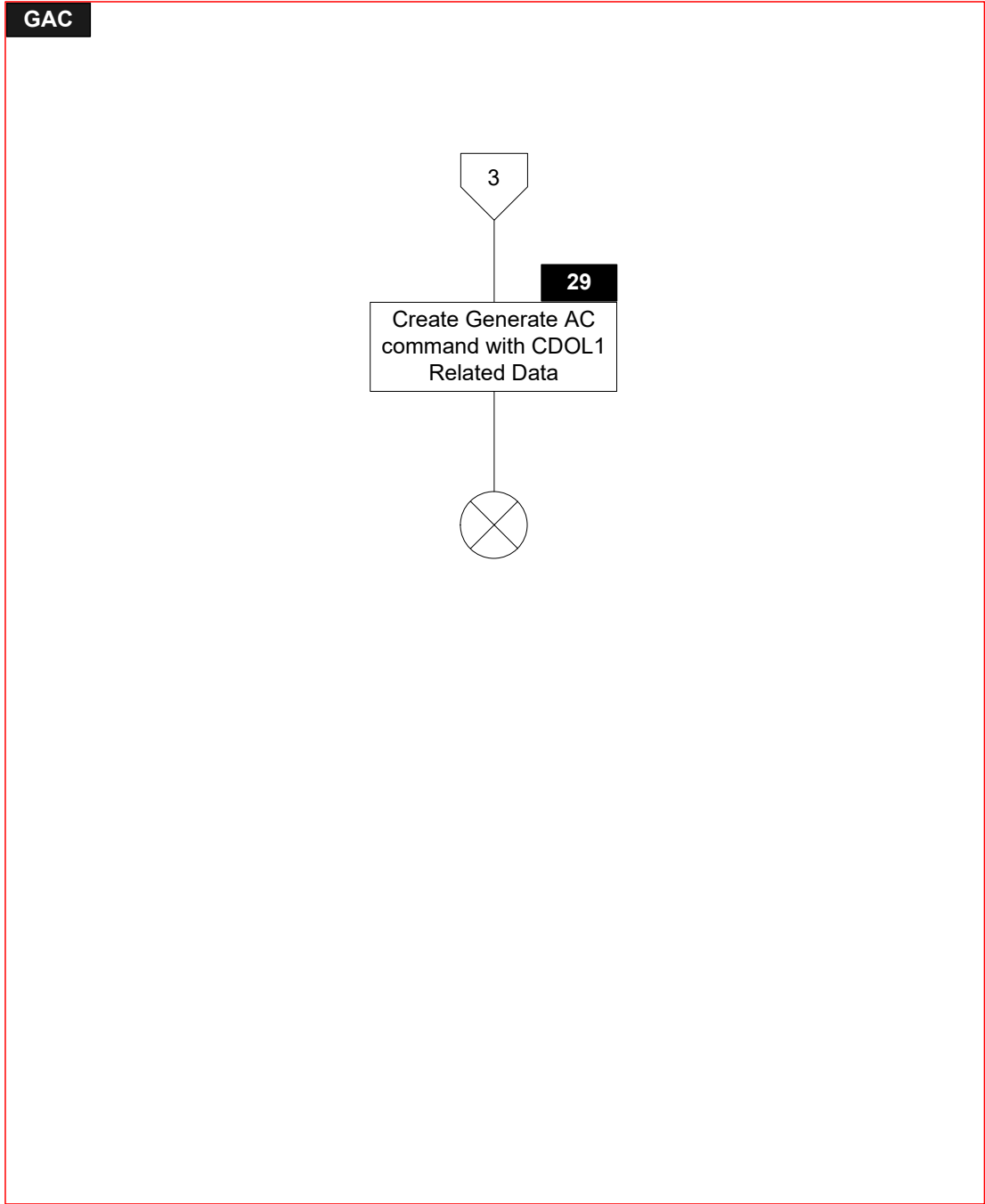
Figure 7.2—Prepare Generate AC Command Flow Diagram

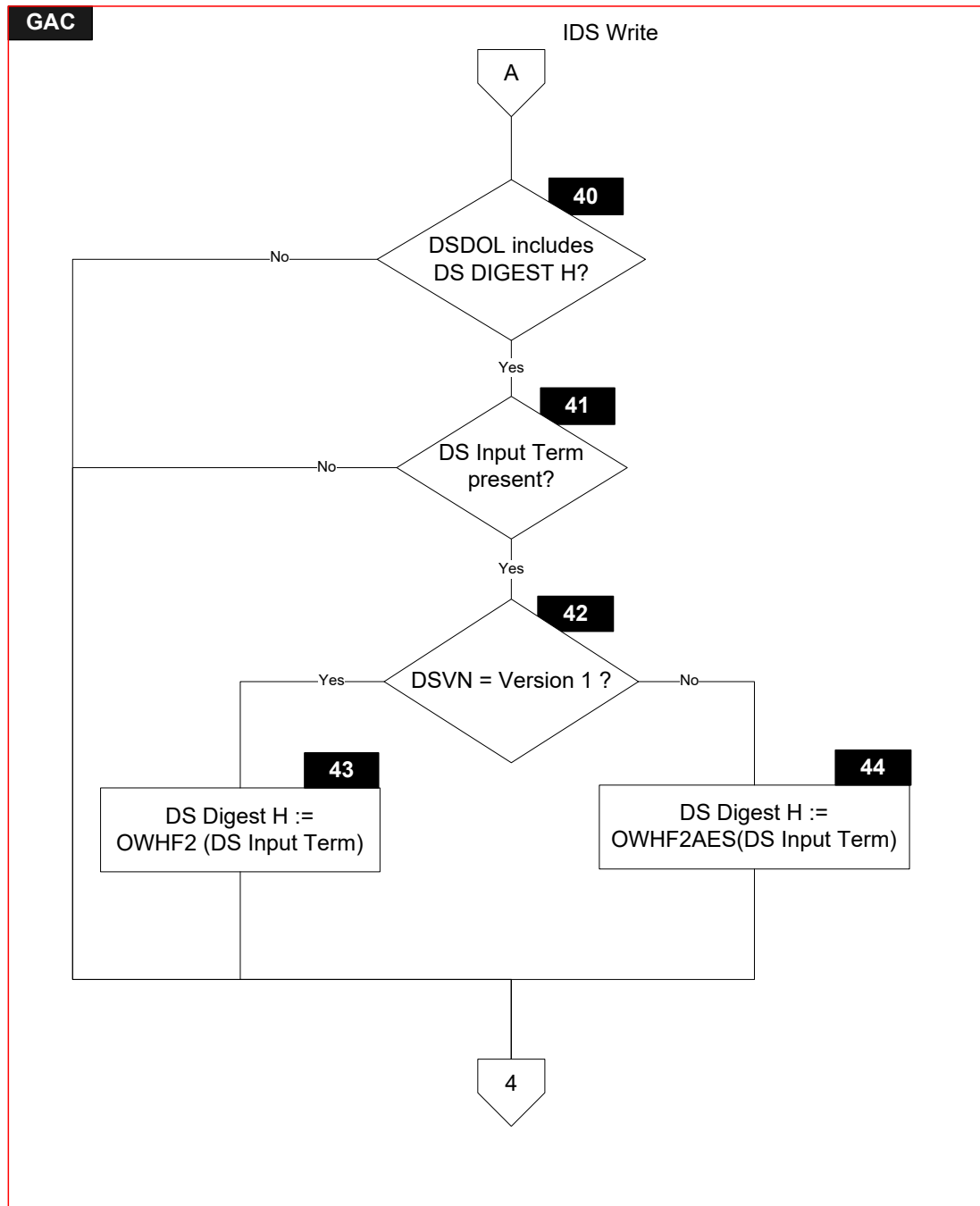


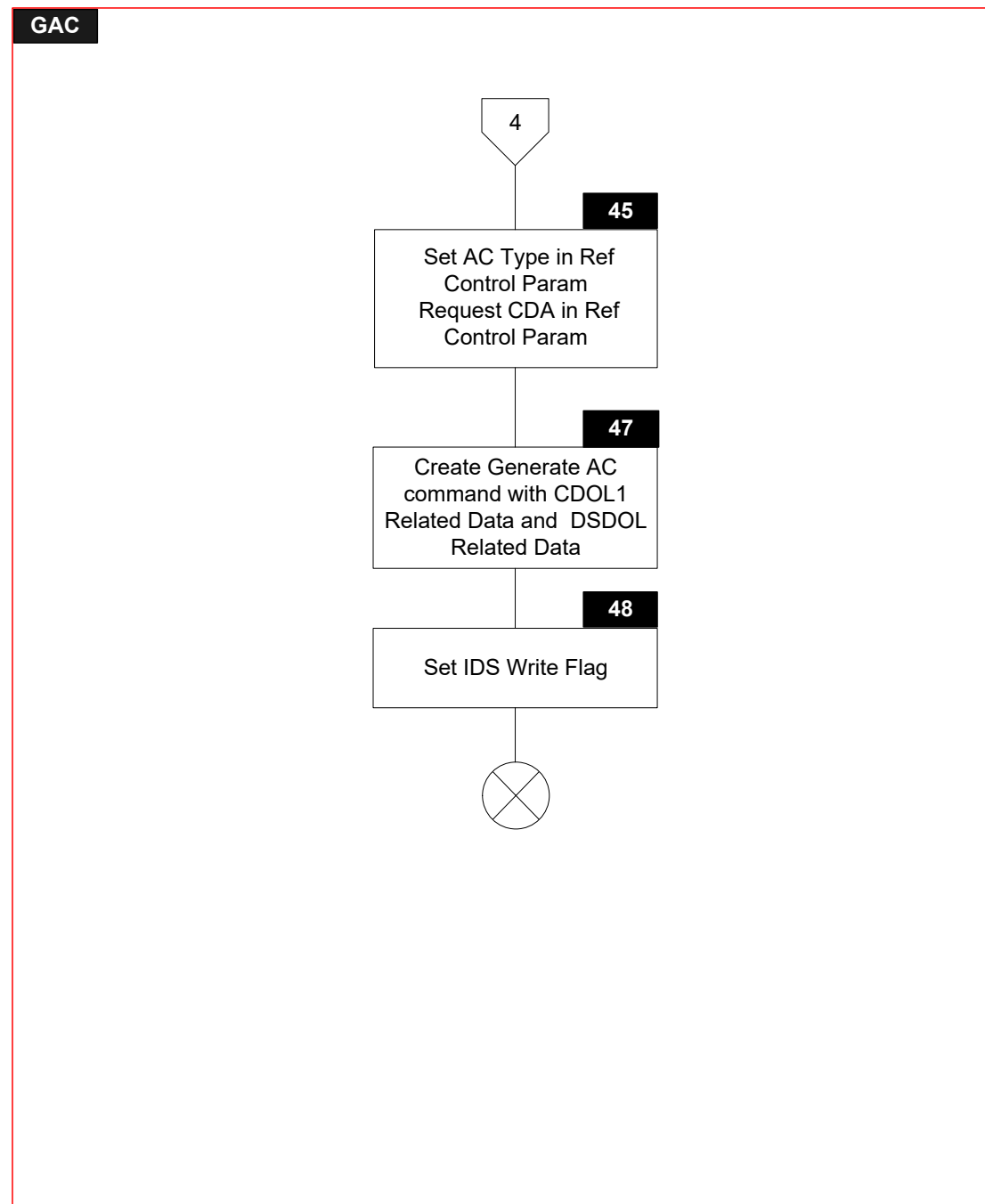












7.2.3 Processing

Note that symbols GAC.1, GAC.2, GAC.3, GAC.4, GAC.5, GAC.6, GAC.7, GAC.8, GAC.9, GAC.10, GAC.11, GAC.12, GAC.13, GAC.40, GAC.41, GAC.42, GAC.43, GAC.44, GAC.45, GAC.47 and GAC.48 only have to be implemented for the DE/DS Implementation Option.

GAC.1

```
IF      ['Read' in IDS Status is set]
THEN
    GOTO GAC.2
ELSE
    GOTO GAC.20
ENDIF
```

GAC.2

```
IF      ['CDA failed' in Terminal Verification Results is set]
THEN
    GOTO GAC.22
ELSE
    GOTO GAC.3
ENDIF
```

GAC.3

```
IF      [IsEmpty(TagOf(DS ODS Info))]
THEN
    GOTO GAC.4
ELSE
    GOTO GAC.27
ENDIF
```

GAC.4

```
IF      [IsEmpty(TagOf(DSDOL))]
THEN
    GOTO GAC.5
ELSE
    GOTO GAC.27
ENDIF
```

GAC.5

```
IF      [IsNotEmpty(TagOf(DS AC Type)) AND
        IsNotEmpty(TagOf(DS ODS Info For Reader))]  
THEN  
    GOTO GAC.7  
ELSE  
    GOTO GAC.6  
ENDIF
```

GAC.6

'L2' in *Error Indication* := IDS DATA ERROR

GAC.7

```
IF      [('AC type' in DS AC Type = AAC)  
        OR  
        ('AC type' in AC Type = 'AC type' in DS AC Type)  
        OR  
        (('AC type' in DS AC Type = ARQC) AND ('AC type' in AC Type = TC))]  
THEN  
    GOTO GAC.8  
ELSE  
    GOTO GAC.9  
ENDIF
```

GAC.8

'AC type' in *AC Type* := 'AC type' in *DS AC Type*

GAC.9

```
IF      [ (('AC type' in AC Type = AAC) AND 'Usable for AAC' in DS ODS Info For  
Reader is set)  
        OR  
        (('AC type' in AC Type = ARQC) AND 'Usable for ARQC' in DS ODS Info  
For Reader is set)]  
THEN  
    GOTO GAC.40  
ELSE  
    GOTO GAC.10  
ENDIF
```

GAC.10

IF [‘Stop if no DS ODS Term’ in *DS ODS Info For Reader* is set]
THEN
 GOTO GAC.11
ELSE
 GOTO GAC.27
ENDIF

GAC.11

‘L2’ in *Error Indication* := IDS NO MATCHING AC

GAC.12

‘Message Identifier’ in *User Interface Request Data* := ERROR – OTHER CARD

‘Status’ in *User Interface Request Data* := NOT READY

GAC.13

‘Status’ in *Outcome Parameter Set* := END APPLICATION

‘Msg On Error’ in *Error Indication* := ERROR – OTHER CARD

CreateEMVDiscretionaryData ()

SET ‘UI Request on Outcome Present’ in *Outcome Parameter Set*

Send OUT(GetTLV(TagOf(*Outcome Parameter Set*)),
 GetTLV(TagOf(*Discretionary Data*)),
 GetTLV(TagOf(*User Interface Request Data*))) Signal

No IDS

GAC.20

```
IF      ['CDA' in ODA Status is set]
THEN
    GOTO GAC.21
ELSE
    GOTO GAC.26
ENDIF
```

GAC.21

```
IF      ['CDA failed' in Terminal Verification Results is set]
THEN
    GOTO GAC.22
ELSE
    GOTO GAC.24
ENDIF
```

GAC.22

```
IF      ['On device cardholder verification is supported' in Application Interchange Profile is set AND
        'On device cardholder verification supported' in Kernel Configuration is set]
THEN
    GOTO GAC.23
ELSE
    GOTO GAC.26
ENDIF
```

GAC.23

'AC type' in *AC Type* := AAC

GAC.24

```
IF      ['AC type' in AC Type = AAC]
THEN
    GOTO GAC.25
ELSE
    GOTO GAC.27
ENDIF
```

GAC.25

IF [IsEmpty(TagOf(*Application Capabilities Information*)) AND
'CDA Indicator' in *Application Capabilities Information* = CDA
SUPPORTED OVER TC, ARQC AND AAC]
THEN
GOTO GAC.27
ELSE
GOTO GAC.26
ENDIF

GAC.26

Reference Control Parameter := '00'
'AC type' in *Reference Control Parameter* := 'AC type' in *AC Type*

GAC.27

Reference Control Parameter := '00'
'AC type' in *Reference Control Parameter* := 'AC type' in *AC Type*
SET 'CDA signature requested' in *Reference Control Parameter*

GAC.29

Prepare GENERATE AC command as specified in section 5.4.2. Use *CDOL1* to create *CDOL1 Related Data* as a concatenated list of data objects without tags or lengths following the rules specified in section 4.1.4.

IDS Write

GAC.40

```
IF      [DSDOL includes TagOf(DS Digest H)]
THEN
    GOTO GAC.41
ELSE
    GOTO GAC.45
ENDIF
```

GAC.41

```
IF      [IsPresent(TagOf(DS Input (Term)))]
THEN
    GOTO GAC.42
ELSE
    GOTO GAC.45
ENDIF
```

GAC.42

```
IF      ['Data Storage Version Number' in Application Capabilities Information =
        VERSION 1]
THEN
    GOTO GAC.43
ELSE
    GOTO GAC.44
ENDIF
```

GAC.43

DS Digest H := OWHF2(*DS Input (Term)*)
Refer to section 8.2 for the description of OWHF2

GAC.44

DS Digest H := OWHF2AES(*DS Input (Term)*)
Refer to section 8.3 for the description of OWHF2AES

GAC.45

Reference Control Parameter := '00'
'AC type' in *Reference Control Parameter* := 'AC type' in *AC Type*
SET 'CDA signature requested' in *Reference Control Parameter*

GAC.47

Prepare GENERATE AC command as specified in section 5.4.2. Use *CDOL1* and *DSDOL* to create *CDOL1 Related Data* and *DSDOL* related data as concatenated lists of data objects without tags or lengths following the rules specified in section 4.1.4.

GAC.48

SET 'Write' in *IDS Status*

7.3 Procedure – Processing Restrictions

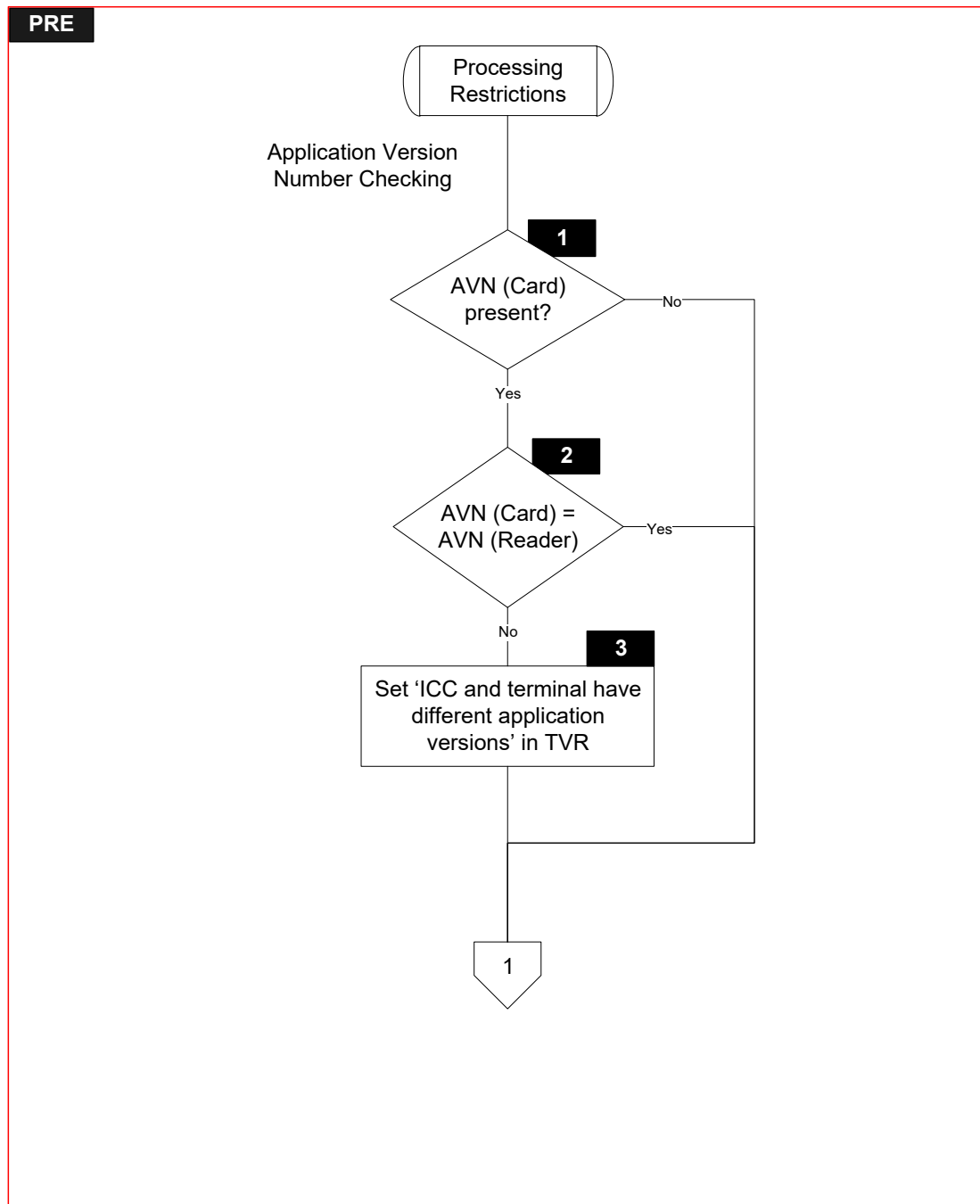
7.3.1 Local Variables

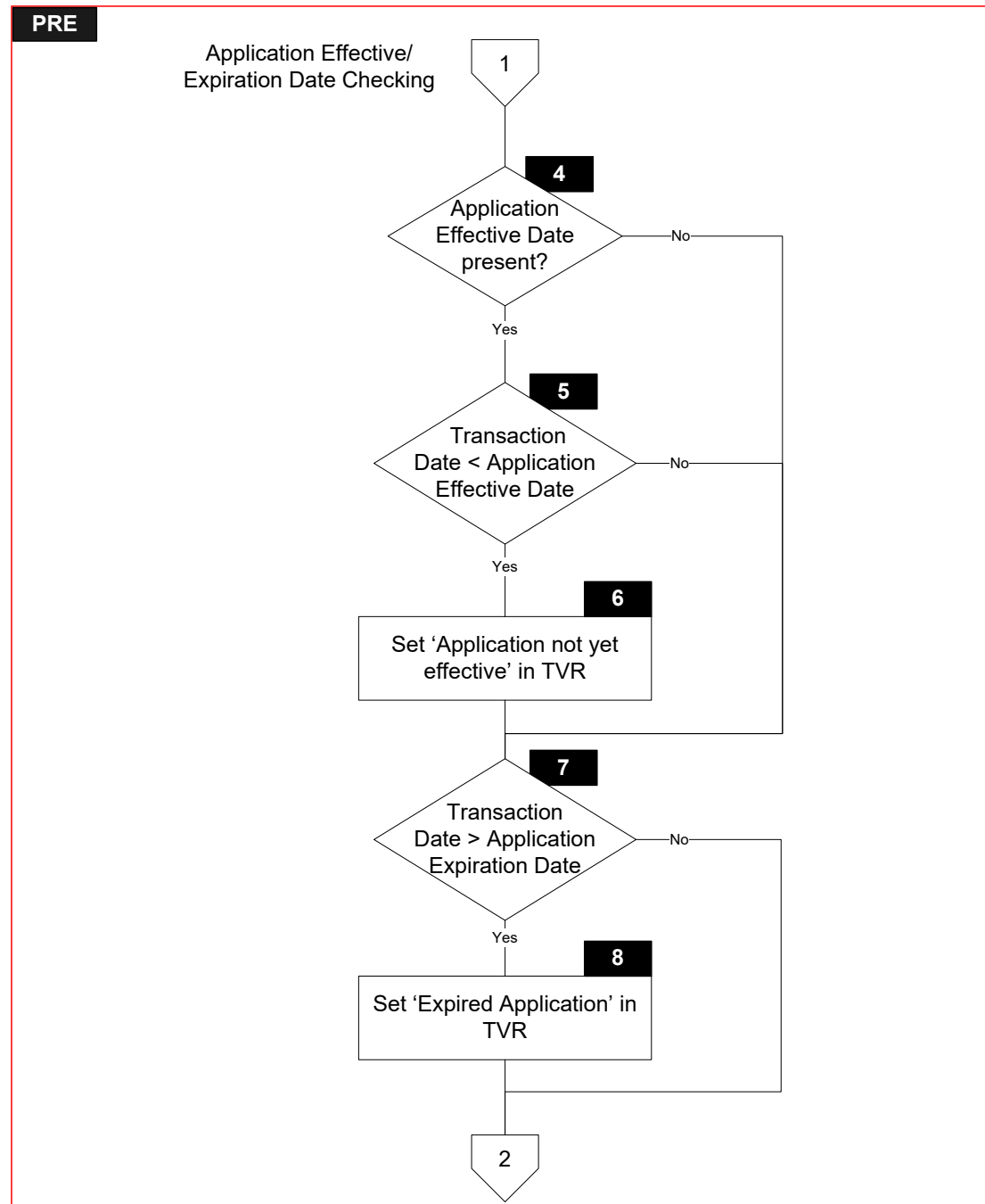
None

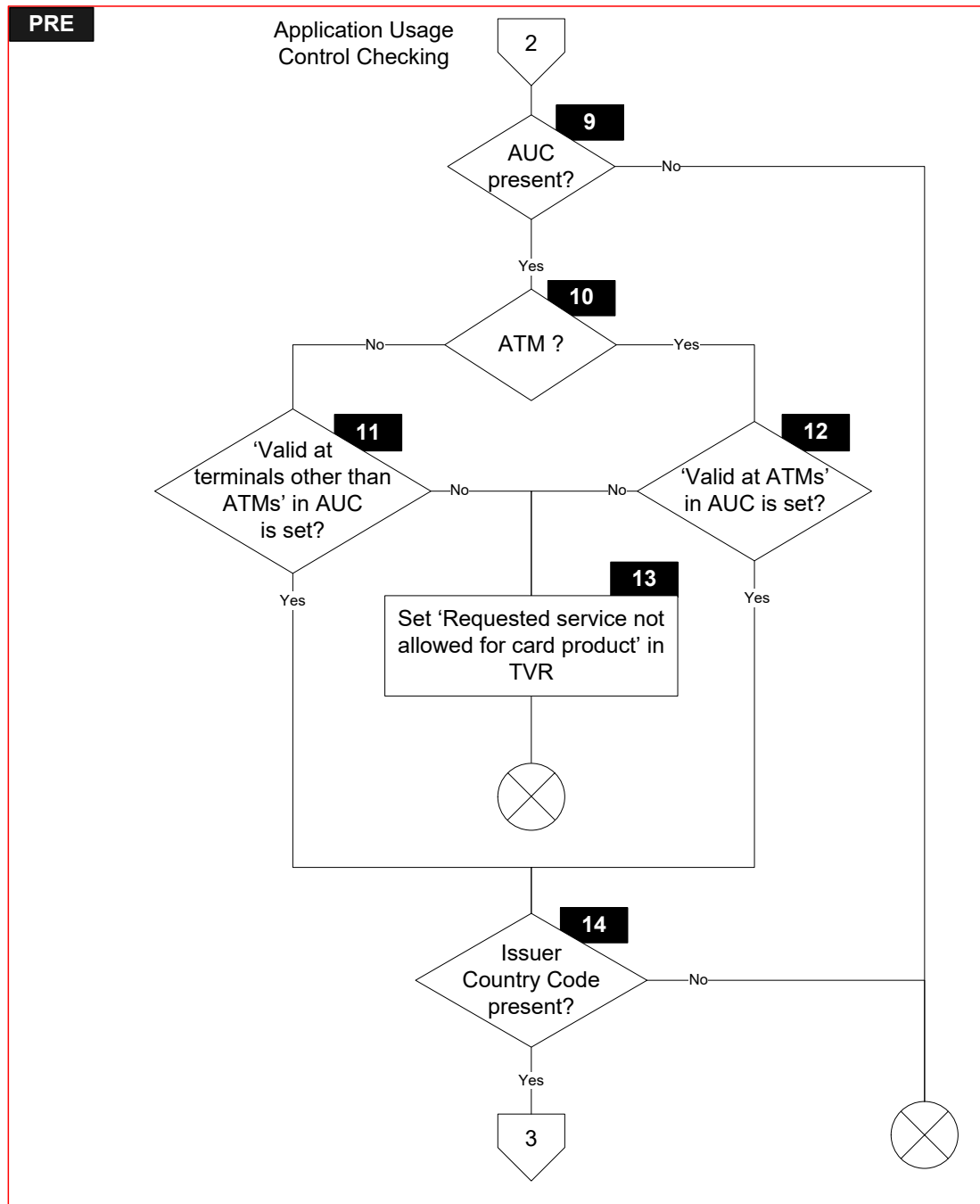
7.3.2 Flow Diagram

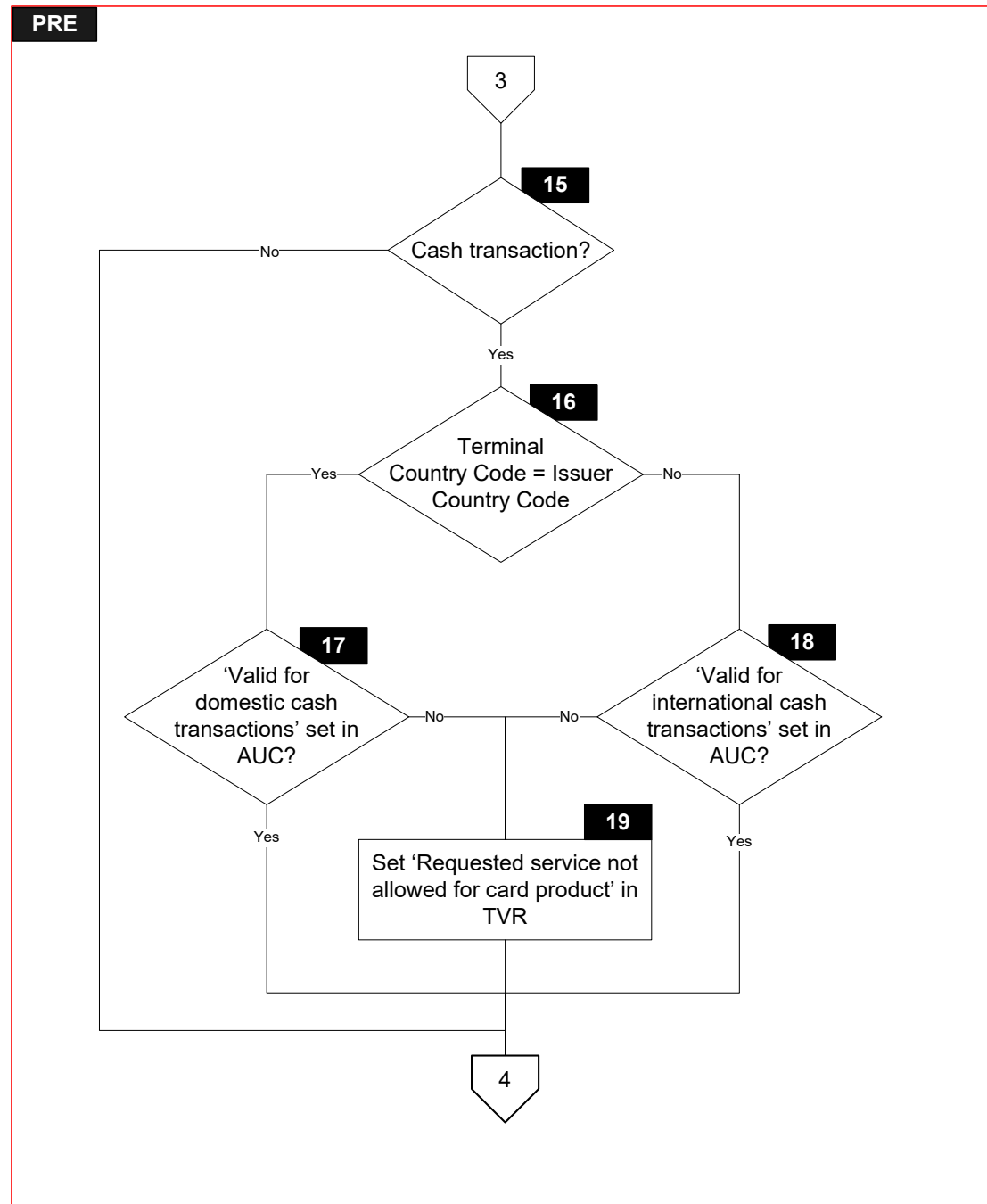
Figure 7.3 shows the flow diagram of the Processing Restrictions procedure. Symbols in this diagram are labelled PRE.X.

Figure 7.3—Processing Restrictions Flow Diagram

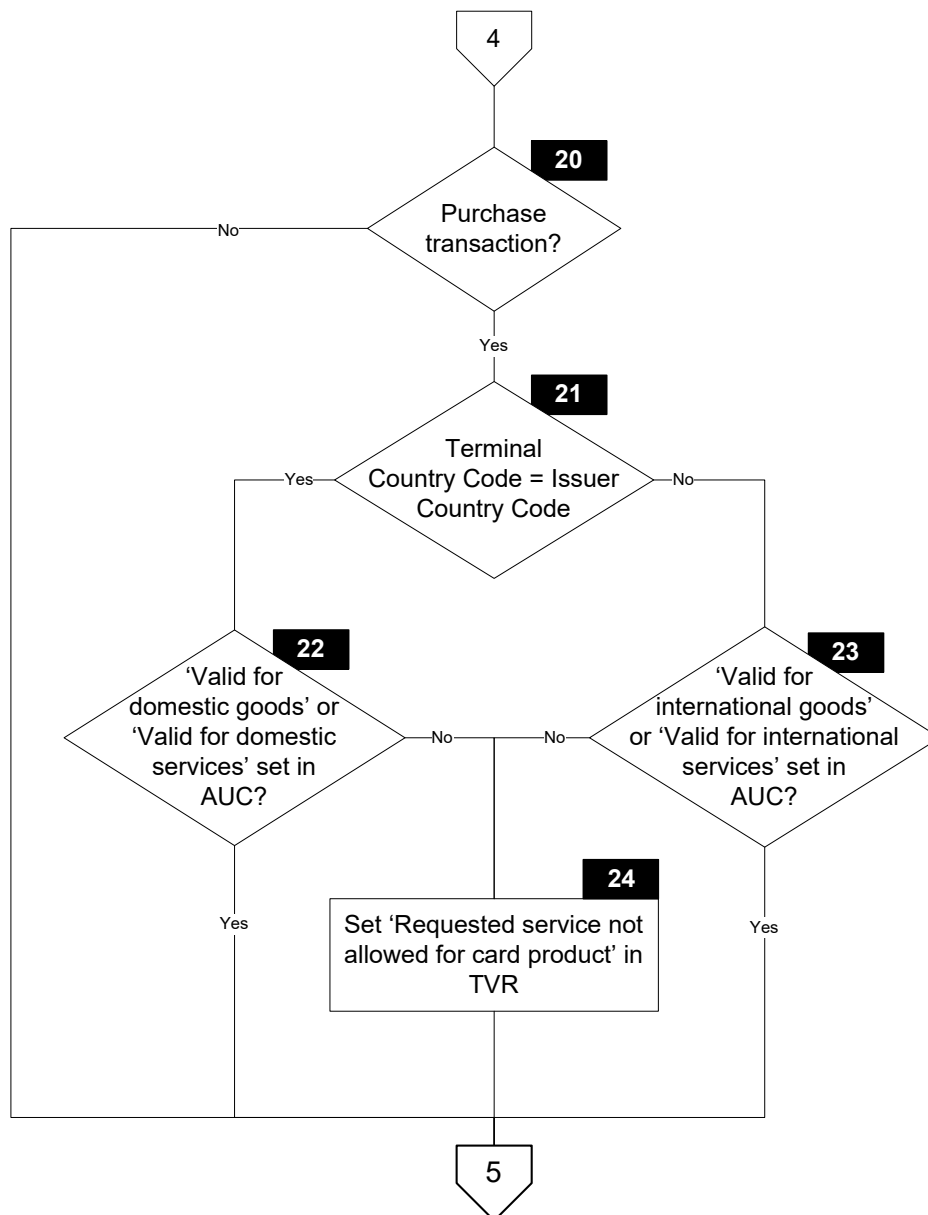


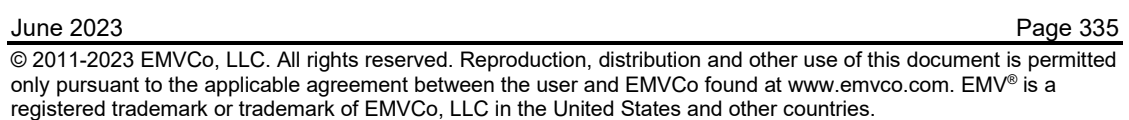






PRE





7.3.3 Processing

Application Version Number Checking

PRE.1

```
IF      [IsEmpty(TagOf(Application Version Number (Card)))]
THEN
    GOTO PRE.2
ELSE
    GOTO PRE.4
ENDIF
```

PRE.2

```
IF      [Application Version Number (Card) = Application Version Number (Reader)]
THEN
    GOTO PRE.4
ELSE
    GOTO PRE.3
ENDIF
```

PRE.3

SET 'ICC and terminal have different application versions' in *Terminal Verification Results*

Application Effective/Expiration Date Checking

PRE.4

```
IF      [IsEmpty(TagOf(Application Effective Date))]
THEN
    GOTO PRE.5
ELSE
    GOTO PRE.7
ENDIF
```

PRE.5

```
IF      [Transaction Date is before Application Effective Date]
THEN
    GOTO PRE.6
ELSE
    GOTO PRE.7
ENDIF
```

PRE.6

SET 'Application not yet effective' in *Terminal Verification Results*

PRE.7

```
IF      [Transaction Date is after Application Expiration Date]
THEN
    GOTO PRE.8
ELSE
    GOTO PRE.9
ENDIF
```

PRE.8

SET 'Expired application' in *Terminal Verification Results*

Application Usage Control Checking

PRE.9

```
IF      [IsEmpty(TagOf(Application Usage Control))]  
THEN  
    GOTO PRE.10  
ELSE  
    EXIT Processing Restrictions  
ENDIF
```

PRE.10

```
IF      [((Terminal Type = '14') OR  
          (Terminal Type = '15') OR  
          (Terminal Type = '16'))  
        AND  
        'Cash' in Additional Terminal Capabilities is set]  
THEN  
    GOTO PRE.12  
ELSE  
    GOTO PRE.11  
ENDIF
```

PRE.11

```
IF      ['Valid at terminals other than ATMs' in Application Usage Control is set]  
THEN  
    GOTO PRE.14  
ELSE  
    GOTO PRE.13  
ENDIF
```

PRE.12

```
IF      ['Valid at ATMs' in Application Usage Control is set]  
THEN  
    GOTO PRE.14  
ELSE  
    GOTO PRE.13  
ENDIF
```

PRE.13

```
SET 'Requested service not allowed for card product' in Terminal Verification Results
```

PRE.14

```
IF      [IsNotEmpty(TagOf(Issuer Country Code))]
THEN
    GOTO PRE.15
ELSE
    EXIT Processing Restrictions
ENDIF
```

PRE.15

Check if *Transaction Type* indicates a cash transaction (cash withdrawal or cash disbursement).

```
IF      [Transaction Type = '01' OR Transaction Type = '17']
THEN
    GOTO PRE.16
ELSE
    GOTO PRE.20
ENDIF
```

PRE.16

```
IF      [Terminal Country Code = Issuer Country Code]
THEN
    GOTO PRE.17
ELSE
    GOTO PRE.18
ENDIF
```

PRE.17

```
IF      ['Valid for domestic cash transactions' in Application Usage Control is set]
THEN
    GOTO PRE.20
ELSE
    GOTO PRE.19
ENDIF
```

PRE.18

```
IF      ['Valid for international cash transactions' in Application Usage Control is set]
THEN
    GOTO PRE.20
ELSE
    GOTO PRE.19
ENDIF
```

PRE.19

SET 'Requested service not allowed for card product' in *Terminal Verification Results*

PRE.20

Check if *Transaction Type* indicates a purchase transaction (purchase or purchase with cashback).

IF [*Transaction Type* = '00' OR *Transaction Type* = '09']

THEN

GOTO PRE.21

ELSE

GOTO PRE.25

ENDIF

PRE.21

IF [*Terminal Country Code* = *Issuer Country Code*]

THEN

GOTO PRE.22

ELSE

GOTO PRE.23

ENDIF

PRE.22

IF ['Valid for domestic goods' in *Application Usage Control* is set OR
'Valid for domestic services' in *Application Usage Control* is set]

THEN

GOTO PRE.25

ELSE

GOTO PRE.24

ENDIF

PRE.23

IF ['Valid for international goods' in *Application Usage Control* is set OR
'Valid for international services' in *Application Usage Control* is set]

THEN

GOTO PRE.25

ELSE

GOTO PRE.24

ENDIF

PRE.24

SET 'Requested service not allowed for card product' in *Terminal Verification Results*

PRE.25

```
IF      [IsPresent(TagOf(Amount, Other (Numeric))) AND
        (Amount, Other (Numeric) ≠ '000000000000')]
THEN
    GOTO PRE.26
ELSE
    EXIT Processing Restrictions
ENDIF
```

PRE.26

```
IF      [Terminal Country Code = Issuer Country Code]
THEN
    GOTO PRE.27
ELSE
    GOTO PRE.28
ENDIF
```

PRE.27

```
IF      ['Domestic cashback allowed' in Application Usage Control is set]
THEN
    EXIT Processing Restrictions
ELSE
    GOTO PRE.29
ENDIF
```

PRE.28

```
IF      ['International cashback allowed' in Application Usage Control is set]
THEN
    EXIT Processing Restrictions
ELSE
    GOTO PRE.29
ENDIF
```

PRE.29

```
SET 'Requested service not allowed for card product' in Terminal Verification Results
```

7.4 Procedure – Terminal Action Analysis

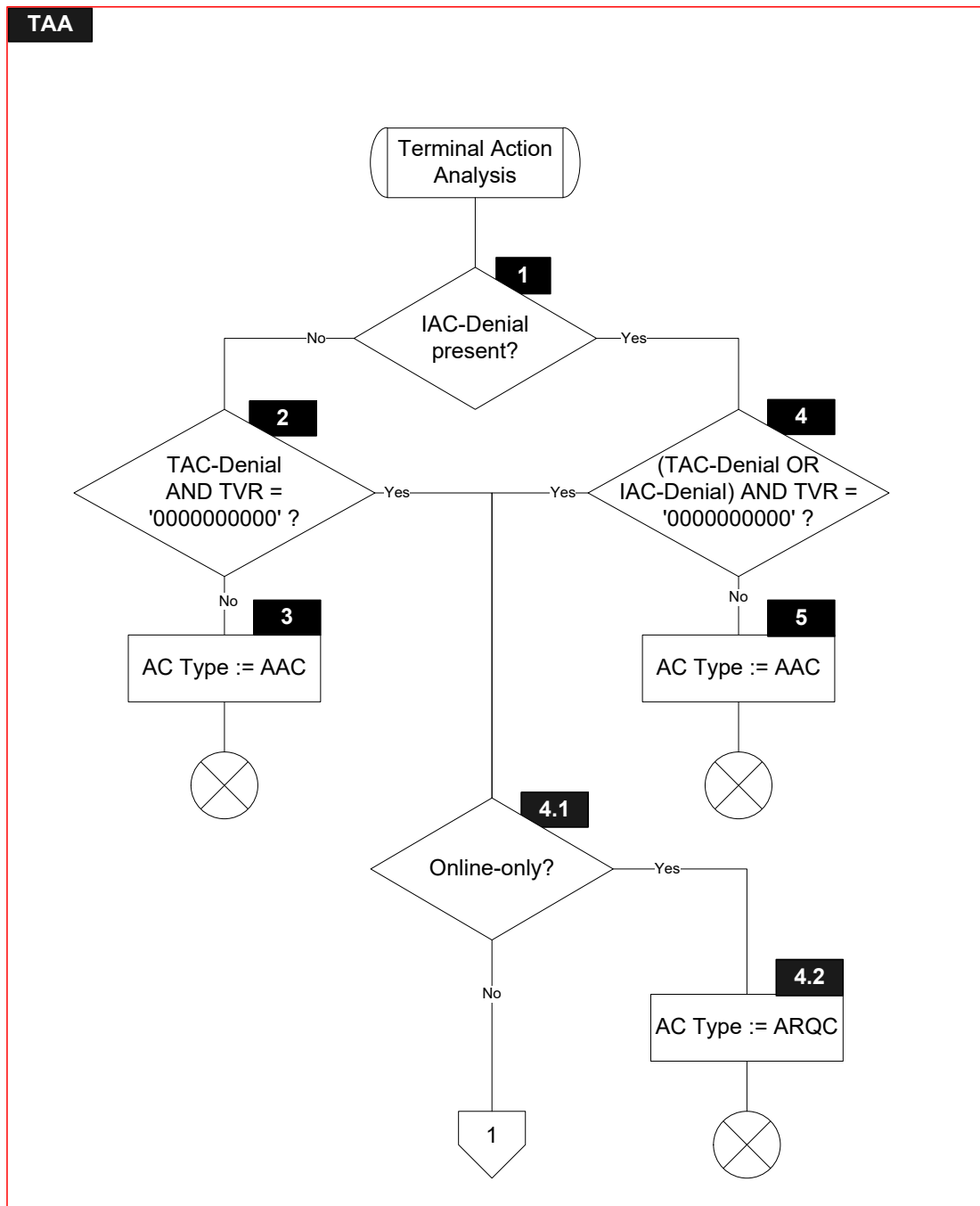
7.4.1 Local Variables

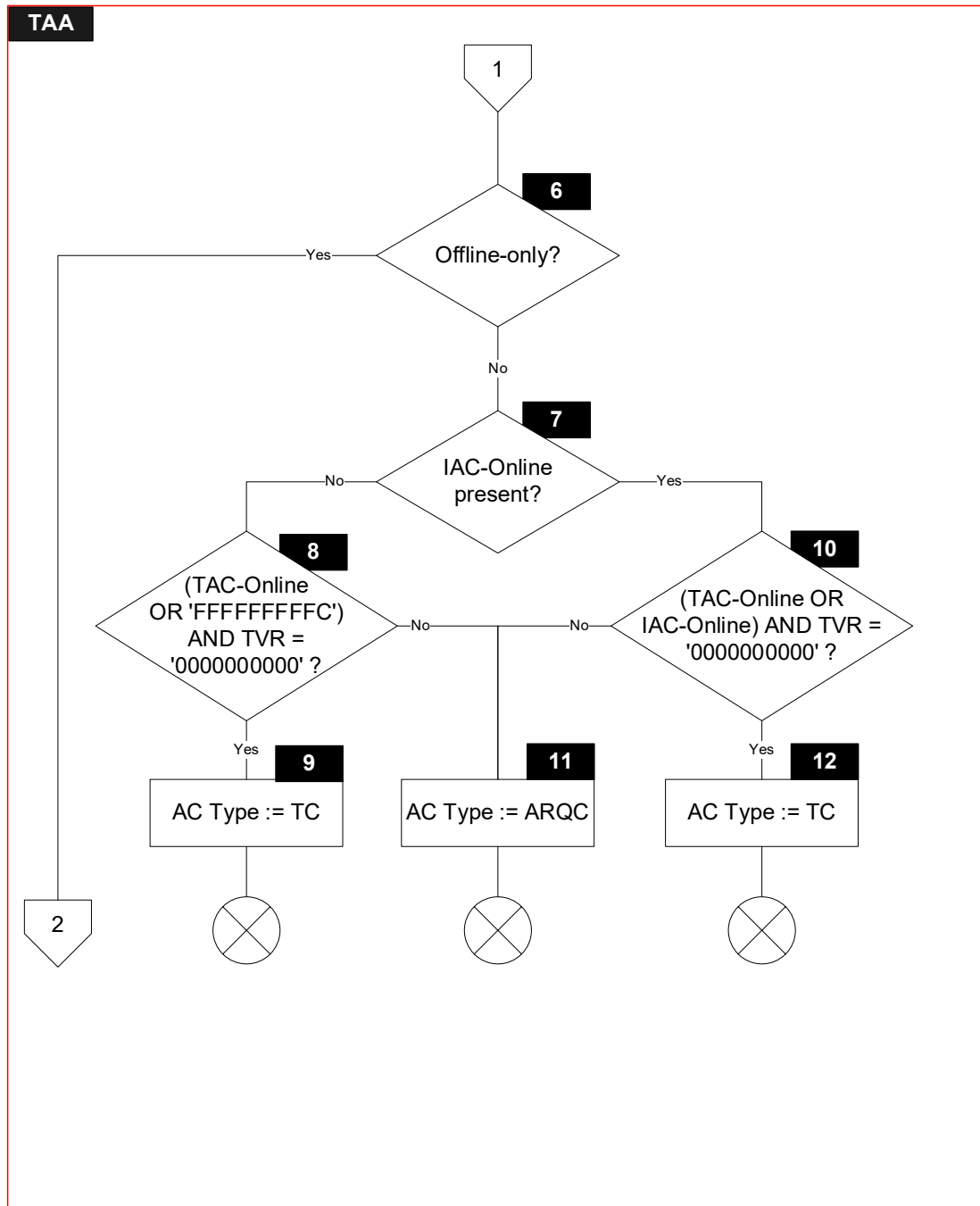
None

7.4.2 Flow Diagram

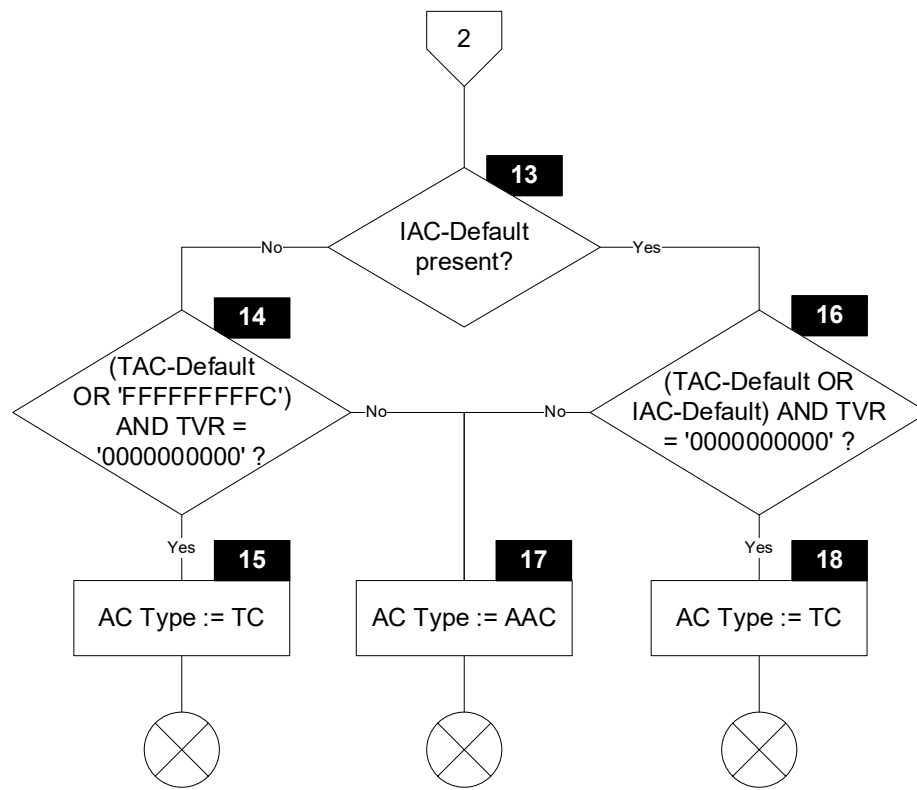
Figure 7.4 shows the flow diagram of the Terminal Action Analysis procedure. Symbols in this diagram are labelled TAA.X.

Figure 7.4—Terminal Action Analysis Flow Diagram





TAA



7.4.3 Processing

TAA.1

```
IF      [IsEmpty(TagOf(Issuer Action Code – Denial))]  
THEN  
    GOTO TAA.4  
ELSE  
    GOTO TAA.2  
ENDIF
```

TAA.2

```
IF      [(Terminal Action Code – Denial AND Terminal Verification Results) =  
        '0000000000']  
THEN  
    GOTO TAA.4.1  
ELSE  
    GOTO TAA.3  
ENDIF
```

TAA.3

'AC type' in *AC Type* := AAC

TAA.4

```
IF      [((Terminal Action Code – Denial OR Issuer Action Code – Denial) AND  
        Terminal Verification Results) = '0000000000']  
THEN  
    GOTO TAA.4.1  
ELSE  
    GOTO TAA.5  
ENDIF
```

TAA.4.1

```
IF      [(Terminal Type = '11') OR (Terminal Type = '21') OR (Terminal Type = '14')  
        OR (Terminal Type = '24') OR (Terminal Type = '34')]  
THEN  
    GOTO TAA.4.2  
ELSE  
    GOTO TAA.6  
ENDIF
```

TAA.4.2

'AC type' in *AC Type* := ARQC

TAA.5

'AC type' in *AC Type* := AAC

TAA.6

IF [(*Terminal Type* = '23') OR (*Terminal Type* = '26') OR (*Terminal Type* = '36')
OR (*Terminal Type* = '13') OR (*Terminal Type* = '16')]

THEN

GOTO TAA.13

ELSE

GOTO TAA.7

ENDIF

TAA.7

IF [IsEmpty(TagOf(*Issuer Action Code – Online*))]

THEN

GOTO TAA.10

ELSE

GOTO TAA.8

ENDIF

TAA.8

IF [((*Terminal Action Code – Online* OR 'FFFFFFFFC') AND *Terminal Verification Results*) = '0000000000']

THEN

GOTO TAA.9

ELSE

GOTO TAA.11

ENDIF

TAA.9

'AC type' in *AC Type* := TC

TAA.10

IF [((*Terminal Action Code – Online* OR *Issuer Action Code – Online*) AND
Terminal Verification Results) = '0000000000']

THEN

GOTO TAA.12

ELSE

GOTO TAA.11

ENDIF

TAA.11

'AC type' in *AC Type* := ARQC

TAA.12

'AC type' in *AC Type* := TC

TAA.13

```
IF      [IsNotEmpty(TagOf(Issuer Action Code – Default))]  
THEN  
    GOTO TAA.16  
ELSE  
    GOTO TAA.14  
ENDIF
```

TAA.14

```
IF      [((Terminal Action Code – Default OR 'FFFFFFFFFC') AND Terminal  
        Verification Results) = '0000000000']  
THEN  
    GOTO TAA.15  
ELSE  
    GOTO TAA.17  
ENDIF
```

TAA.15

'AC type' in *AC Type* := TC

TAA.16

```
IF      [((Terminal Action Code – Default OR Issuer Action Code – Default) AND  
        Terminal Verification Results) = '0000000000']  
THEN  
    GOTO TAA.18  
ELSE  
    GOTO TAA.17  
ENDIF
```

TAA.17

'AC type' in *AC Type* := AAC

TAA.18

'AC type' in *AC Type* := TC

8 Security Algorithms

8.1 Unpredictable Number Generation

Random numbers needed by the Kernel (for example for the *Unpredictable Number* and *Unpredictable Number (Numeric)*) should be generated in a hardware Random Number Generator. Any hardware random number generator must be tested in operation according to [NIST SP 800-22A]. A software random number generator must be seeded from an unpredictable source of data. A software whitening process may be applied to a hardware Number Generator if required. Regardless of the method used, there must be no observable correlation from one set of random data to a preceding set of random data and the Terminal must raise a suitable alarm in the event of a random number generator failure. A software Number Generator may be temporarily used until a hardware Number Generator is reinstated.

All values of random number (for example when used as the 4 byte *Unpredictable Number*) must be equally likely to occur, and the value of the random numbers must be unpredictable from the perspective of an attacker (even given knowledge of previous values). This may be achieved using a Random Number Generator compliant with [ISO 18031:2005] and tested using [NIST SP800-22A].

As generation of random data can be a slow process and transaction performance is important, an implementation may consider generating random data before it is needed, for example in a frequently refreshed buffer of random data. If random data is generated ahead of its use it must not be possible to observe this externally and thus to predict all or part of a number that may be used for a specific transaction.

The random number generator must not be susceptible to external perturbation that might reduce its quality, for example EM fields, glitch or other attacks. It must also not be possible for an attacker to deliberately cause fallback from the hardware RNG to a software one.

8.2 OWHF2¹⁴

OWHF2 is the DES-based variant of the one-way function for computing the digest. OWHF2 computes an 8-byte output R based on an 8-byte input PD.

Let PL be the length in bytes of *DS ID*.

Compute two 6-byte values DSPKL and DSPKR as follows:

$DSPKL[i] := ((DS\ ID\ [i] \div 16) \times 10 + (DS\ ID\ [i] \bmod 16)) \times 2$, for $i = 1, 2, \dots, 6$

$DSPKR[i] := ((DS\ ID\ [PL - 6 + i] \div 16) \times 10 + (DS\ ID\ [PL - 6 + i] \bmod 16)) \times 2$, for $i = 1, 2, \dots, 6$

Compute an 8 byte value OID as follows:

IF [IsEmpty(TagOf(*DS Slot Management Control*)) AND
'Permanent slot type' in *DS Slot Management Control* is set AND
'Volatile slot type' in *DS ODS Info* is set]

THEN

OID := '0000000000000000'

ELSE

OID := *DS Requested Operator ID*

ENDIF

Generate two DES keys KL and KR as follows:

$KL[i] := DSPKL[i]$, for $i = 1, 2, \dots, 6$

$KL[i] := OID\ [i - 2]$, for $i = 7, \dots, 8$

$KR[i] := DSPKR[i]$, for $i = 1, 2, \dots, 6$

$KR[i] := OID[i]$, for $i = 7, \dots, 8$

Compute R as follows:

$R := DES(KL)[DES^{-1}(KR)[DES(KL)[OID \oplus PD]]] \oplus PD$

¹⁴ OWHF2 is a function specific to IDS and is only implemented for the DE/DS Implementation Option.

8.3 OWHF2AES¹⁵

OWHF2AES is the AES-based variant of the one-way function for computing the digest. OWHF2AES computes an 8-byte output R based on an 8-byte input C.

Compute an 8 byte value OID as follows:

```
IF      [IsNotEmpty(TagOf(DS Slot Management Control)) AND  
        'Permanent slot type' in DS Slot Management Control is set AND  
        'Volatile slot type' in DS ODS Info is set]  
THEN  
    OID := '0000000000000000'  
ELSE  
    OID := DS Requested Operator ID  
ENDIF
```

Compute R as follows:

Create a 16-byte message M by concatenating the following data:

$M := C \parallel \text{OID}$

Create an 11-byte value Y by padding *DS ID* to the left with zeroes up to 11 bytes

Create a 16-byte AES key K by concatenating the following data:

$K := Y \parallel \text{OID}[5..8] \parallel \text{'3F'}$

Compute a 16-byte value T as follows:

$T := \text{AES}(K)[M] \oplus M$

Compute R as the 8 leftmost bytes from T

¹⁵ OWHF2AES is a function specific to IDS and is only implemented for the DE/DS Implementation Option.

Annex A Data Dictionary

This section contains the data dictionary of the Kernel. It lists all the data objects known to the Kernel other than local working variables.

A.1 Data Objects by Name

A.1.1 Account Type

Tag:	'5F57'
Template:	—
Length:	1
Format:	n 2
Update:	K/ACT/DET
Description:	Indicates the type of account selected on the Terminal, coded as specified in Annex G of [EMV Book 3].

A.1.2 Acquirer Identifier

Tag:	'9F01'
Template:	—
Length:	6
Format:	n 6-11
Update:	K
Description:	Uniquely identifies the acquirer within each payment system.

A.1.3 Active AFL

Tag:	—
Template:	—
Length:	var. up to 248
Format:	b
Update:	K
Description:	Contains the AFL indicating the (remaining) terminal file records to be read from the Card. The <i>Active AFL</i> is updated after each successful READ RECORD.

A.1.4 Active Tag

Tag: —
 Template: —
 Length: var. up to 2
 Format: b
 Update: K
 Description: Contains the tag requested by the GET DATA command.

A.1.5 AC Type

Tag: —
 Template: —
 Length: 1
 Format: b
 Update: K
 Description: Contains the AC type to be requested from the Card with the GENERATE AC command. This is the outcome of Terminal Action Analysis.

AC Type			
Byte 1	b8-7	AC type	
			00: AAC
			01: TC
			10: ARQC
			11: RFU
	b6-1	Each bit RFU	

A.1.6 Additional Terminal Capabilities

Tag: '9F40'
 Template: —
 Length: 5
 Format: b
 Update: K
 Description: Indicates the data input and output capabilities of the Terminal and Reader.

Additional Terminal Capabilities		
Byte 1	b8	Cash
	b7	Goods
	b6	Services
	b5	Cashback
	b4	Inquiry
	b3	Transfer
	b2	Payment
	b1	Administrative
Byte 2	b8	Cash Deposit
	b7-1	Each bit RFU
Byte 3	b8	Numeric keys
	b7	Alphabetical and special characters keys
	b6	Command keys
	b5	Function keys
	b4-1	Each bit RFU
Byte 4	b8	Print, attendant
	b7	Print, cardholder
	b6	Display, attendant
	b5	Display, cardholder
	b4-3	Each bit RFU
	b2	Code table 10
	b1	Code table 9
Byte 5	b8	Code table 8
	b7	Code table 7
	b6	Code table 6
	b5	Code table 5
	b4	Code table 4
	b3	Code table 3
	b2	Code table 2
	b1	Code table 1

A.1.7 Amount, Authorized (Numeric)

Tag:	'9F02'
Template:	—
Length:	6
Format:	n 12
Update:	K/ACT/DET
Description:	<p>Authorized amount of the transaction (excluding adjustments).</p> <p>This amount is expressed with implicit decimal point corresponding to the minor unit of currency as defined by [ISO 4217] (for example the six bytes '00 00 00 00 01 23' represent USD 1.23 when the currency code is '840').</p> <p>If the initial transaction amount needs to be replaced with a revised transaction amount, the Terminal must provide it before the chokepoint.</p>

A.1.8 Amount, Other (Numeric)

Tag:	'9F03'
Template:	—
Length:	6
Format:	n 12
Update:	K/ACT/DET
Description:	<p>Secondary amount associated with the transaction representing a cash back amount.</p> <p>This amount is expressed with implicit decimal point corresponding to the minor unit of currency as defined by [ISO 4217] (for example the 6 bytes '00 00 00 00 01 23' represent GBP 1.23 when the currency code is '826').</p>

A.1.9 Application Capabilities Information

Tag: '9F5D'
 Template: 'BF0C'
 Length: 3
 Format: b
 Update: K/RA
 Description: Lists a number of card features beyond regular payment.

Application Capabilities Information		
Byte 1	b8-5	ACI Version number
		0000: VERSION 0
		Other values: RFU
	b4-1	Data Storage Version Number
		0000: DATA STORAGE NOT SUPPORTED
		0001: VERSION 1
		0010: VERSION 2
		Other values: RFU
Byte 2	b8-4	Each bit RFU
	b3	Support for field off detection
	b2	RFU
	b1	CDA Indicator
		0: CDA SUPPORTED AS IN EMV
		1: CDA SUPPORTED OVER TC, ARQC AND AAC

Application Capabilities Information		
Byte 3	b8-1	SDS Scheme Indicator
		00000000: Undefined SDS configuration
		00000001: All 10 tags 32 bytes
		00000010: All 10 tags 48 bytes
		00000011: All 10 tags 64 bytes
		00000100: All 10 tags 96 bytes
		00000101: All 10 tags 128 bytes
		00000110: All 10 tags 160 bytes
		00000111: All 10 tags 192 bytes
		00001000: All SDS tags 32 bytes except '9F78' which is 64 bytes
		Other values: RFU

A.1.10 Application Cryptogram

Tag: '9F26'
 Template: '77'
 Length: 8
 Format: b
 Update: K/RA
 Description: Cryptogram returned by the Card in response to the GENERATE AC command.

A.1.11 Application Currency Code

Tag: '9F42'
 Template: '70' or '77'
 Length: 2
 Format: n 3
 Update: K/RA
 Description: Indicates the currency in which the account is managed in accordance with [ISO 4217].

A.1.12 Application Currency Exponent

Tag:	'9F44'
Template:	'70' or '77'
Length:	1
Format:	n 1
Update:	K/RA
Description:	Indicates the implied position of the decimal point from the right of the amount represented in accordance with [ISO 4217].

A.1.13 Application Effective Date

Tag:	'5F25'
Template:	'70' or '77'
Length:	3
Format:	n 6 (YYMMDD)
Update:	K/RA
Description:	Date from which the application may be used. The date is expressed in the YYMMDD format.

A.1.14 Application Expiration Date

Tag:	'5F24'
Template:	'70' or '77'
Length:	3
Format:	n 6 (YYMMDD)
Update:	K/RA
Description:	Date after which application expires. The date is expressed in the YYMMDD format.

A.1.15 Application File Locator

Tag:	'94'
Template:	'77'
Length:	var. multiple of 4 between 4 and 248
Format:	b
Update:	K/RA
Description:	Indicates the location (SFI range of records) of the Application Elementary Files associated with a particular AID, and read by the Kernel during a transaction.

The *Application File Locator* is a list of entries of 4 bytes each. Each entry codes an SFI and a range of records as follows:

- The five most significant bits of the first byte indicate the SFI.
- The second byte indicates the first (or only) record number to be read for that SFI.
- The third byte indicates the last record number to be read for that SFI. When the third byte is greater than the second byte, all the records ranging from the record number in the second byte to and including the record number in the third byte must be read for that SFI. When the third byte is equal to the second byte, only the record number coded in the second byte must be read for that SFI.
- The fourth byte indicates the number of records involved in offline data authentication starting with the record number coded in the second byte. The fourth byte may range from zero to the value of the third byte less the value of the second byte plus 1.

A.1.16 Application Interchange Profile

Tag: '82'
 Template: '77'
 Length: 2
 Format: b
 Update: K/RA
 Description: Indicates the capabilities of the Card to support specific functions in the application.
 The *Application Interchange Profile* is returned in the response message of the GET PROCESSING OPTIONS command.

Application Interchange Profile		
Byte 1	b8	RFU
	b7	SDA Supported
	b6	DDA supported
	b5	Cardholder verification is supported
	b4	Terminal risk management is to be performed
	b3	Issuer Authentication is supported
	b2	On device cardholder verification is supported
	b1	CDA supported
Byte 2	b8	EMV mode is supported
	b7-2	Each bit RFU
	b1	Relay resistance protocol is supported

A.1.17 Application Label

Tag: '50'
 Template: 'A5'
 Length: var. up to 16
 Format: ans
 Update: K/RA
 Description: Name associated with the AID, in accordance with [ISO/IEC 7816-5].

A.1.18 Application Preferred Name

Tag:	'9F12'
Template:	'A5'
Length:	var. up to 16
Format:	ans
Update:	K/RA
Description:	Preferred name associated with the AID.

A.1.19 Application PAN

Tag:	'5A'
Template:	'70' or '77'
Length:	var. up to 10
Format:	cn var. up to 19
Update:	K/RA
Description:	Valid cardholder account number.

A.1.20 Application PAN Sequence Number

Tag:	'5F34'
Template:	'70' or '77'
Length:	1
Format:	n 2
Update:	K/RA
Description:	Identifies and differentiates cards with the same <i>Application PAN</i> .

A.1.21 Application Priority Indicator

Tag:	'87'
Template:	'A5'
Length:	1
Format:	b
Update:	K/RA
Description:	Indicates the priority of a given application or group of applications in a directory.

A.1.22 Application Transaction Counter

Tag: '9F36'
 Template: '77'
 Length: 2
 Format: b
 Update: K/RA
 Description: Counter maintained by the application in the Card (incrementing the *Application Transaction Counter* is managed by the Card).

A.1.23 Application Usage Control

Tag: '9F07'
 Template: '70' or '77'
 Length: 2
 Format: b
 Update: K/RA
 Description: Indicates the issuer's specified restrictions on the geographic use and services allowed for the application.

Application Usage Control		
Byte 1	b8	Valid for domestic cash transactions
	b7	Valid for international cash transactions
	b6	Valid for domestic goods
	b5	Valid for international goods
	b4	Valid for domestic services
	b3	Valid for international services
	b2	Valid at ATMs
	b1	Valid at terminals other than ATMs
Byte 2	b8	Domestic cashback allowed
	b7	International cashback allowed
	b6-1	Each bit RFU

A.1.24 Application Version Number (Card)

Tag:	'9F08'
Template:	'70' or '77'
Length:	2
Format:	b
Update:	K/RA
Description:	Version number assigned by the payment system for the application in the Card.

A.1.25 Application Version Number (Reader)

Tag:	'9F09'
Template:	—
Length:	2
Format:	b
Update:	K
Description:	Version number assigned by the payment system for the Kernel application.

A.1.26 CA Public Key Index (Card)

Tag:	'8F'
Template:	'70' or '77'
Length:	1
Format:	b
Update:	K/RA
Description:	Identifies the CA public key in conjunction with the RID.

A.1.27 Card Data Input Capability

Tag: 'DF8117'
 Template: —
 Length: 1
 Format: b
 Update: K
 Description: Indicates the card data input capability of the Terminal and Reader.

Card Data Input Capability		
Byte 1	b8	Manual key entry
	b7	Magnetic stripe
	b6	IC with contacts
	b5-1	Each bit RFU

A.1.28 CDOL1

Tag: '8C'
 Template: '70' or '77'
 Length: var. up to 250
 Format: b
 Update: K/RA
 Description: A data object in the Card that provides the Kernel with a list of data objects that must be passed to the Card in the data field of the GENERATE AC command.

A.1.29 CDOL1 Related Data

Tag: 'DF8107'
 Template: —
 Length: var.
 Format: b
 Update: K
 Description: Command data field of the GENERATE AC command, coded according to *CDOL1*.

A.1.30 Cryptogram Information Data

Tag:	'9F27'
Template:	'77'
Length:	1
Format:	b
Update:	K/RA
Description:	Indicates the type of cryptogram and the actions to be performed by the Kernel. The <i>Cryptogram Information Data</i> is coded according to Table 14 of [EMV Book 3].

A.1.31 CVC3 (Track1)

Tag:	'9F60'
Template:	'77'
Length:	2
Format:	b
Update:	K/RA
Description:	The <i>CVC3 (Track1)</i> is a 2-byte cryptogram returned by the Card in the response to the COMPUTE CRYPTOGRAPHIC CHECKSUM command.

A.1.32 CVC3 (Track2)

Tag:	'9F61'
Template:	'77'
Length:	2
Format:	b
Update:	K/RA
Description:	The <i>CVC3 (Track2)</i> is a 2-byte cryptogram returned by the Card in the response to the COMPUTE CRYPTOGRAPHIC CHECKSUM command.

A.1.33 CVM Capability – CVM Required

Tag: 'DF8118'
 Template: —
 Length: 1
 Format: b
 Update: K
 Description: Indicates the CVM capability of the Terminal and Reader when the transaction amount is greater than the *Reader CVM Required Limit*.

CVM Capability – CVM Required		
Byte 1	b8	Plaintext PIN for ICC verification
	b7	Enciphered PIN for online verification
	b6	Signature (paper)
	b5	Enciphered PIN for offline verification
	b4	No CVM required
	b3-1	Each bit RFU

A.1.34 CVM Capability – No CVM Required

Tag: 'DF8119'
 Template: —
 Length: 1
 Format: b
 Update: K
 Description: Indicates the CVM capability of the Terminal and Reader when the transaction amount is less than or equal to the *Reader CVM Required Limit*.

CVM Capability – No CVM Required		
Byte 1	b8	Plaintext PIN for ICC verification
	b7	Enciphered PIN for online verification
	b6	Signature (paper)
	b5	Enciphered PIN for offline verification
	b4	No CVM required
	b3-1	Each bit RFU

A.1.35 CVM List

Tag: '8E'
Template: '70' or '77'
Length: var. 10 to 250
Format: b
Update: K/RA
Description: Identifies the methods of verification of the cardholder supported by the application.
The *CVM List* is coded as specified in section 10.5 of [EMV Book 3].

A.1.36 CVM Results

Tag: '9F34'
Template: —
Length: 3
Format: b
Update: K
Description: Indicates the results of the last CVM performed.
The *CVM Results* are coded as specified in Annex A.4 of [EMV Book 4].

CVM Results		
Byte 1	b8-1	CVM Performed
Byte 2	b8-1	CVM Condition
Byte 3	b8-1	CVM Result

A.1.37 Data Needed

Tag: 'DF8106'
Template: —
Length: var.
Format: b
Update: K
Description: List of tags included in the DEK Signal to request information from the Terminal.

A.1.38 Data Record

Tag:	'FF8105'
Template:	—
Length:	var.
Format:	b
Update:	K
Description:	The <i>Data Record</i> is a list of TLV encoded data objects returned with the <i>Outcome Parameter Set</i> on the completion of transaction processing.

A.1.39 Data To Send

Tag:	'FF8104'
Template:	—
Length:	var.
Format:	b
Update:	K
Description:	List of data objects that contains the accumulated data sent by the Kernel to the Terminal in a DEK Signal. These data may correspond to Terminal reading requests, obtained from the Card by means of GET DATA or READ RECORD commands, or may correspond to data that the Kernel posts to the Terminal as part of its own processing.

A.1.40 DD Card (Track1)

Tag:	'DF812A'
Template:	—
Length:	var. up to 56
Format:	ans
Update:	K
Description:	If <i>Track 1 Data</i> is present, then <i>DD Card (Track1)</i> contains a copy of the discretionary data field of <i>Track 1 Data</i> as returned by the Card in the file read using the READ RECORD command during a Mag-stripe Mode transaction (i.e. without <i>Unpredictable Number (Numeric)</i> , <i>Application Transaction Counter</i> , <i>CVC3 (Track1)</i> and <i>nUN</i> included).

A.1.41 DD Card (Track2)

Tag:	'DF812B'
Template:	—
Length:	var. up to 11 bytes
Format:	cn
Update:	K
Description:	<i>DD Card (Track2)</i> contains a copy of the discretionary data field of <i>Track 2 Data</i> as returned by the Card in the file read using the READ RECORD command during a Mag-stripe Mode transaction (i.e. without <i>Unpredictable Number (Numeric)</i> , <i>Application Transaction Counter</i> , <i>CVC3 (Track2)</i> and <i>nUN</i> included).

A.1.42 Default UDOL

Tag:	'DF811A'
Template:	—
Length:	3
Format:	b
Update:	K
Description:	The <i>Default UDOL</i> is the <i>UDOL</i> to be used for constructing the value field of the COMPUTE CRYPTOGRAPHIC CHECKSUM command if the <i>UDOL</i> in the Card is not present. The <i>Default UDOL</i> must contain as its only entry the tag and length of the <i>Unpredictable Number (Numeric)</i> and has the value: '9F6A04'.

A.1.43 Device Estimated Transmission Time For Relay Resistance R-APDU

Tag:	'DF8305'
Template:	—
Length:	2
Format:	b
Update:	K/RA
Description:	Indicates the time the Card expects to need for transmitting the EXCHANGE RELAY RESISTANCE DATA R-APDU. The <i>Device Estimated Transmission Time For Relay Resistance R-APDU</i> is expressed in units of hundreds of microseconds.

A.1.44 Device Relay Resistance Entropy

Tag:	'DF8302'
Template:	—
Length:	4
Format:	b
Update:	K/RA
Description:	Random number returned by the Card in the response to the EXCHANGE RELAY RESISTANCE DATA command.

A.1.45 DF Name

Tag:	'84'
Template:	'6F'
Length:	5-16
Format:	b
Update:	K/RA
Description:	Identifies the name of the DF, as described in [ISO 7816-4].

A.1.46 Discretionary Data

Tag:	'FF8106'
Template:	—
Length:	var.
Format:	b
Update:	K
Description:	The <i>Discretionary Data</i> is a list of Kernel-specific data objects sent to the Terminal as a separate field in the OUT Signal.

A.1.47 DS AC Type

Tag: 'DF8108'
 Template: —
 Length: 1
 Format: b
 Update: K/ACT/DET
 Description: Contains the AC type indicated by the Terminal for which IDS data must be stored in the Card.

DS AC Type			
Byte 1	b8-7	AC type	
			00: AAC
			01: TC
			10: ARQC
			11: RFU
	b6-1	Each bit RFU	

A.1.48 DS Digest H

Tag: 'DF61'
 Template: —
 Length: 8
 Format: b
 Update: K
 Description: Contains the result of OWHF2(*DS Input (Term)*) or OWHF2AES(*DS Input (Term)*), if *DS Input (Term)* is provided by the Terminal.
 This data object is to be supplied to the Card with the GENERATE AC command, as per *DSDOL* formatting.

A.1.49 DSDOL

Tag:	'9F5B'
Template:	'70'
Length:	var. up to 250
Format:	b
Update:	K/RA
Description:	<p>A data object in the Card that provides the Kernel with a list of data objects that must be passed to the Card in the data field of the GENERATE AC command after the <i>CDOL1 Related Data</i>.</p> <p>An example of value for <i>DSDOL</i> is 'DF6008DF6108DF6201DF63A0', representing <i>TLDS Input (Card) TLDS Digest H TLDS ODS Info TLDS ODS Term</i>.</p> <p>The Kernel must not presume that this is a given though, as the sequence and presence of data objects can vary.</p> <p>The presence of <i>TL DS ODS Info</i> is mandated and the processing of the last TL entry in <i>DSDOL</i> is different from normal TL processing as described in section 4.1.4.</p>

A.1.50 DS ID

Tag:	'9F5E'
Template:	'BF0C'
Length:	var. 8 to 11
Format:	n, 16 to 22
Update:	K/RA
Description:	<p>Data Storage Identifier constructed as follows:</p> <p><i>Application PAN</i> (without any 'F' padding) <i>Application PAN Sequence Number</i></p> <p>If necessary, it is padded to the left with one hexadecimal zero to ensure whole bytes.</p> <p>If necessary, it is padded to the left with hexadecimal zeroes to ensure a minimum length of 8 bytes.</p>

A.1.51 DS Input (Card)

Tag:	'DF60'
Template:	—
Length:	8
Format:	b
Update:	K/ACT/DET
Description:	Contains Terminal provided data if permanent data storage in the Card was applicable (<i>DS Slot Management Control</i> [8]=1b), remains applicable, or becomes applicable (<i>DS ODS Info</i> [8]=1b). Otherwise this data item is a filler to be supplied by the Kernel. The data is forwarded to the Card with the GENERATE AC command, as per <i>DSDOL</i> formatting.

A.1.52 DS Input (Term)

Tag:	'DF8109'
Template:	—
Length:	8
Format:	b
Update:	K/ACT/DET
Description:	Contains Terminal provided data if permanent data storage in the Card was applicable (<i>DS Slot Management Control</i> [8]=1b), remains applicable or becomes applicable (<i>DS ODS Info</i> [8]=1b). <i>DS Input (Term)</i> is used by the Kernel as input to calculate <i>DS Digest H</i> .

A.1.53 DS ODS Card

Tag:	'9F54'
Template:	'77'
Length:	var. up to 160
Format:	b
Update:	K/RA
Description:	Contains the Card stored operator proprietary data obtained in the response to the GET PROCESSING OPTIONS command.

A.1.54 DS ODS Info

Tag:	'DF62'
Template:	—
Length:	1
Format:	b
Update:	K/ACT/DET
Description:	Contains Terminal provided data to be forwarded to the Card with the GENERATE AC command, as per <i>DSDOL</i> formatting.

DS ODS Info		
Byte 1	b8	Permanent slot type
	b7	Volatile slot type
	b6	Low volatility
	b5	RFU
	b4	Decline payment transaction in case of data storage error
	b3-1	Each bit RFU

A.1.55 DS ODS Info For Reader

Tag:	'DF810A'
Template:	—
Length:	1
Format:	b
Update:	K/ACT/DET
Description:	<p>Contains instructions from the Terminal on how to proceed with the transaction if:</p> <ul style="list-style-type: none"> • The AC requested by the Terminal does not match the AC proposed by the Kernel • The update of the slot data has failed

DS ODS Info For Reader		
Byte 1	b8	Usable for TC
	b7	Usable for ARQC
	b6	Usable for AAC
	b5-4	Each bit RFU
	b3	Stop if no DS ODS Term
	b2	Stop if write failed
	b1	RFU

A.1.56 DS ODS Term

Tag: 'DF63'
 Template: —
 Length: var. up to 160
 Format: b
 Update: K/ACT/DET
 Description: Contains Terminal provided data to be forwarded to the Card with the GENERATE AC command, as per *DSDOL* formatting.

A.1.57 DS Requested Operator ID

Tag: '9F5C'
 Template: —
 Length: 8
 Format: b
 Update: K/ACT/DET
 Description: Contains the Terminal determined operator identifier for data storage. It is sent to the Card in the GET PROCESSING OPTIONS command.

A.1.58 DS Slot Availability

Tag: '9F5F'
 Template: '77'
 Length: 1
 Format: b
 Update: K/RA
 Description: Contains the Card indication, obtained in the response to the GET PROCESSING OPTIONS command, about the slot type(s) available for data storage.

DS Slot Availability		
Byte 1	b8	Permanent slot type
	b7	Volatile slot type
	b6-1	Each bit RFU

A.1.59 DS Slot Management Control

Tag: '9F6F'
 Template: '77'
 Length: 1
 Format: b
 Update: K/RA
 Description: Contains the Card indication, obtained in the response to the GET PROCESSING OPTIONS command, about the status of the slot containing data associated to the *DS Requested Operator ID*.

DS Slot Management Control		
Byte 1	b8	Permanent slot type
	b7	Volatile slot type
	b6	Low volatility
	b5	Locked slot
	b4-2	Each bit RFU
	b1	Deactivated slot

A.1.60 DS Summary 1

Tag:	'9F7D'
Template:	'77'
Length:	8 or 16
Format:	b
Update:	K/RA
Description:	Contains the Card indication, obtained in the response to the GET PROCESSING OPTIONS command, about either the stored summary associated with <i>DS ODS Card</i> if present, or about a default zero-filled summary if <i>DS ODS Card</i> is not present and <i>DS Unpredictable Number</i> is present.

A.1.61 DS Summary 2

Tag:	'DF8101'
Template:	—
Length:	8 or 16
Format:	b
Update:	K/RA
Description:	<p>This data allows the Kernel to check the consistency between <i>DS Summary 1</i> and <i>DS Summary 2</i>, and so to ensure that <i>DS ODS Card</i> is provided by a genuine Card.</p> <p>It is located in the ICC Dynamic Data recovered from the <i>Signed Dynamic Application Data</i>.</p>

A.1.62 DS Summary 3

Tag:	'DF8102'
Template:	—
Length:	8 or 16
Format:	b
Update:	K/RA
Description:	<p>This data allows the Kernel to check whether the Card has seen the same transaction data as were sent by the Terminal/Kernel.</p> <p>It is located in the ICC Dynamic Data recovered from the <i>Signed Dynamic Application Data</i>.</p>

A.1.63 DS Summary Status

Tag: 'DF810B'
Template: —
Length: 1
Format: b
Update: K
Description: Information reported by the Kernel to the Terminal about:

- The consistency between *DS Summary 1* and *DS Summary 2* (successful read)
- The difference between *DS Summary 2* and *DS Summary 3* (successful write)

This data object is part of the *Discretionary Data*.

DS Summary Status		
Byte 1	b8	Successful Read
	b7	Successful Write
	b6-1	Each bit RFU

A.1.64 DS Unpredictable Number

Tag: '9F7F'
Template: '77'
Length: 4
Format: b
Update: K/RA
Description: Contains the Card challenge (random), obtained in the response to the GET PROCESSING OPTIONS command, to be used by the Terminal in the summary calculation when providing *DS ODS Term*.

A.1.65 DSVN Term

Tag: 'DF810D'

Template: —

Length: var.

Format: b

Update: K

Description: Integrated data storage support by the Kernel depends on the presence of this data object. If it is absent, or is present with a length of zero, integrated data storage is not supported.

Its value is '02' for this version of data storage functionality.

This variable length data item has an initial byte that defines the maximum version number supported by the Terminal and a variable number of subsequent bytes that define how the Terminal supports earlier versions of the specification. As this is the first version, no legacy support is described and no additional bytes are present.

A.1.66 Error Indication

Tag: 'DF8115'

Template: —

Length: 6

Format: b

Update: K

Description: Contains information regarding the nature of the error that has been encountered during the transaction processing.

This data object is part of the *Discretionary Data*.

Data Field	Length	Format
L1	1	b (see below)
L2	1	b (see below)
L3	1	b (see below)
SW12	2	b
Msg On Error	1	b (see Message Identifier as defined in A.1.186)

L1		
Byte 1	b8-1	L1
		00000000: OK
		00000001: TIME OUT ERROR
		00000010: TRANSMISSION ERROR
		00000011: PROTOCOL ERROR
		Other values: RFU

L2		
Byte 1	b8-1	L2
		00000000: OK
		00000001: CARD DATA MISSING
		00000010: CAM FAILED
		00000011: STATUS BYTES
		00000100: PARSING ERROR
		00000101: MAX LIMIT EXCEEDED
		00000110: CARD DATA ERROR
		00000111: MAGSTRIPE NOT SUPPORTED
		00001000: NO PPSE
		00001001: PPSE FAULT
		00001010: EMPTY CANDIDATE LIST
		00001011: IDS READ ERROR
		00001100: IDS WRITE ERROR
		00001101: IDS DATA ERROR
		00001110: IDS NO MATCHING AC
		00001111: TERMINAL DATA ERROR
		Other values: RFU

L3		
Byte 1	b8-1	L3
		00000000: OK
		00000001: TIME OUT
		00000010: STOP
		00000011: AMOUNT NOT PRESENT
		Other values: RFU

A.1.67 Failed MS Cntr

Tag: —

Template: —

Length: 1

Format: b

Update: K

Description: Counts the number of failed consecutive Mag-stripe Mode transactions. The *Failed MS Cntr* is stored in the scratch pad provided to the Kernel at instantiation.

A.1.68 File Control Information Issuer Discretionary Data

Tag: 'BF0C'

Template: 'A5'

Length: var. up to 220

Format: b

Update: K/RA

Description: Issuer discretionary part of the *File Control Information Proprietary Template*.

A.1.69 File Control Information Proprietary Template

Tag:	'A5'
Template:	'6F'
Length:	var. up to 240
Format:	b
Update:	K/RA
Description:	Identifies the data object proprietary to this specification in the <i>File Control Information Template</i> , in accordance with [ISO 7816-4].

A.1.70 File Control Information Template

Tag:	'6F'
Template:	—
Length:	var. up to 250
Format:	b
Update:	K/RA
Description:	Identifies the <i>File Control Information Template</i> , in accordance with [ISO 7816-4]. The table below gives the <i>File Control Information Template</i> expected in response to a successful selection of a Card application matching Kernel 2.

Tag	Value		Presence
'6F'	<i>File Control Information Template</i>		M
	'84'	<i>DF Name (AID)</i>	M
	'A5'	<i>File Control Information Proprietary Template</i>	M ¹⁶
		'50' <i>Application Label</i>	O
		'87' <i>Application Priority Indicator</i>	O
		'5F2D' <i>Language Preference</i>	O
		'9F38' <i>PDOL</i>	O
		'9F11' <i>Issuer Code Table Index</i>	O
		'9F12' <i>Application Preferred Name</i>	O
		'BF0C' <i>File Control Information Issuer Discretionary Data</i>	O
		'9F6E' <i>Third Party Data</i>	O
		'9F5D' <i>Application Capabilities Information</i>	O
		'XXXX' One or more additional data objects from application provider, Issuer, or ICC supplier	O

A.1.71 Hold Time Value

Tag: 'DF8130'

Template: —

Length: 1

Format: b

Update: K

Description: Indicates the time that the field is to be turned off after the transaction is completed if requested to do so by the cardholder device. The *Hold Time Value* is in units of 100ms.

¹⁶The *File Control Information Proprietary Template* may be empty. In this case the length must be set to zero.

A.1.72 ICC Dynamic Number

Tag:	'9F4C'
Template:	—
Length:	var. 2 to 8
Format:	b
Update:	K/RA
Description:	Time-variant number generated by the Card, to be captured by the Kernel.

A.1.73 ICC Public Key Certificate

Tag:	'9F46'
Template:	'70' or '77'
Length:	N_i (var. up to 248)
Format:	b
Update:	K/RA
Description:	ICC public key certified by the issuer.

A.1.74 ICC Public Key Exponent

Tag:	'9F47'
Template:	'70' or '77'
Length:	1 or 3
Format:	b
Update:	K/RA
Description:	Exponent used for the verification of the <i>Signed Dynamic Application Data</i> .

A.1.75 ICC Public Key Remainder

Tag:	'9F48'
Template:	'70' or '77'
Length:	var.
Format:	b
Update:	K/RA
Description:	Remaining digits of the modulus of the ICC public key.

A.1.76 IDS Status

Tag: 'DF8128'
Template: —
Length: 1
Format: b
Update: K
Description: Indicates if the transaction performs an IDS read and/or write.

IDS Status		
Byte 1	b8	Read
	b7	Write
	b6-1	Each bit RFU

A.1.77 Interface Device Serial Number

Tag: '9F1E'
Template: —
Length: 8
Format: an
Update: K
Description: Unique and permanent serial number assigned to the IFD by the manufacturer.

A.1.78 Issuer Action Code – Default

Tag: '9F0D'
Template: '70' or '77'
Length: 5
Format: b
Update: K/RA
Description: Specifies the issuer's conditions that cause a transaction to be rejected on an offline only Terminal.

A.1.79 Issuer Action Code – Denial

Tag:	'9F0E'
Template:	'70' or '77'
Length:	5
Format:	b
Update:	K/RA
Description:	Specifies the issuer's conditions that cause the denial of a transaction without any attempt to go online.

A.1.80 Issuer Action Code – Online

Tag:	'9F0F'
Template:	'70' or '77'
Length:	5
Format:	b
Update:	K/RA
Description:	Specifies the issuer's conditions that cause a transaction to be transmitted online on an online capable Terminal.

A.1.81 Issuer Application Data

Tag:	'9F10'
Template:	'77'
Length:	var. up to 32
Format:	b
Update:	K/RA
Description:	Contains proprietary application data for transmission to the issuer in an online transaction.

A.1.82 Issuer Code Table Index

Tag:	'9F11'
Template:	'A5'
Length:	1
Format:	n 2
Update:	K/RA
Description:	Indicates the code table, in accordance with [ISO/IEC 8859], for displaying the <i>Application Preferred Name</i> . The <i>Issuer Code Table Index</i> is coded as specified in Annex C.4 of [EMV Book 3].

A.1.83 Issuer Country Code

Tag:	'5F28'
Template:	'70' or '77'
Length:	2
Format:	n 3
Update:	K/RA
Description:	Indicates the country of the issuer, in accordance with [ISO 3166-1].

A.1.84 Issuer Public Key Certificate

Tag:	'90'
Template:	'70' or '77'
Length:	N _{CA} (var. up to 248)
Format:	b
Update:	K/RA
Description:	Issuer public key certified by a certification authority.

A.1.85 Issuer Public Key Exponent

Tag:	'9F32'
Template:	'70' or '77'
Length:	1 or 3
Format:	b
Update:	K/RA
Description:	Exponent used for the recovery and verification of the <i>ICC Public Key Certificate</i> .

A.1.86 Issuer Public Key Remainder

Tag: '92'
 Template: '70' or '77'
 Length: $N_I - N_{CA} + 36$
 Format: b
 Update: K/RA
 Description: Remaining digits of the modulus of the Issuer public key.

A.1.87 Kernel Configuration

Tag: 'DF811B'
 Template: —
 Length: 1
 Format: b
 Update: K
 Description: Indicates the Kernel configuration options.

Kernel Configuration		
Byte 1	b8	Mag-stripe mode contactless transactions not supported ⁽¹⁾
	b7	EMV mode contactless transactions not supported ⁽¹⁾
	b6	On device cardholder verification supported
	b5	Relay resistance protocol supported
	b4	Reserved for Payment system
	b3	Read all records even when no CDA
	b2-1	Each bit RFU

⁽¹⁾ Not applicable if MAG not implemented

A.1.88 Kernel ID

Tag: 'DF810C'
 Template: —
 Length: 1
 Format: b
 Update: K
 Description: Contains a value that uniquely identifies each Kernel. There is one occurrence of this data object for each Kernel in the Reader.

A.1.89 Language Preference

Tag:	'5F2D'
Template:	'A5'
Length:	2-8
Format:	an
Update:	K/RA
Description:	1-4 languages stored in order of preference, each represented by two alphabetical characters, in accordance with [ISO 639-1].

A.1.90 Log Entry

Tag:	'9F4D'
Template:	'BF0C'
Length:	2
Format:	b
Update:	K/RA
Description:	Provides the SFI of the Transaction Log file and its number of records.

A.1.91 Mag-stripe Application Version Number (Reader)

Tag:	'9F6D'
Template:	—
Length:	2
Format:	b
Update:	K
Description:	Version number assigned by the payment system for the specific Mag-stripe Mode functionality of the Kernel.

A.1.92 Mag-stripe CVM Capability – CVM Required

Tag:	'DF811E'
Template:	—
Length:	1
Format:	b
Update:	K
Description:	Indicates the CVM capability of the Terminal/Reader in the case of a Mag-stripe Mode transaction when the <i>Amount, Authorized (Numeric)</i> is greater than the <i>Reader CVM Required Limit</i> .

Mag-stripe CVM Capability – CVM Required			
Byte 1	b8-5	CVM	
			0000: NO CVM
			0001: OBTAIN SIGNATURE
			0010: ONLINE PIN
			1111: N/A
			Other values: RFU
	b4-1	Each bit RFU	

A.1.93 Mag-stripe CVM Capability – No CVM Required

Tag: 'DF812C'
 Template: —
 Length: 1
 Format: b
 Update: K
 Description: Indicates the CVM capability of the Terminal/Reader in the case of a mag-stripe mode transaction when the *Amount, Authorized (Numeric)* is less than or equal to the *Reader CVM Required Limit*.

Mag-stripe CVM Capability – No CVM Required			
Byte 1	b8-5	CVM	
			0000: NO CVM
			0001: OBTAIN SIGNATURE
			0010: ONLINE PIN
			1111: N/A
			Other values: RFU
	b4-1	Each bit RFU	

A.1.94 Maximum Relay Resistance Grace Period

Tag: 'DF8133'
 Template: —
 Length: 2
 Format: b
 Update: K
 Description: The *Minimum Relay Resistance Grace Period* and *Maximum Relay Resistance Grace Period* represent how far outside the window defined by the Card that the measured time may be and yet still be considered acceptable. The *Maximum Relay Resistance Grace Period* is expressed in units of hundreds of microseconds.

A.1.95 Max Time For Processing Relay Resistance APDU

Tag:	'DF8304'
Template:	—
Length:	2
Format:	b
Update:	K/RA
Description:	Indicates the maximum estimated time the Card requires for processing the EXCHANGE RELAY RESISTANCE DATA command. The <i>Max Time For Processing Relay Resistance APDU</i> is expressed in units of hundreds of microseconds.

A.1.96 Measured Relay Resistance Processing Time

Tag:	'DF8306'
Template:	—
Length:	2
Format:	b
Update:	K
Description:	Contains the time measured by the Kernel for processing the EXCHANGE RELAY RESISTANCE DATA command. The <i>Measured Relay Resistance Processing Time</i> is expressed in units of hundreds of microseconds.

A.1.97 Merchant Category Code

Tag:	'9F15'
Template:	—
Length:	2
Format:	n 4
Update:	K
Description:	Classifies the type of business being done by the merchant, represented in accordance with [ISO 8583:1993] for Card Acceptor Business Code.

A.1.98 Merchant Custom Data

Tag:	'9F7C'
Template:	—
Length:	20
Format:	b
Update:	K/ACT/DET
Description:	Proprietary merchant data that may be requested by the Card.

A.1.99 Merchant Identifier

Tag:	'9F16'
Template:	—
Length:	15
Format:	ans 15
Update:	K
Description:	When concatenated with the <i>Acquirer Identifier</i> , uniquely identifies a given merchant.

A.1.100 Merchant Name and Location

Tag:	'9F4E'
Template:	—
Length:	var.
Format:	ans
Update:	K
Description:	Indicates the name and location of the merchant.

A.1.101 Message Hold Time

Tag:	'DF812D'
Template:	—
Length:	3
Format:	n 6
Update:	K
Description:	Indicates the default delay for the processing of the next MSG Signal. The <i>Message Hold Time</i> is an integer in units of 100ms.

A.1.102 Minimum Relay Resistance Grace Period

Tag:	'DF8132'
Template:	—
Length:	2
Format:	b
Update:	K
Description:	The <i>Minimum Relay Resistance Grace Period</i> and <i>Maximum Relay Resistance Grace Period</i> represent how far outside the window defined by the Card that the measured time may be and yet still be considered acceptable. The <i>Minimum Relay Resistance Grace Period</i> is expressed in units of hundreds of microseconds.

A.1.103 Min Time For Processing Relay Resistance APDU

Tag:	'DF8303'
Template:	—
Length:	2
Format:	b
Update:	K/RA
Description:	Indicates the minimum estimated time the Card requires for processing the EXCHANGE RELAY RESISTANCE DATA command. The <i>Min Time For Processing Relay Resistance APDU</i> is expressed in units of hundreds of microseconds.

A.1.104 Mobile Support Indicator

Tag:	'9F7E'
Template:	—
Length:	1
Format:	b
Update:	K
Description:	The <i>Mobile Support Indicator</i> informs the Card that the Kernel supports extensions for mobile and requires on device cardholder verification.

Mobile Support Indicator		
Byte 1	b8-3	Each bit RFU
	b2	OD-CVM Required
	b1	Mobile supported

A.1.105 NATC(Track1)

Tag: '9F64'
 Template: '70'
 Length: 1
 Format: b
 Update: K/RA
 Description: The value of *NATC(Track1)* represents the number of digits of the *Application Transaction Counter* to be included in the discretionary data field of *Track 1 Data*.

A.1.106 NATC(Track2)

Tag: '9F67'
 Template: '70'
 Length: 1
 Format: b
 Update: K/RA
 Description: The value of *NATC(Track2)* represents the number of digits of the *Application Transaction Counter* to be included in the discretionary data field of *Track 2 Data*.

A.1.107 Next Cmd

Tag:	—
Template:	—
Length:	1
Format:	b
Update:	K
Description:	An internal working variable used to indicate the C-APDU that is currently being processed by the Card.

Next Cmd			
Byte 1	b8-7	Next Cmd	
			00: READ RECORD
			01: GET DATA
			10: NONE
			11: RFU
	b6-1	Each bit RFU	

A.1.108 nUN

Tag:	—
Template:	—
Length:	1
Format:	b
Update:	K
Description:	Number of non-zero bits in $PUNATC(Track2) - NATC(Track2)$

A.1.109 ODA Status

Tag: —
Template: —
Length: 1
Format: b
Update: K
Description: Indicates if CDA is to be performed for the transaction in progress.

ODA Status		
Byte 1	b8	CDA
	b7-1	Each bit RFU

A.1.110 Outcome Parameter Set

Tag:	'DF8129'
Template:	—
Length:	8
Format:	b
Update:	K
Description:	This data object is used to indicate to the Terminal the outcome of the transaction processing by the Kernel. Its value is an accumulation of results about applicable parts of the transaction.

Outcome Parameter Set			
Byte 1	b8-5	Status	
			0001: APPROVED
			0010: DECLINED
			0011: ONLINE REQUEST
			0100: END APPLICATION
			0101: SELECT NEXT
			0110: TRY ANOTHER INTERFACE
			0111: TRY AGAIN
			1111: N/A
			Other values: RFU
	b4-1	Each bit RFU	
Byte 2	b8-5	Start	
			0000: A
			0001: B
			0010: C
			0011: D
			1111: N/A
			Other values: RFU
	b4-1	Each bit RFU	

Outcome Parameter Set			
Byte 3	b8-5	Online Response Data	
			1111: N/A
			Other values: RFU
	b4-1	Each bit RFU	
Byte 4	b8-5	CVM	
			0000: NO CVM
			0001: OBTAIN SIGNATURE
			0010: ONLINE PIN
			0011: CONFIRMATION CODE VERIFIED
			1111: N/A
			Other values: RFU
	b4-1	Each bit RFU	
Byte 5	b8	UI Request on Outcome Present	
	b7	UI Request on Restart Present	
	b6	Data Record Present	
	b5	Discretionary Data Present	
	b4	Receipt	
			0: N/A
			1: YES
	b3-1	Each bit RFU	
Byte 6	b8-5	Alternate Interface Preference	
			1111: N/A
			Other values: RFU
	b4-1	Each bit RFU	
Byte 7	b8-1	Field Off Request	
			11111111: N/A
			Other values: Hold time in units of 100 ms
Byte 8	b8-1	Removal Timeout in units of 100 ms	

A.1.111 Payment Account Reference

Tag:	'9F24'
Template:	'70' or '77'
Length:	29
Format:	an
Update:	K/RA
Description:	<p>The <i>Payment Account Reference</i> is a data object associated with an <i>Application PAN</i>. It allows acquirers and merchants to link transactions, whether tokenised or not, that are associated to the same underlying <i>Application PAN</i>.</p> <p>Lower case alphabetic characters are not permitted for the <i>Payment Account Reference</i>, however the Kernel is not expected to check this.</p>

A.1.112 PCVC3(Track1)

Tag:	'9F62'
Template:	'70'
Length:	6
Format:	b
Update:	K/RA
Description:	<p><i>PCVC3(Track1)</i> indicates to the Kernel the positions in the discretionary data field of the <i>Track 1 Data</i> where the <i>CVC3 (Track1)</i> digits must be copied.</p>

A.1.113 PCVC3(Track2)

Tag:	'9F65'
Template:	'70'
Length:	2
Format:	b
Update:	K/RA
Description:	<p><i>PCVC3(Track2)</i> indicates to the Kernel the positions in the discretionary data field of the <i>Track 2 Data</i> where the <i>CVC3 (Track2)</i> digits must be copied.</p>

A.1.114 PDOL

Tag:	'9F38'
Template:	'A5'
Length:	var. up to 240
Format:	b
Update:	K/RA
Description:	A data object in the Card that provides the Kernel with a list of data objects that must be passed to the Card in the GET PROCESSING OPTIONS command.

A.1.115 PDOL Related Data

Tag:	'DF8111'
Template:	—
Length:	var.
Format:	b
Update:	K
Description:	Command data field of the GET PROCESSING OPTIONS command, coded according to <i>PDOL</i> .

A.1.116 PDOL Values

Tag:	—
Length:	var.
Format:	b
Update:	K
Description:	<i>PDOL Values</i> is the value field of the Command Template (tag '83') stored in <i>PDOL Related Data</i> . <i>PDOL Values</i> is created by the Kernel according to <i>PDOL</i> as a concatenated list of data objects without tags or lengths following the rules specified in section 4.1.4. If <i>PDOL</i> is not returned by the Card, then <i>PDOL Values</i> is an empty byte string.

A.1.117 Phone Message Table

Tag:	'DF8131'
Template:	—
Length:	var.
Format:	b
Update:	K
Description:	The <i>Phone Message Table</i> is a variable length list of entries of eight bytes each, and defines for the selected AID the message and status identifiers as a function of the <i>POS Cardholder Interaction Information</i> . Each entry in the <i>Phone Message Table</i> contains the fields shown in the table below.

Data Field	Length	Format
PCII Mask	3	b
PCII Value	3	b
Message Identifier	1	b
Status	1	b

Note that the last entry in the *Phone Message Table* must always have PCII Mask and PCII Value set to '000000'.

A.1.118 POS Cardholder Interaction Information

Tag: 'DF4B'
 Template: '77'
 Length: 3
 Format: b
 Update: K/RA
 Description: The *POS Cardholder Interaction Information* informs the Kernel about the indicators set in the mobile phone that may influence the action flow of the merchant and cardholder.

POS Cardholder Interaction Information		
Byte 1	b8-1	Version Number
Byte 2	b8-6	Each bit RFU
	b5	OD-CVM verification successful
	b4	Context is conflicting
	b3	Offline change PIN required
	b2	ACK required
	b1	OD-CVM required
Byte 3	b8-2	Each bit RFU
	b1	Wallet requires second tap

A.1.119 Post-Gen AC Put Data Status

Tag:	'DF810E'
Template:	—
Length:	1
Format:	b
Update:	K
Description:	<p>Information reported by the Kernel to the Terminal, about the processing of PUT DATA commands after processing the GENERATE AC command.</p> <p>Possible values are 'completed' or 'not completed'. In the latter case, this status is not specific about which of the PUT DATA commands failed, or about how many of these commands have failed or succeeded.</p> <p>This data object is part of the <i>Discretionary Data</i> provided by the Kernel to the Terminal.</p>

Post-Gen AC Put Data Status		
Byte 1	b8	Completed
	b7-1	Each bit RFU

A.1.120 Pre-Gen AC Put Data Status

Tag: 'DF810F'

Template: —

Length: 1

Format: b

Update: K

Description: Information reported by the Kernel to the Terminal, about the processing of PUT DATA commands before sending the GENERATE AC command.

Possible values are 'completed' or 'not completed'. In the latter case, this status is not specific about which of the PUT DATA commands failed, or about how many of these commands have failed or succeeded.

This data object is part of the *Discretionary Data* provided by the Kernel to the Terminal.

Pre-Gen AC Put Data Status		
Byte 1	b8	Completed
	b7-1	Each bit RFU

A.1.121 Proceed To First Write Flag

Tag:	'DF8110'
Template:	—
Length:	1
Format:	b
Update:	K/ACT/DET
Description:	<p>Indicates that the Terminal will send no more requests to read data other than as indicated in <i>Tags To Read</i>. This data item indicates the point at which the Kernel shifts from the Card reading phase to the Card writing phase.</p> <p>If <i>Proceed To First Write Flag</i> is not present or is present with non zero length and value different from zero, then the Kernel proceeds without waiting.</p> <p>If <i>Proceed To First Write Flag</i> is present with zero length, then the Kernel sends a DEK Signal to the Terminal and waits for the DET Signal.</p> <p>If <i>Proceed To First Write Flag</i> is present with non zero length and value equal to zero, then the Kernel waits for a DET Signal from the Terminal without sending a DEK Signal.</p>

A.1.122 Protected Data Envelope 1

Tag:	'9F70'
Template:	—
Length:	var. up to 192
Format:	b
Update:	K/RA/ACT/DET
Description:	<p>The Protected Data Envelopes contain proprietary information from the issuer, payment system or third party. The Protected Data Envelope can be retrieved with the GET DATA command. Updating the Protected Data Envelope with the PUT DATA command requires secure messaging and is outside the scope of this specification.</p>

A.1.123 Protected Data Envelope 2

Tag: '9F71'
Template: —
Length: var. up to 192
Format: b
Update: K/RA/ACT/DET
Description: Same as *Protected Data Envelope 1*.

A.1.124 Protected Data Envelope 3

Tag: '9F72'
Template: —
Length: var. up to 192
Format: b
Update: K/RA/ACT/DET
Description: Same as *Protected Data Envelope 1*.

A.1.125 Protected Data Envelope 4

Tag: '9F73'
Template: —
Length: var. up to 192
Format: b
Update: K/RA/ACT/DET
Description: Same as *Protected Data Envelope 1*.

A.1.126 Protected Data Envelope 5

Tag: '9F74'
Template: —
Length: var. up to 192
Format: b
Update: K/RA/ACT/DET
Description: Same as *Protected Data Envelope 1*.

A.1.127 PUNATC(Track1)

Tag: '9F63'
Template: '70'
Length: 6
Format: b
Update: K/RA
Description: *PUNATC(Track1)* indicates to the Kernel the positions in the discretionary data field of *Track 1 Data* where the *Unpredictable Number (Numeric)* digits and *Application Transaction Counter* digits have to be copied.

A.1.128 PUNATC(Track2)

Tag: '9F66'
Template: '70'
Length: 2
Format: b
Update: K/RA
Description: *PUNATC(Track2)* indicates to the Kernel the positions in the discretionary data field of *Track 2 Data* where the *Unpredictable Number (Numeric)* digits and *Application Transaction Counter* digits have to be copied.

A.1.129 Reader Contactless Floor Limit

Tag: 'DF8123'
Template: —
Length: 6
Format: n 12
Update: K
Description: Indicates the transaction amount above which transactions must be authorized online.

A.1.130 Reader Contactless Transaction Limit

Tag:	—
Template:	—
Length:	6
Format:	n 12
Update:	K
Description:	Indicates the transaction amount above which the transaction is not allowed. This data object is instantiated with <i>Reader Contactless Transaction Limit (On-device CVM)</i> if on device cardholder verification is supported by the Card and with <i>Reader Contactless Transaction Limit (No On-device CVM)</i> otherwise.

A.1.131 Reader Contactless Transaction Limit (No On-device CVM)

Tag:	'DF8124'
Template:	—
Length:	6
Format:	n 12
Update:	K
Description:	Indicates the transaction amount above which the transaction is not allowed, when on device cardholder verification is not supported.

A.1.132 Reader Contactless Transaction Limit (On-device CVM)

Tag:	'DF8125'
Template:	—
Length:	6
Format:	n 12
Update:	K
Description:	Indicates the transaction amount above which the transaction is not allowed, when on device cardholder verification is supported.

A.1.133 Reader CVM Required Limit

Tag:	'DF8126'
Template:	—
Length:	6
Format:	n 12
Update:	K
Description:	Indicates the transaction amount above which the Kernel instantiates the CVM capabilities field in <i>Terminal Capabilities</i> with <i>CVM Capability – CVM Required</i> .

A.1.134 Read Record Response Message Template

Tag:	'70'
Template:	—
Length:	var. up to 253
Format:	b
Update:	K/RA
Description:	Template containing the data objects returned by the Card in response to a READ RECORD command.

A.1.135 Reference Control Parameter

Tag: 'DF8114'
 Template: —
 Length: 1
 Format: b
 Update: K
 Description: Working variable to store the reference control parameter of the GENERATE AC command.

Reference Control Parameter			
Byte 1	b8-7	AC type	
			00: AAC
			01: TC
			10: ARQC
			11: RFU
	b6	RFU	
	b5	CDA signature requested	
	b4-1	Each bit RFU	

A.1.136 Relay Resistance Accuracy Threshold

Tag: 'DF8136'
 Template: —
 Length: 2
 Format: b
 Update: K
 Description: Represents the threshold above which the Kernel considers the variation between *Measured Relay Resistance Processing Time* and *Min Time For Processing Relay Resistance APDU* no longer acceptable. The *Relay Resistance Accuracy Threshold* is expressed in units of hundreds of microseconds.

A.1.137 Relay Resistance Transmission Time Mismatch Threshold

Tag:	'DF8137'
Template:	—
Length:	1
Format:	b
Update:	K
Description:	Represents the threshold above which the Kernel considers the variation between <i>Device Estimated Transmission Time For Relay Resistance R-APDU</i> and <i>Terminal Expected Transmission Time For Relay Resistance R-APDU</i> no longer acceptable. The <i>Relay Resistance Transmission Time Mismatch Threshold</i> is a percentage and expressed as an integer.

A.1.138 Response Message Template Format 1

Tag:	'80'
Template:	—
Length:	var. up to 253
Format:	b
Update:	K/RA
Description:	Contains the data objects (without tags and lengths) returned by the Card in response to a command.

A.1.139 Response Message Template Format 2

Tag:	'77'
Template:	—
Length:	var. up to 253
Format:	b
Update:	K/RA
Description:	Contains the data objects (with tags and lengths) returned by the Card in response to a command.

A.1.140 RRP Counter

Tag: 'DF8307'
 Template: —
 Length: 1
 Format: b
 Update: K
 Description: Represents the number of retry attempts to send the EXCHANGE RELAY RESISTANCE DATA command to the Card within one transaction.

A.1.141 Security Capability

Tag: 'DF811F'
 Template: —
 Length: 1
 Format: b
 Update: K
 Description: Indicates the security capability of the Kernel.

Security Capability		
Byte 1	b8	SDA
	b7	DDA
	b6	Card capture
	b5	RFU
	b4	CDA
	b3-1	Each bit RFU

A.1.142 Service Code

Tag: '5F30'
 Template: '70' or '77'
 Length: 2
 Format: n 3
 Update: K/RA
 Description: Service code as defined in *Track 1 Data* and *Track 2 Data*.

A.1.143 Signed Dynamic Application Data

Tag:	'9F4B'
Template:	'77'
Length:	N _{IC}
Format:	b
Update:	K/RA
Description:	Digital signature on critical application parameters for CDA.

A.1.144 Static Data Authentication Tag List

Tag:	'9F4A'
Template:	'70' or '77'
Length:	var. up to 250
Format:	b
Update:	K/RA
Description:	List of tags of primitive data objects defined in this specification for which the value fields must be included in the static data to be signed.

A.1.145 Static Data To Be Authenticated

Tag:	—
Template:	—
Length:	var. up to 2048
Format:	b
Update:	K
Description:	Buffer used to concatenate records that are involved in offline data authentication.

A.1.146 Tags To Read

Tag:	'DF8112'
Template:	—
Length:	var.
Format:	b
Update:	K/ACT/DET
Description:	<p>List of tags indicating the data the Terminal has requested to be read. This data item is present if the Terminal wants any data back from the Card before the <i>Data Record</i>. This could be in the context of SDS, or for non data storage usage reasons, for example the PAN. This data item may contain configured data.</p> <p>This data object may be provided several times by the Terminal. Therefore, the values of each of these tags must be accumulated in the <i>Tags To Read Yet</i> buffer.</p>

A.1.147 Tags To Read Yet

Tag:	—
Template:	—
Length:	var.
Format:	b
Update:	K
Description:	<p>List of tags that contains the accumulated Terminal data reading requests received in <i>Tags To Read</i>. Requested data objects that are sent to the Terminal are spooled from this buffer.</p> <p><i>Tags To Read Yet</i> is initiated when the Kernel is started with <i>Tags To Read</i> if present in the ACT Signal. This list can be augmented with Terminal requested data items provided during Kernel processing in DET Signals.</p> <p>The Kernel sends the requested data objects to the Terminal with the DEK Signal in <i>Data To Send</i>.</p>

A.1.148 Tags To Write After Gen AC

Tag:	'FF8103'
Template:	—
Length:	var.
Format:	b
Update:	K/ACT/DET
Description:	Contains the Terminal data writing requests to be sent to the Card after processing the GENERATE AC command. The value of this data object is composed of a series of TLVs. This data object may be provided several times by the Terminal in a DET Signal. Therefore, these values must be accumulated in <i>Tags To Write Yet After Gen AC</i> .

A.1.149 Tags To Write Before Gen AC

Tag:	'FF8102'
Template:	—
Length:	var.
Format:	b
Update:	K/ACT/DET
Description:	List of data objects indicating the Terminal data writing requests to be sent to the Card before processing the GENERATE AC command. This data object may be provided several times by the Terminal in a DET Signal. Therefore, these values must be accumulated in <i>Tags To Write Yet Before Gen AC</i> buffer.

A.1.150 Tags To Write Yet After Gen AC

Tag:	—
Template:	—
Length:	var.
Format:	b
Update:	K
Description:	List of data objects that contains the accumulated Terminal data writing requests received in <i>Tags To Write After Gen AC</i> .

A.1.151 Tags To Write Yet Before Gen AC

Tag:	—
Template:	—
Length:	var.
Format:	b
Update:	K
Description:	List of data objects that contains the accumulated Terminal data writing requests received in <i>Tags To Write Before Gen AC</i> .

A.1.152 Terminal Action Code – Default

Tag:	'DF8120'
Template:	—
Length:	5
Format:	b
Update:	K
Description:	Specifies the acquirer's conditions that cause a transaction to be rejected on an offline only Terminal.

A.1.153 Terminal Action Code – Denial

Tag:	'DF8121'
Template:	—
Length:	5
Format:	b
Update:	K
Description:	Specifies the acquirer's conditions that cause the denial of a transaction without attempting to go online.

A.1.154 Terminal Action Code – Online

Tag:	'DF8122'
Template:	—
Length:	5
Format:	b
Update:	K
Description:	Specifies the acquirer's conditions that cause a transaction to be transmitted online on an online capable Terminal.

A.1.155 Terminal Capabilities

Tag: '9F33'
 Template: —
 Length: 3
 Format: b
 Update: K
 Description: Indicates the card data input, CVM, and security capabilities of the Terminal and Reader. The CVM capability (Byte 2) is instantiated with values depending on the transaction amount.

Terminal Capabilities		
Byte 1	b8	Manual key entry
	b7	Magnetic stripe
	b6	IC with contacts
	b5-1	Each bit RFU
Byte 2	b8	Plaintext PIN for ICC verification
	b7	Enciphered PIN for online verification
	b6	Signature (paper)
	b5	Enciphered PIN for offline verification
	b4	No CVM required
	b3-1	Each bit RFU
Byte 3	b8	SDA
	b7	DDA
	b6	Card capture
	b5	RFU
	b4	CDA
	b3-1	Each bit RFU

A.1.156 Terminal Country Code

Tag:	'9F1A'
Template:	—
Length:	2
Format:	n 3
Update:	K
Description:	Indicates the country of the Terminal, represented in accordance with [ISO 3166-1].

A.1.157 Terminal Expected Transmission Time For Relay Resistance C-APDU

Tag:	'DF8134'
Template:	—
Length:	2
Format:	b
Update:	K
Description:	Represents the time that the Kernel expects to need for transmitting the EXCHANGE RELAY RESISTANCE DATA command to the Card. The <i>Terminal Expected Transmission Time For Relay Resistance C-APDU</i> is expressed in units of hundreds of microseconds.

A.1.158 Terminal Expected Transmission Time For Relay Resistance R-APDU

Tag:	'DF8135'
Template:	—
Length:	2
Format:	b
Update:	K
Description:	Represents the time that the Kernel expects that the Card will need for transmitting the EXCHANGE RELAY RESISTANCE DATA R-APDU. The <i>Terminal Expected Transmission Time For Relay Resistance R-APDU</i> is expressed in units of hundreds of microseconds.

A.1.159 Terminal Identification

Tag:	'9F1C'
Template:	—
Length:	8
Format:	an 8
Update:	K
Description:	Designates the unique location of the Terminal.

A.1.160 Terminal Relay Resistance Entropy

Tag:	'DF8301'
Template:	—
Length:	4
Format:	b
Update:	K
Description:	Contains a Kernel challenge (random) to be used in the value field of the EXCHANGE RELAY RESISTANCE DATA command.

A.1.161 Terminal Risk Management Data

Tag:	'9F1D'
Template:	—
Length:	8
Format:	b
Update:	K
Description:	Application-specific value used by the cardholder device for risk management purposes.

Terminal Risk Management Data		
Byte 1	8	Restart supported
	7	Enciphered PIN verified online (Contactless)
	6	Signature (paper) (Contactless)
	5	Enciphered PIN verification performed by ICC (Contactless)
	4	No CVM required (Contactless)
	3	CDCVM (Contactless)
	2	Plaintext PIN verification performed by ICC (Contactless)
	1	Present and Hold supported
Byte 2	8	CVM Limit exceeded
	7	Enciphered PIN verified online (Contact)
	6	Signature (paper) (Contact)
	5	Enciphered PIN verification performed by ICC (Contact)
	4	No CVM required (Contact)
	3	CDCVM (Contact)
	2	Plaintext PIN verification performed by ICC (Contact)
	1	RFU
Byte 3	8	Mag-stripe mode contactless transactions not supported
	7	EMV mode contactless transactions not supported
	6	CDCVM without CDA supported
	5-1	RFU
Byte 4	8	CDCVM bypass requested
	7	SCA exempt
	6-1	RFU
Byte 5	8-1	RFU
Byte 6	8-1	RFU
Byte 7	8-1	RFU
Byte 8	8-1	RFU

A.1.162 Terminal Type

Tag:	'9F35'
Template:	—
Length:	1
Format:	n 2
Update:	K
Description:	Indicates the environment of the Terminal, its communications capability, and its operational control. The <i>Terminal Type</i> is coded according to Annex A.1 of [EMV Book 4].

A.1.163 Terminal Verification Results

Tag:	'95'
Template:	—
Length:	5
Format:	b
Update:	K
Description:	Status of the different functions from the Terminal perspective. The <i>Terminal Verification Results</i> is coded according to Annex C.5 of [EMV Book 3]. Bits that have been reserved for use by contactless specifications are defined as shown.

Terminal Verification Results		
Byte 1	b8	Offline data authentication was not performed
	b7	SDA failed
	b6	ICC data missing
	b5	Card appears on terminal exception file
	b4	DDA failed
	b3	CDA failed
	b2-1	Each bit RFU

Terminal Verification Results		
Byte 2	b8	ICC and terminal have different application versions
	b7	Expired application
	b6	Application not yet effective
	b5	Requested service not allowed for card product
	b4	New card
	b3-1	Each bit RFU
Byte 3	b8	Cardholder verification was not successful
	b7	Unrecognised CVM
	b6	PIN Try Limit exceeded
	b5	PIN entry required and PIN pad not present or not working
	b4	PIN entry required, PIN pad present, but PIN was not entered
	b3	Online PIN entered
	b2-1	Each bit RFU
Byte 4	b8	Transaction exceeds floor limit
	b7	Lower consecutive offline limit exceeded
	b6	Upper consecutive offline limit exceeded
	b5	Transaction selected randomly for online processing
	b4	Merchant forced transaction online
	b3-1	Each bit RFU

Terminal Verification Results			
Byte 5	b8	Default TDOL used	
	b7	Issuer authentication failed	
	b6	Script processing failed before final GENERATE AC	
	b5	Script processing failed after final GENERATE AC	
	b4	Relay resistance threshold exceeded	
	b3	Relay resistance time limits exceeded	
	b2-1	Relay resistance performed	
			00: Relay resistance protocol not supported (not used by this version of the specification)
			01: RRP NOT PERFORMED
			10: RRP PERFORMED
			11: RFU

A.1.164 Token Requestor ID

Tag:	'9F19'
Template:	'70' or '77'
Length:	6
Format:	n 11
Update:	K/RA
Description:	The <i>Token Requestor ID</i> uniquely identifies the pairing of the Token Requestor with the Token Domain, as defined in [EMV Token].

A.1.165 Third Party Data

Tag: '9F6E'
 Template: 'BF0C' or '70'
 Length: var. 5 to 32
 Format: b
 Update: K/RA
 Description: The *Third Party Data* contains various information, possibly including information from a third party. If present in the Card, the *Third Party Data* must be returned in a file read using the READ RECORD command or in the *File Control Information Template*.
 'Device Type' is present when the most significant bit of byte 1 of 'Unique Identifier' is set to 0b. In this case, the maximum length of 'Proprietary Data' is 26 bytes. Otherwise it is 28 bytes.

Data Field	Length	Format
Country Code	2	Country Code according to [ISO 3166-1]
Unique Identifier	2	b (value assigned by MasterCard)
Device Type	0 or 2	an
Proprietary Data	1-26 or 28	b

A.1.166 Time Out Value

Tag: 'DF8127'
 Template: —
 Length: 2
 Format: b
 Update: K
 Description: Defines the time in ms before the timer generates a TIMEOUT Signal.

A.1.167 Track 1 Data

Tag: '56'
 Template: '70'
 Length: var. up to 76
 Format: ans
 Update: K/RA
 Description: *Track 1 Data* contains the data objects of the track 1 according to [ISO/IEC 7813] Structure B, excluding start sentinel, end sentinel and LRC. The *Track 1 Data* may be present in the file read using the READ RECORD command during a Mag-stripe Mode transaction. It is made up of the following sub-fields:

Data Field	Length	Format
Format Code	1	'42'
Primary Account Number	var up to 19	digits
Field Separator	1	'5E'
Name	2-26	(see ISO/IEC 7813)
Field Separator	1	'5E'
Expiry Date	4	YYMM
Service Code	3	digits
Discretionary Data	var.	ans

A.1.168 Track 1 Discretionary Data

Tag: '9F1F'
 Template: '70' or '77'
 Length: var. up to 54
 Format: ans
 Update: K/RA
 Description: Discretionary part of track 1 according to [ISO/IEC 7813].

A.1.169 Track 2 Data

Tag: '9F6B'
 Template: '70'
 Length: var. up to 19
 Format: b
 Update: K/RA
 Description: *Track 2 Data* contains the data objects of the track 2 according to [ISO/IEC 7813], excluding start sentinel, end sentinel and LRC. The *Track 2 Data* has a maximum length of 37 positions and is present in the file read using the READ RECORD command during a Mag-stripe Mode transaction. It is made up of the following sub-fields:

Data Field	Length	Format
Primary Account Number	var. up to 19 nibbles	n
Field Separator	1 nibble	b ('D')
Expiry Date	2	n (YYMM)
Service Code	3 nibbles	n
Discretionary Data	var.	n
Padded with 'F' if needed to ensure whole bytes.	1 nibble	b

A.1.170 Track 2 Discretionary Data

Tag: '9F20'
 Template: '70' or '77'
 Length: var. up to 16
 Format: cn
 Update: K/RA
 Description: Discretionary part of track 2 according to [ISO/IEC 7813].

A.1.171 Track 2 Equivalent Data

Tag: '57'
 Template: '70' or '77'
 Length: var. up to 19
 Format: b
 Update: K/RA
 Description: Contains the data objects of the track 2, in accordance with [ISO/IEC 7813], excluding start sentinel, end sentinel, and LRC. The *Track 2 Equivalent Data* has a maximum length of 37 positions and is present in the file read using the READ RECORD command during an EMV mode transaction. It is made up of the following sub-fields:

Data Field	Length	Format
Primary Account Number	var. up to 19 nibbles	n
Field Separator	1 nibble	b ('D')
Expiration Date (YYMM)	2	n (YYMM)
Service Code	3 nibbles	n
Discretionary Data	var.	n
Padded with 'F' if needed to ensure whole bytes	1 nibble	b

A.1.172 Transaction Category Code

Tag: '9F53'
 Template: —
 Length: 1
 Format: an
 Update: K/ACT/DET
 Description: This is a data object defined by MasterCard which indicates the type of transaction being performed, and which may be used in card risk management.

A.1.173 Transaction Currency Code

Tag:	'5F2A'
Template:	—
Length:	2
Format:	n 3
Update:	K/ACT/DET
Description:	Indicates the currency code of the transaction, in accordance with [ISO 4217].

A.1.174 Transaction Currency Exponent

Tag:	'5F36'
Template:	—
Length:	1
Format:	n 1
Update:	K/ACT/DET
Description:	Indicates the implied position of the decimal point from the right of the transaction amount represented, in accordance with [ISO 4217].

A.1.175 Transaction Date

Tag:	'9A'
Template:	—
Length:	3
Format:	n 6 (YYMMDD)
Update:	K/ACT/DET
Description:	Local date that the transaction was performed.

A.1.176 Transaction Time

Tag:	'9F21'
Template:	—
Length:	3
Format:	n 6 (HHMMSS)
Update:	K/ACT/DET
Description:	Local time at which the transaction was performed.

A.1.177 Transaction Type

Tag:	'9C'
Template:	—
Length:	1
Format:	n 2
Update:	K/ACT/DET
Description:	Indicates the type of financial transaction, represented by the first two digits of [ISO 8583:1987] Processing Code.

A.1.178 UDOL

Tag:	'9F69'
Template:	'70'
Length:	var. up to 250
Format:	b
Update:	K/RA
Description:	The <i>UDOL</i> is the DOL that specifies the data objects to be included in the data field of the COMPUTE CRYPTOGRAPHIC CHECKSUM command. The <i>UDOL</i> must at least include the <i>Unpredictable Number (Numeric)</i> . The <i>UDOL</i> is not mandatory for the Card. If it is not present in the Card, then the <i>Default UDOL</i> is used.

A.1.179 Unpredictable Number

Tag:	'9F37'
Template:	—
Length:	4
Format:	b
Update:	K
Description:	Contains a Kernel challenge (random) to be used by the Card to ensure the variability and uniqueness to the generation of a cryptogram during an EMV mode transaction.

A.1.180 Unpredictable Number (Numeric)

Tag:	'9F6A'
Template:	—
Length:	4
Format:	n 8
Update:	K
Description:	Unpredictable number generated by the Kernel during a Mag-stripe Mode transaction. The <i>Unpredictable Number (Numeric)</i> is passed to the Card in the data field of the COMPUTE CRYPTOGRAPHIC CHECKSUM command. The 8- <i>nUN</i> most significant digits must be set to zero.

A.1.181 Unprotected Data Envelope 1

Tag:	'9F75'
Template:	—
Length:	var. up to 192
Format:	b
Update:	K/RA/ACT/DET
Description:	The Unprotected Data Envelopes contain proprietary information from the issuer, payment system or third party. Unprotected Data Envelopes can be retrieved with the GET DATA command and can be updated with the PUT DATA (CLA='80') command without secure messaging.

A.1.182 Unprotected Data Envelope 2

Tag:	'9F76'
Template:	—
Length:	var. up to 192
Format:	b
Update:	K/RA/ACT/DET
Description:	Same as <i>Unprotected Data Envelope 1</i> .

A.1.183 Unprotected Data Envelope 3

Tag: '9F77'
Template: —
Length: var. up to 192
Format: b
Update: K/RA/ACT/DET
Description: Same as *Unprotected Data Envelope 1*.

A.1.184 Unprotected Data Envelope 4

Tag: '9F78'
Template: —
Length: var. up to 192
Format: b
Update: K/RA/ACT/DET
Description: Same as *Unprotected Data Envelope 1*.

A.1.185 Unprotected Data Envelope 5

Tag: '9F79'
Template: —
Length: var. up to 192
Format: b
Update: K/RA/ACT/DET
Description: Same as *Unprotected Data Envelope 1*.

A.1.186 User Interface Request Data

Tag: 'DF8116'
 Template: —
 Length: 13
 Format: b
 Update: K
 Description: Combines all parameters to be sent with the OUT Signal or MSG Signal.

Data Field	Length	Format
Message Identifier	1	b (see below)
Status	1	b (see below)
Hold Time	3	n 6
Language Preference	8	an (padded with hexadecimal zeroes if length of tag '5F2D' is less than 8 bytes)

Message Identifier		
Byte 1	b8-1	Message Identifier
		00010111: CARD READ OK
		00100001: TRY AGAIN
		00000011: APPROVED
		00011010: APPROVED – SIGN
		00000111: DECLINED
		00011100: ERROR – OTHER CARD
		00011101: INSERT CARD
		00100000: SEE PHONE
		00011011: AUTHORISING – PLEASE WAIT
		00011110: CLEAR DISPLAY
		11111111: N/A
		Other values: RFU

Status			
Byte 1	b8-1	Status	
			00000000: NOT READY
			00000001: IDLE
			00000010: READY TO READ
			00000011: PROCESSING
			00000100: CARD READ SUCCESSFULLY
			00000101: PROCESSING ERROR
			11111111: N/A
			Other values: RFU

A.2 Data Objects by Tag

Table A.1 lists all data objects of Kernel 2 ordered by tag number and indicates for the different Implementation Options if the data object is included in the TLV Database of that Implementation Option.

Table A.1—Data Objects by Tag

Tag	Data Object	NOT MAG & NOT DE/DS	MAG & NOT DE/DS	NOT MAG & DE/DS	MAG & DE/DS
'50'	Application Label	✓	✓	✓	✓
'56'	Track 1 Data		✓		✓
'57'	Track 2 Equivalent Data	✓	✓	✓	✓
'5A'	Application PAN	✓	✓	✓	✓
'5F24'	Application Expiration Date	✓	✓	✓	✓
'5F25'	Application Effective Date	✓	✓	✓	✓
'5F28'	Issuer Country Code	✓	✓	✓	✓
'5F2A'	Transaction Currency Code	✓	✓	✓	✓
'5F2D'	Language Preference	✓	✓	✓	✓
'5F30'	Service Code	✓	✓	✓	✓
'5F34'	Application PAN Sequence Number	✓	✓	✓	✓
'5F36'	Transaction Currency Exponent	✓	✓	✓	✓

Tag	Data Object	NOT MAG & NOT DE/DS	MAG & NOT DE/DS	NOT MAG & DE/DS	MAG & DE/DS
'5F57'	Account Type	✓	✓	✓	✓
'6F'	File Control Information Template	✓	✓	✓	✓
'70'	Read Record Template	✓	✓	✓	✓
'77'	Response Message Template Format 2	✓	✓	✓	✓
'80'	Response Message Template Format 1	✓	✓	✓	✓
'82'	Application Interchange Profile	✓	✓	✓	✓
'84'	DF Name	✓	✓	✓	✓
'87'	Application Priority Indicator	✓	✓	✓	✓
'8C'	CDOL1	✓	✓	✓	✓
'8E'	CVM List	✓	✓	✓	✓
'8F'	CA Public Key Index (Card)	✓	✓	✓	✓
'90'	Issuer Public Key Certificate	✓	✓	✓	✓
'92'	Issuer Public Key Remainder	✓	✓	✓	✓
'94'	Application File Locator	✓	✓	✓	✓
'95'	Terminal Verification Results	✓	✓	✓	✓
'9A'	Transaction Date	✓	✓	✓	✓
'9C'	Transaction Type	✓	✓	✓	✓
'9F01'	Acquirer Identifier	✓	✓	✓	✓

Tag	Data Object	NOT MAG & NOT DE/DS	MAG & NOT DE/DS	NOT MAG & DE/DS	MAG & DE/DS
'9F02'	Amount, Authorized (Numeric)	✓	✓	✓	✓
'9F03'	Amount, Other (Numeric)	✓	✓	✓	✓
'9F07'	Application Usage Control	✓	✓	✓	✓
'9F08'	Application Version Number (Card)	✓	✓	✓	✓
'9F09'	Application Version Number (Reader)	✓	✓	✓	✓
'9F0D'	Issuer Action Code – Default	✓	✓	✓	✓
'9F0E'	Issuer Action Code – Denial	✓	✓	✓	✓
'9F0F'	Issuer Action Code – Online	✓	✓	✓	✓
'9F10'	Issuer Application Data	✓	✓	✓	✓
'9F11'	Issuer Code Table Index	✓	✓	✓	✓
'9F12'	Application Preferred Name	✓	✓	✓	✓
'9F15'	Merchant Category Code	✓	✓	✓	✓
'9F16'	Merchant Identifier	✓	✓	✓	✓
'9F19'	Token Requestor ID	✓	✓	✓	✓
'9F1A'	Terminal Country Code	✓	✓	✓	✓
'9F1C'	Terminal Identification	✓	✓	✓	✓
'9F1D'	Terminal Risk Management Data	✓	✓	✓	✓
'9F1E'	Interface Device Serial Number	✓	✓	✓	✓

Tag	Data Object	NOT MAG & NOT DE/DS	MAG & NOT DE/DS	NOT MAG & DE/DS	MAG & DE/DS
'9F1F'	Track 1 Discretionary Data	✓	✓	✓	✓
'9F20'	Track 2 Discretionary Data	✓	✓	✓	✓
'9F21'	Transaction Time	✓	✓	✓	✓
'9F24'	Payment Account Reference	✓	✓	✓	✓
'9F26'	Application Cryptogram	✓	✓	✓	✓
'9F27'	Cryptogram Information Data	✓	✓	✓	✓
'9F32'	Issuer Public Key Exponent	✓	✓	✓	✓
'9F33'	Terminal Capabilities	✓	✓	✓	✓
'9F34'	CVM Results	✓	✓	✓	✓
'9F35'	Terminal Type	✓	✓	✓	✓
'9F36'	Application Transaction Counter	✓	✓	✓	✓
'9F37'	Unpredictable Number	✓	✓	✓	✓
'9F38'	PDOL	✓	✓	✓	✓
'9F40'	Additional Terminal Capabilities	✓	✓	✓	✓
'9F42'	Application Currency Code	✓	✓	✓	✓
'9F44'	Application Currency Exponent	✓	✓	✓	✓
'9F46'	ICC Public Key Certificate	✓	✓	✓	✓
'9F47'	ICC Public Key Exponent	✓	✓	✓	✓

Tag	Data Object	NOT MAG & NOT DE/DS	MAG & NOT DE/DS	NOT MAG & DE/DS	MAG & DE/DS
'9F48'	ICC Public Key Remainder	✓	✓	✓	✓
'9F4A'	Static Data Authentication Tag List	✓	✓	✓	✓
'9F4B'	Signed Dynamic Application Data	✓	✓	✓	✓
'9F4C'	ICC Dynamic Number	✓	✓	✓	✓
'9F4D'	Log Entry	✓	✓	✓	✓
'9F4E'	Merchant Name and Location	✓	✓	✓	✓
'9F53'	Transaction Category Code	✓	✓	✓	✓
'9F54'	DS ODS Card			✓	✓
'9F5B'	DSDOL			✓	✓
'9F5C'	DS Requested Operator ID			✓	✓
'9F5D'	Application Capabilities Information	✓	✓	✓	✓
'9F5E'	DS ID			✓	✓
'9F5F'	DS Slot Availability			✓	✓
'9F60'	CVC3 (Track1)		✓		✓
'9F61'	CVC3 (Track2)		✓		✓
'9F62'	PCVC3(Track1)		✓		✓
'9F63'	PUNATC(Track1)		✓		✓
'9F64'	NATC(Track1)		✓		✓

Tag	Data Object	NOT MAG & NOT DE/DS	MAG & NOT DE/DS	NOT MAG & DE/DS	MAG & DE/DS
'9F65'	PCVC3(Track2)		✓		✓
'9F66'	PUNATC(Track2)		✓		✓
'9F67'	NATC(Track2)		✓		✓
'9F69'	UDOL		✓		✓
'9F6A'	Unpredictable Number (Numeric)		✓		✓
'9F6B'	Track 2 Data		✓		✓
'9F6D'	Mag-stripe Application Version Number (Reader)		✓		✓
'9F6E'	Third Party Data	✓	✓	✓	✓
'9F6F'	DS Slot Management Control			✓	✓
'9F70'	Protected Data Envelope 1			✓	✓
'9F71'	Protected Data Envelope 2			✓	✓
'9F72'	Protected Data Envelope 3			✓	✓
'9F73'	Protected Data Envelope 4			✓	✓
'9F74'	Protected Data Envelope 5			✓	✓
'9F75'	Unprotected Data Envelope 1			✓	✓
'9F76'	Unprotected Data Envelope 2			✓	✓
'9F77'	Unprotected Data Envelope 3			✓	✓

Tag	Data Object	NOT MAG & NOT DE/DS	MAG & NOT DE/DS	NOT MAG & DE/DS	MAG & DE/DS
'9F78'	Unprotected Data Envelope 4			✓	✓
'9F79'	Unprotected Data Envelope 5			✓	✓
'9F7C'	Merchant Custom Data	✓	✓	✓	✓
'9F7D'	DS Summary 1			✓	✓
'9F7E'	Mobile Support Indicator	✓	✓	✓	✓
'9F7F'	DS Unpredictable Number			✓	✓
'A5'	File Control Information Proprietary Template	✓	✓	✓	✓
'BF0C'	File Control Information Issuer Discretionary Data	✓	✓	✓	✓
'DF4B'	POS Cardholder Interaction Information	✓	✓	✓	✓
'DF60'	DS Input (Card)			✓	✓
'DF61'	DS Digest H			✓	✓
'DF62'	DS ODS Info			✓	✓
'DF63'	DS ODS Term			✓	✓
'DF8101'	DS Summary 2			✓	✓
'DF8102'	DS Summary 3			✓	✓
'DF8106'	Data Needed			✓	✓

Tag	Data Object	NOT MAG & NOT DE/DS	MAG & NOT DE/DS	NOT MAG & DE/DS	MAG & DE/DS
'DF8107'	CDOL1 Related Data	✓	✓	✓	✓
'DF8108'	DS AC Type			✓	✓
'DF8109'	DS Input (Term)			✓	✓
'DF810A'	DS ODS Info For Reader			✓	✓
'DF810B'	DS Summary Status			✓	✓
'DF810C'	Kernel ID	✓	✓	✓	✓
'DF810D'	DSVN Term			✓	✓
'DF810E'	Post-Gen AC Put Data Status			✓	✓
'DF810F'	Pre-Gen AC Put Data Status			✓	✓
'DF8110'	Proceed To First Write Flag			✓	✓
'DF8111'	PDOL Related Data	✓	✓	✓	✓
'DF8112'	Tags To Read			✓	✓
'DF8114'	Reference Control Parameter	✓	✓	✓	✓
'DF8115'	Error Indication	✓	✓	✓	✓
'DF8116'	User Interface Request Data	✓	✓	✓	✓
'DF8117'	Card Data Input Capability	✓	✓	✓	✓
'DF8118'	CVM Capability – CVM Required	✓	✓	✓	✓
'DF8119'	CVM Capability – No CVM Required	✓	✓	✓	✓

Tag	Data Object	NOT MAG & NOT DE/DS	MAG & NOT DE/DS	NOT MAG & DE/DS	MAG & DE/DS
'DF811A'	Default UDOL		✓		✓
'DF811B'	Kernel Configuration	✓	✓	✓	✓
'DF811E'	Mag-stripe CVM Capability – CVM Required		✓		✓
'DF811F'	Security Capability	✓	✓	✓	✓
'DF8120'	Terminal Action Code – Default	✓	✓	✓	✓
'DF8121'	Terminal Action Code – Denial	✓	✓	✓	✓
'DF8122'	Terminal Action Code – Online	✓	✓	✓	✓
'DF8123'	Reader Contactless Floor Limit	✓	✓	✓	✓
'DF8124'	Reader Contactless Transaction Limit (No On-device CVM)	✓	✓	✓	✓
'DF8125'	Reader Contactless Transaction Limit (On-device CVM)	✓	✓	✓	✓
'DF8126'	Reader CVM Required Limit	✓	✓	✓	✓
'DF8127'	Time Out Value			✓	✓
'DF8128'	IDS Status			✓	✓
'DF8129'	Outcome Parameter Set	✓	✓	✓	✓
'DF812A'	DD Card (Track1)		✓		✓
'DF812B'	DD Card (Track2)		✓		✓

Tag	Data Object	NOT MAG & NOT DE/DS	MAG & NOT DE/DS	NOT MAG & DE/DS	MAG & DE/DS
'DF812C'	Mag-stripe CVM Capability – No CVM Required		✓		✓
'DF812D'	Message Hold Time	✓	✓	✓	✓
'DF8130'	Hold Time Value	✓	✓	✓	✓
'DF8131'	Phone Message Table	✓	✓	✓	✓
'DF8132'	Minimum Relay Resistance Grace Period	✓	✓	✓	✓
'DF8133'	Maximum Relay Resistance Grace Period	✓	✓	✓	✓
'DF8134'	Terminal Expected Transmission Time For Relay Resistance C-APDU	✓	✓	✓	✓
'DF8135'	Terminal Expected Transmission Time For Relay Resistance R-APDU	✓	✓	✓	✓
'DF8136'	Relay Resistance Accuracy Threshold	✓	✓	✓	✓
'DF8137'	Relay Resistance Transmission Time Mismatch Threshold	✓	✓	✓	✓
'DF8301'	Terminal Relay Resistance Entropy	✓	✓	✓	✓
'DF8302'	Device Relay Resistance Entropy	✓	✓	✓	✓
'DF8303'	Min Time For Processing Relay Resistance APDU	✓	✓	✓	✓

Tag	Data Object	NOT MAG & NOT DE/DS	MAG & NOT DE/DS	NOT MAG & DE/DS	MAG & DE/DS
'DF8304'	Max Time For Processing Relay Resistance APDU	✓	✓	✓	✓
'DF8305'	Device Estimated Transmission Time For Relay Resistance R-APDU	✓	✓	✓	✓
'DF8306'	Measured Relay Resistance Processing Time	✓	✓	✓	✓
'DF8307'	RRP Counter	✓	✓	✓	✓
'FF8102'	Tags To Write Before Gen AC			✓	✓
'FF8103'	Tags To Write After Gen AC			✓	✓
'FF8104'	Data To Send			✓	✓
'FF8105'	Data Record	✓	✓	✓	✓
'FF8106'	Discretionary Data	✓	✓	✓	✓

Annex B Kernel State Machine

B.1 Application States

The Kernel defined in this document is specified as an abstract state machine using states, transitions and Signals to model its behavior.

The application states are listed in Table B.1.

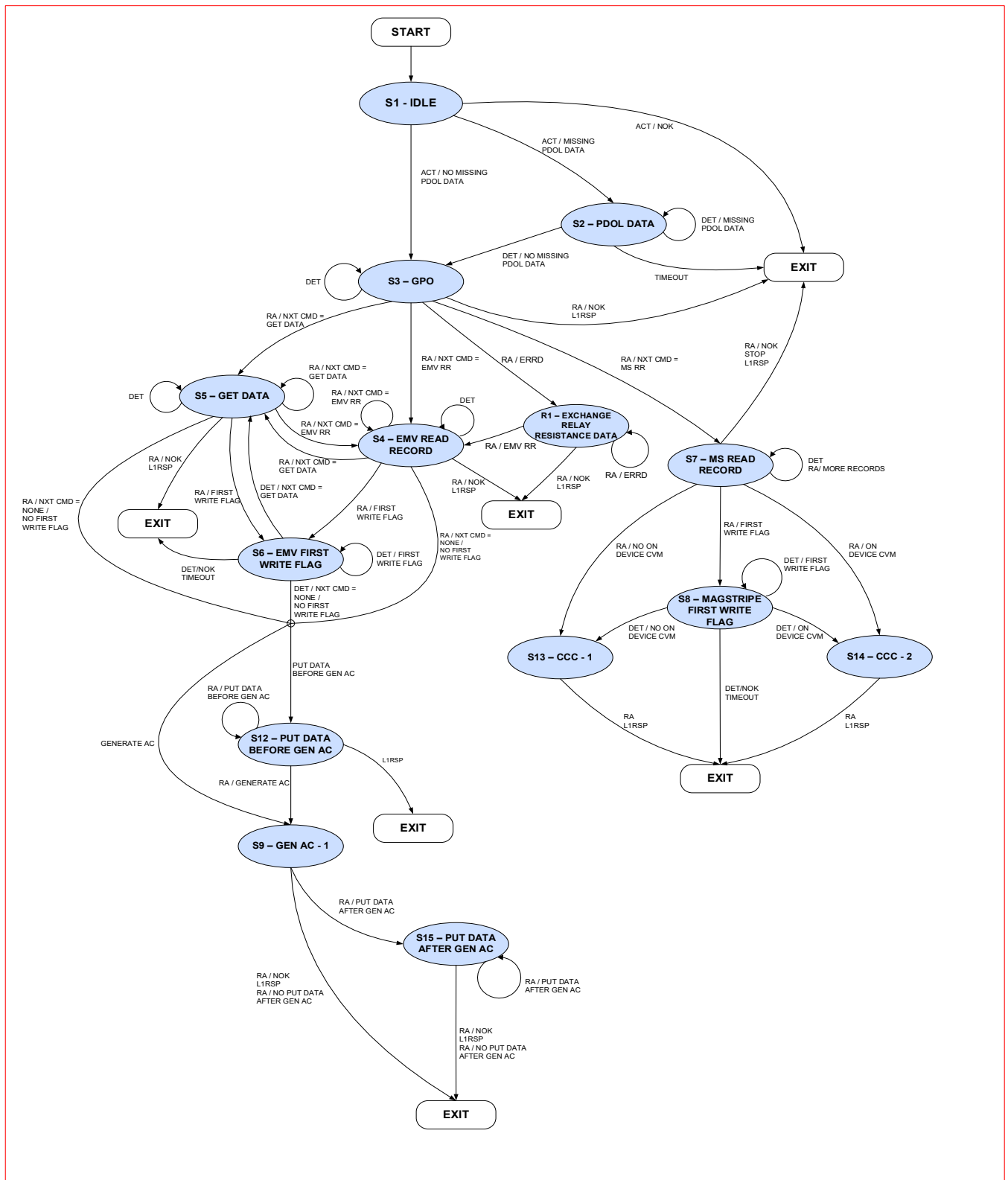
Table B.1—Kernel Application States

State	Description	Implementation Options
1 – Idle	Process has been created	Always
2 – Waiting for PDOL Data	Waiting for data to be provided by Terminal	DE/DS
3 – Waiting for GPO Response	Waiting for Card response to GET PROCESSING OPTIONS command	Always
R1 – Waiting for Exchange Relay Resistance Data Response	Waiting for Card response to EXCHANGE RELAY RESISTANCE DATA command	Always
4 – Waiting for EMV Read Record Response	Waiting for Card response to READ RECORD command during EMV mode transaction	Always
5 – Waiting for Get Data Response	Waiting for Card response to GET DATA command	DE/DS
6 – Waiting for EMV Mode First Write Flag	Waiting for Terminal to indicate that transaction can be completed	DE/DS
7 – Waiting for Mag-stripe Read Record Response	Waiting for Card response to READ RECORD command during Mag-stripe Mode transaction	MAG
8 – Waiting for Mag-stripe Mode First Write Flag	Waiting for Terminal to indicate that transaction can be completed during Mag-stripe Mode transaction	MAG & DE/DS
9 – Waiting for Generate AC Response	Waiting for Card response to GENERATE AC command	Always

State	Description	Implementation Options
12 – Waiting for Put Data Response Before Generate AC	Waiting for Card response to PUT DATA command sent before GENERATE AC	DE/DS
13 – Waiting for CCC Response – 1	Waiting for Card response to COMPUTE CRYPTOGRAPHIC CHECKSUM command (On device cardholder verification not supported)	MAG
14 – Waiting for CCC Response – 2	Waiting for Card response to COMPUTE CRYPTOGRAPHIC CHECKSUM command (On device cardholder verification supported)	MAG
15 – Waiting for Put Data Response After Generate AC	Waiting for Card response to PUT DATA command sent after GENERATE AC	DE/DS

B.2 State Machine

Figure B.1—Kernel State Machine



Annex C Glossary

The following terms are used in this document.

Table C.1—Terms

Term	Description
Card	Card, as used in these specifications, is a consumer device supporting contactless transactions.
Combination	Combination is the combination of an AID and a Kernel ID.
Configuration Option	A Configuration Option allows activation or deactivation of the Kernel software behind this option. The Configuration Option may change the execution path of the software but it does not change the software itself. A Configuration Option is set in the Kernel database per AID and transaction type.
EMV Mode	Operating mode that indicates that the contactless payment transaction is validated based on EMV minimum data.
EMVCo	The organization that manages the EMV Specifications and their related testing processes.
EMV®	A trademark owned by EMVCo, referring to the technical specifications published by EMVCo.
Implementation Option	An Implementation Option allows the vendor to select whether the functionality behind the option will be implemented in a particular installation.
Kernel	The Kernel contains the interface routines, security and control functions, and logic to manage a set of commands and responses to retrieve all the necessary data from the Card to complete a transaction. The Kernel processing covers the interaction with the Card between the selection of the card application (excluded) and the processing of the transaction's outcome (excluded).

Term	Description
Mag-stripe Mode	Mag-stripe Mode describes an operating mode of the POS System that indicates that this particular acceptance environment and acceptance rules support magnetic stripe infrastructure. It is typically used in conjunction with the term “transaction” (i.e. Mag-stripe Mode transaction) to indicate contactless payment based on Track 1 and/or Track 2 Data obtained from the Card.
POS System	The POS System is the collective term given to the payment infrastructure present at the merchant. It is made up of the Terminal and Reader.
Process	A Process is a logical component within a Reader that has one or more Queues to receive Signals. The processing of Signals, in combination with the carried data, may then generate other Signals to be sent. Processing can continue until all the Queues of a Process are empty, or until the Process terminates.
Queue	A Queue is a buffer that stores events to be processed. The events are stored in the order received.
Reader	The Reader is the device that supports the Kernel(s) and provides the contactless interface used by the Card. In this specification, the Reader is considered as a separate logical entity, although it can be an integral part of the POS System.
Signal	A Signal is an asynchronous event that is placed in a Queue. A Signal can convey data as parameters, and the data provided in this way is used in the processing of the Signal.
Terminal	The Terminal is the device that connects to the authorization and/or clearing network and that together with the Reader makes up the POS System. The Terminal and the Reader may exist in a single integrated device. However, in this specification, they are considered separate logical entities.

The following abbreviations are used in this document.

Table C.2—Abbreviations

Abbreviation	Description
AAC	Application Authentication Cryptogram
AC	Application Cryptogram
ADF	Application Definition File
AES	Advanced Encryption Standard
AFL	Application File Locator
AID	Application Identifier
AIP	Application Interchange Profile
an	Alphanumeric characters
ans	Alphanumeric and Special characters
APDU	Application Protocol Data Unit
ARQC	Authorization Request Cryptogram
ATC	Application Transaction Counter
b	Binary
BCD	Binary Coded Decimal
BER	Basic Encoding Rules
C	Conditional
CA	Certification Authority
C-APDU	Command APDU
CDA	Combined DDA/AC Generation
CDOL	Card Risk Management Data Object List
CID	Cryptogram Information Data
CLA	Class byte of command message
cn	Compressed Numeric
CRL	Certification Revocation List
CVC	Card Verification Code
CVM	Cardholder Verification Method
DE	Data Exchange
DEK	Data Exchange Kernel
DES	Data Encryption Standard
DET	Data Exchange Terminal

Abbreviation	Description
DF	Dedicated File
DOL	Data Object List
DSDOL	Data Storage Data Object List
ERRD	Exchange Relay Resistance Data
FCI	File Control Information
FIFO	First In First Out
IAD	Issuer Application Data
ICC	Integrated Circuit Card
IDS	Integrated Data Storage
INS	Instruction byte of command message
ISO	International Organization for Standardization
M	Mandatory
n	Numeric
N/A	Not applicable
N _{CA}	Length of CA Public Key Modulus
N _I	Length of Issuer Public Key Modulus
N _{IC}	Length of ICC Public Key Modulus
O	Optional
ODA	Offline Data Authentication
OD-CVM	On-Device Cardholder Verification Method
ODS	Operator Data Set
OWF	One Way Function
PAN	Primary Account Number
PCII	POS Cardholder Interaction Information
PDOL	Processing Options Data Object List
POS	Point of Sale
PPSE	Proximity Payment System Environment
PIN	Personal Identification Number
R/CNS	Rejected, Conditions Not Satisfied
RFU	Reserved for Future Use
R-APDU	Response APDU
RID	Registered Application Provider Identifier
RRP	Relay Resistance Protocol

Abbreviation	Description
SDAD	Signed Dynamic Application Data
SDS	Standalone Data Storage
SFI	Short File Identifier
SHA	Secure Hash Algorithm
SW12	Status bytes 1-2
TC	Transaction Certificate
TL	Tag Length
TLV	Tag Length Value
TVR	Terminal Verification Results
UDOL	Unpredictable Number Data Object List
UN	Unpredictable Number
var.	Variable

*** END OF DOCUMENT ***