Reference	GED24035_015_TDC_Est_eID_Developer_Guide	Release	1.21
Classification		Pages	101

THALES

Estonian eID 2025 - Developer Guide

	Name	Date	Signature
Issued by	**Markku Sievänen**	06/03/2025	
Verified by	Peter Widjeskog		
	Matias Hytönen		
	Manu Nurminen		
Approved by			

Distribution list			
Name	Role or Function	Name	Role or Function

Document Information

Release	Date	Author	Modifications
1.0	06/09/2024	Markku Sievänen	Initial version.

1.1	20/11/2024	Markku Sievänen	• Paragraph 8.3.2 Change PIN
			  ○ INS corrected from 0x2C to 0x24
			• Paragraph 8.3.3 Unblocking blocked PIN
			  ○ INS corrected from 0x0C to 0x2C

1.2	17/01/2025	Markku Sievänen	• Added paragraphs 2.1.1 What is PACE Authentication?, 2.1.2 How Does PACE Authentication Work?, 2.1.3 PACE Authentication Levels and 2.1.4 Impacts of PACE. • Added content of ATR and ATS bytes. • Added description of card file structure. • Added guidelines to read binary files, see paragraph 8.5 Certificate objects. • Added examples (APDU trace for main functionalities), see paragraph Annex A (Informative): example APDU trace for main functionalities
1.21	06/03/2025	Markku Sievänen	• Added to paragraph 1 Introduction ○ This work is licensed under CC BY 4.0. To view a copy of this license, visit https://creativecommons.org/licenses/by/4.0/

Table of content

1 Introduction

The aim of this document is to provide technical information to software developers/system integrators to be able to integrate the Estonian eID card as part of their applications.

1.1 Document scope

This document describes

- Card platform
- eID application
- APDU command formats
- Command APDUs
- Implementation guidelines for software developer

1.2 References

1. ISO, Information Technology - Identification cards - Integrated circuit(s) cards with contacts
   Part 1: Physical Characteristics, ISO/IEC 7816-1
   Part 2: Dimensions and location of the contacts, ISO/IEC 7816-2
   Part 3: Electronic signals and transmission protocols, ISO/IEC 7816-3
   Part 4: Interindustry commands for interchange, ISO/IEC 7816-4
   Part 5: Numbering system and registration procedure for application identifiers, ISO/IEC 7816-5
   Part 6: Inter-industry data elements, ISO/IEC 7816-6
   Part 8: Security related interindustry commands, ISO/IEC 7816-8
   Part 9: Commands for card management, ISO/IEC 7816-9
   Part 15: Cryptographic information application, ISO/IEC7816-15
2. ISO/IEC 14443–3:2018: Identification cards, Contactless integrated circuit(s) cards, Proximity cards, Part 3: Initialization and anticollision, Fourth Edition 2018
3. ISO/IEC 14443–4:2018: Identification cards, Contactless integrated circuit(s) cards, Proximity cards, Part 4: Transmission Protocol, Fourth Edition 2018
4. Global Platform, Card Specification, version 2.3.1
5. Doc 9303, Machine Readable Travel Documents, Part 11: Security Mechanisms for MRTDs, Eighth Edition 2021

1.3 Notations

xxxxxxxxb	Binary number, where x is 0 or 1.
xxh	Hexadecimal number, where x is 0-9 or A-F.
'xxxx'	Hexadecimal number, where x is 0-9 or A-F.
A | B	Concatenation of A and B.

1.4 Abbreviations

AC	Access Condition
AID	Application IDentifier
APDU	Application Protocol Data Unit
ASN.1	Abstract Syntax Notation One

CIA	Cryptographic Information Application
CLA	Class byte
CT	Confidentiality Template
CRDO	Control Reference Data Object
CRT	Chinese Remainder Theorem
DF	Dedicated File
DO	Data Object
DST	Digital Signature Template
EF	Elementary File
ELC	ELliptic Curve
FCI	File Control Information
FCP	File Control Parameter
Licc	Length of data available on the card
MF	Master File
MSE	Manage Security Environment
PACE	Password Authenticated Connection Establishment
PIN	Personal Identification Number
PKCS	Public-Key Cryptography Standards
PSO	Perform Security Operation
PUK	PIN Unblocking Key
RFU	Reserved for Future Use
RSA	Rivest, Shamir, Adleman
SC	Security Condition
SE	Security Environment
SFID	Short File IDentifier
S/MIME	Secure Multipurpose Internet Mail Extensions
SW1-SW2	Status bytes

2 Card platform

2.1 Dual interface

Estonian eID card platform supports both contact and contactless interfaces. Contactless interface is protected by Password Authenticated Connection Establishment (PACE) and secure messaging. PACE is a security feature that require the terminal and the card to share a simple secret to authenticate each other. PACE is based on the Diffie-Hellman protocol for strong mutual authentication.

2.1.1 What is PACE Authentication?

PACE is a password authenticated Diffie-Hellman key agreement protocol that provides secure communication (secure messaging) and password-based authentication of the smart card and the inspection system (that is, the smart card and inspection system share the same password).

PACE is independent of the eID application. It interacts between the software application that wants to communicate with the card and the eID application.

PACE can specify two levels of security:

- PACE authentication only.
- PACE authentication + secure messaging (always ENC + MAC).

In eID application context PACE authentication + secure messaging (always ENC + MAC) is used. These two levels are described in "PACE Authentication Levels".

2.1.2 How Does PACE Authentication Work?

Below are listed the steps needed to execute PACE protocol:

1. Card power-up or re-set
2. Determine if mode is contact or contactless (based on the ATR/ATS historical bytes)
3. Read CardAccess EF to determine how card has been personalized with PACE.
4. Do PACE authentication, then select eID application using PACE secure messaging and continue communication using PACE secure messaging.

2.1.3 PACE Authentication Levels

This section provides a brief description of the different authentication levels.

2.1.3.1 PACE Authentication Only

PACE authentication means that a secret shared by the card and terminal must be successfully presented in order to access the eID application. In eID apllication context following passwords are used.

- Machine Readable Zone (MRZ) : not for Digital/Diplomatic ID Document types.
- Card Access Number (CAN)

2.1.3.2 PACE Authentication + Secure Messaging

PACE secure messaging is applied to an APDU at the platform level. The PACE SM is then unwrapped and the APDU data is available in clear text for the eID application (with no possibility of the external world seeing this data in clear text, but note the Caution in the paragraph "PACE and eID application Secure Messaging" below). After being processed by the eID application, the APDU is then wrapped in PACE SM again before being returned to the external world.

2.1.3.3 PACE and eID application Secure Messaging

The important point to note is that PACE SM and eID application SM are never in use at the same time.

Once eID application SM is established, the PACE SM "sleeps" until the eID application completes the APDU processing and returns its SW (and maybe DataOut).

Note: PACE SM sleeps only on the logical channel on which eID application is selected and is still active on other logical channels.

PACE SM wakes under one of the following circumstances:

- eID application secure messaging error
- eID application is reselected.

Caution: When a eID application AES SM session is in progress, sending an APDU in clear text is considered as an SM error and therefore wakes PACE SM. In 3DES SM, it is not considered as an error, therefore the 3DES SM session is not broken and PACE SM does not take over. Therefore the only way to guarantee protection against eavesdropping with the PACE SM is to make sure that the eID application SM is AES.

2.1.3.4 Effect of Breaking PACE SM

If PACE SM is broken (for example, due to an APDU sent with incorrect SM or sent in clear when SM was expected) then all applications using PACE SM on other logical channels are deselected on their respective channels.

2.1.4 Impacts of PACE

2.1.4.1.1 Effect on APDU Command and Response Length

When using PACE SM, the APDU commands are unwrapped before being sent to the eID application. However, because some bytes are taken up with the SM construction, the unwrapped APDU command is smaller. The effect of this is that there is a stricter limitation on the number of bytes available for APDU commands at the eID application level.

The same is true when the eID application returns data to the software application. It has to allow for the response to be sent in PACE SM and therefore for some bytes to be used for the SM. Therefore fewer bytes are available for the response at the eID application level.

The number of bytes taken up by PACE SM construction differs according to whether the PACE SM is DES or AES.

The maximum number of bytes available for APDU commands and responses are

- For DES PACE SM: 239 bytes
- For AES PACE SM: 231 bytes

2.2 Contact interface

Supported protocols: T=0 and T=1

Default protocol: T=0

2.2.1 ATR bytes

The table below describes the ATR (Answer To Reset) bytes of the card according to ISO 7816-3 and 7816-4 when contact communication is used.

Character	Value (hex)	Comment
TS	3B	Initial character: direct convention
T0	FF	Format character: 'F' indicates that TA1, TB1, TC1 and TD1 are present, 'F' indicates the number of historical characters (15).
TA1	96	Fi = 512 (clock rate conversion integer), Di = 32 (baud rate adjustment integer), f(max.) =5 MHz.

TB1	00			VPP not required.
TC1	00			Indicates the amount of extra guard time required.
TD1	80			Indicate further interface characters. Y = 1000. Protocol T = 0 [Asynchronous half duplex character transmission]
TD2	31			Indicate further interface characters. Y = 0011. Protocol T = 1 [Asynchronous half duplex block transmission]
TA3	FE			Maximum length of information field : IFSC = 254
TB3	43			Character waiting time : CWT = 19.0 etu (CWI = 3). Block waiting time : BWT = 11 etu + 1.60 s
T1	80			Historical characters, max.15 bytes (T1-T15): T1 = Category Indicator, 80 = status information, if present, is contained in an optional COMPACT-TLV data object.
T2		31		Card service data (tag 3, length 1)
T3			B8	Application selection: by full DF name. DOs available: in EF.DIR and in EF.ATR. EF.DIR and AF.ATR access services: by the READ BINARY command (transparent structure). Card with MF.
T4		53		Card Issuer's data (tag 5, length 3)
T5			65	'e'
T6			49	'I'
T7			44	'D'
T8		64		Pre-issuing data (tag 6, length 4).
T9			B0	Family name.
T10			85	Product name.
T11			05	OS version.
T12			10	Program version.
T13		12		Country code and subsequent data (tag 1, length 2). A country indicator consists of a country code (three quartets with values from '0' to '9'), see ISO 3166-1(21) followed by subsequent data (at least one quartet, odd number of quartets). Country code = EST=233 = 0x233 = 0010 0011 0011 b Subsequent data = 1111 b
T14			23	
T15			3F	
TCK	XX			Check byte TCK

2.3 Contactless interface

Used protocol: T=CL type A

2.3.1 ATS bytes

The table below describes the ATS (Answer To Select) bytes of the card according to ISO 14443-4 and 7816-4 when contactless communication is used.

Character	Value (hex)	Comment
TL	14	Total length of the ATS (including TL).
T0	78	TA1 & TB1 & TC are present, Maximum frame size (called FSCI, codes FSC) = 256 bytes.
TA1	77	106, 212, 424 and 847 kilobits supported; reception and transmission may use different bit rates.
TB1	95	The most significant half-byte b8 to b5 is called FWI and codes FWT = 154 ms, the least significant half byte b4 to b1 is called SFGI and codes a multiplier value used to define the SFGT.
TC1	02	CID supported, NAD not supported.
HC1	80	Historical characters, max.15 bytes (T1-T15): T1 = Category Indicator, 80 = status information, if present, is contained in an optional COMPACT-TLV data object.
HC2	31	C-TLV Card service data, length 1 byte
HC3	D8	Application selection: by full DF name + by partial DF name ; DOs available: in EF.ATR/INFO; EF.DIR and EF.ATR/INFO access services: by the read binary command (transparent structure); Card with MF.
HC4	53	Card Issuer's data (tag 5, length 3)
HC5	65	'e'
HC6	49	'I'
HC7	44	'D'
HC8	64	Pre-issuing data (tag 6, length 4).
HC9	B0	Family name.
HC10	85	Product name.
HC11	05	OS version.
HC12	10	Program version.
HC13	12	Country code and subsequent data (tag 1, length 2).
HC14	23	A country indicator consists of a country code (three quartets with values from '0' to '9'), see ISO 3166-1(21) followed by subsequent data (at least one quartet, odd number of quartets).
HC15	3F	Country code = EST=233 = 0x233 = 0010 0011 0011 b Subsequent data = 1111 b
CRC1	XX	CRC1
CRC2	XX	CRC2

3 eID application

3.1 PIN and PUK codes

PIN1 (authentication PIN) protects the usage of authentication key. PIN1 is also required during the re-key operation. PIN2 (signature PIN) protects the usage of signature key and dedicated only for that purpose. PUK (PIN unblocking key) is used to unblock PIN1/PIN2 codes when those codes are locked after 3 unsuccessful verifications.

Setting	PIN1	PIN2	PUK
Label (EF.AOD)	"authentication PIN"	"signature PIN"	"PIN unblocking key"
PIN reference	0x81	0x82	0x83
Global/local	Local	Local	Local
Coding	Numeric	Numeric	Numeric
Min. length	4	5	8
Stored length	12	12	12
Try limit	3	3	3
Change Condition	PIN1	PIN2	NEV
Unblock Condition	PUK	PUK	NEV
Unblock Counter	No limit	No limit	NEV
Usage Counter	No limit	No limit	No limit
Non-repudiation	No	Yes	No
Enforce PIN change before first use	No	No	No
PIN complexity rule	NOT USED	NOT USED	NOT USED

3.2 Key pairs and certificates

Two EC key pairs and certificates are stored on the eID application. One for authentication and one for signature creation purposes. The usage of authentication key is protected by authentication PIN (PIN1) and the usage of signature key is protected by signature PIN (PIN2). Signature PIN must be verified before each signature creation operation (signature PIN verification status is set to unverified after each signature key operation). Authentication PIN keeps the verified status once it has been verified until chip reset or eID application re-selection is performed.

Label	Access Conditions	X509 key usage	Domain parameters
“authentication key”	Get Data (public key): ALW Put Data: NEV Generate Public Key Pair: CHV (PIN 1) + MA+SM Decipher, Compute Signature: CHV (PIN 1)	digitalSignature + keyAgreement	secp384r1 / brainpoolP512r1
“authentication certificate”	Read: ALW Delete, Update: CHV (PIN 1) + MA+SM		
“signature key”	Get Data (public key): ALW Put Data: NEV Generate Public Key Pair: CHV (PIN 1) + MA+SM Compute Signature: CHV (PIN 2)	nonRepudiation	secp384r1 / brainpoolP512r1
“signature certificate”	Read: ALW Delete, Update: CHV (PIN 1) + MA+SM		

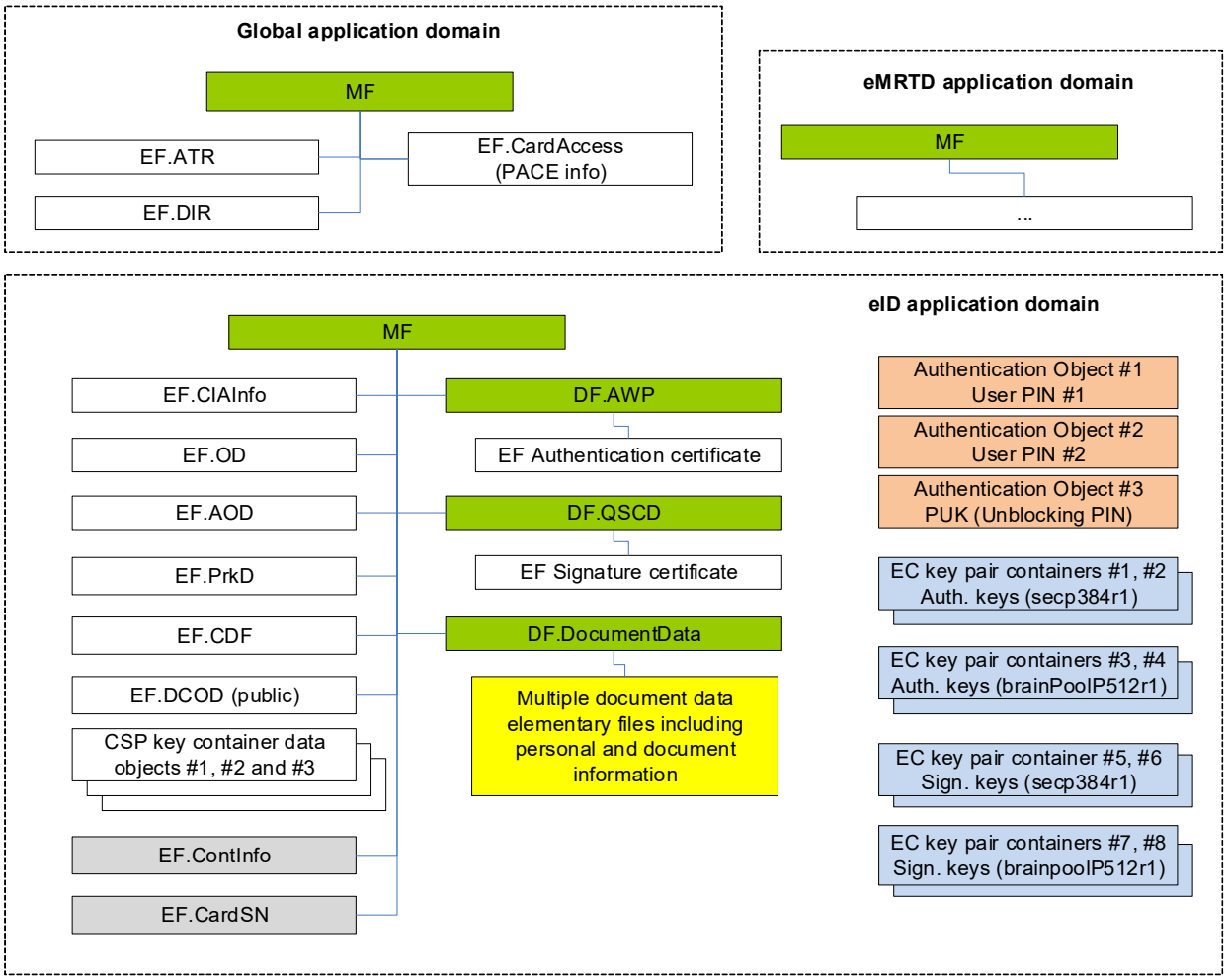
4 Card issuer objects

4.1 Mutual authentication keys

Key ref.	Algorithm	Type	Generation method
1	AES-CBC-256	Card Issuer Authentication key	Derived from Card Issuer master key by CSN

5 eID application file structure

The file structure of the eID application is described in the figure below. It is based on the ISO/IEC 7816-15 specification. Notice that the eID application must be selected prior to being able to access this file structure.



5.1 File and object access condition definitions

File/object types and their access methods:

File/object type	Access method	Meaning
DF	Create	Creation of EFs under current DF
DF	Delete	DF self-deletion
EF	Read	Reading binary file data content
EF	Update	Updating binary file data content
EF	Delete	EF self-deletion
Public key object	Get Data	Read out public key data
Private key object	Compute signature	Computing digital signature
Private key object	Decipher	Deciphering data

Each access method can have the following conditions:

Type	Meaning
NEV	The operation is never allowed, not even after document holder verification
ALW	The operation is always allowed without document holder verification
CHV (PIN X)	The operation is allowed after a successful document holder verification (see ref 3.1)
MA	Mutual authentication with card issuer mutual authentication key (see ref **4.1**)
MA+SM	Mutual authentication and secure messaging with card issuer mutual authentication key (see ref **4.1**)

5.2 Global application domain

5.2.1 EF.ATR

Description

EF.ATR file is used to give additional information to the inspection system about the capabilities of the chip (e.g. if extended length is supported or not). The following table defines the content of EF.ATR file.

Value			Explanation of values
0x47			**Tag for "Card capabilities" data object**
	0x03		**Length:** 3 bytes
		0xB4	**Selection method:** DF selection - by full DF name supported - by partial DF name NOT supported - by path supported - by file identifier supported No implicit DF selection Short EF identifier (SFI) supported Record numbers/identifiers NOT supported
		0x41	**Data coding byte:** - EFs of BER-TLV structure not supported - Behavior of write functions: write OR - value 'FF' is not a valid tag (but padding) - data unit size = one byte
		0xF3	**Command chaining, length fields and logical channels:** - command chaining supported - extended Lc and Le fields supported - extended length information in EF.ATR - logical channels supported, assigned by card, max. number 4.
0x7F66			**Tag for "Extended length information"**
	0x09		**Length:** 9 bytes
		0x020203FC	Command APDU size limitation: 1020 bytes (coded as INTEGER)
		0x020300FFAA	Response APDU size limitation: 65450 bytes (coded as INTEGER)

Access conditions

Access methods	Access condition
Read	ALW
Update	NEV

5.2.2 EF.DIR

Description

This file under MF contains list of Cryptographic Information Applications found from the Document.

Access conditions

Access methods	Access condition
Read	ALW
Update	NEV

Value notation example (ID/RP card)

ISO 7816-15 interpretation

Application templates	Value
application template #1	
aid	'A0000002471001' *(LDS1 eMRTD Application International)*
application template #2	
aid	'A000000063504B43532D3135'
label	"Estonian eID"
path	‘3F00’

Value notation example (Digital/Diplomatic ID)

ISO 7816-15 interpretation

Application templates	Value
application template #1	
aid	'A000000063504B43532D3135'
label	"Estonian eID"
path	‘3F00’

5.2.3 EF.CardAccess

Description

This file contains security information for PACE.

Access conditions

Access methods	Access condition
Read	ALW
Update	NEV

Value notation example

ASN.1

```
SET { -- SecurityInfos
 SEQUENCE { -- Security Info
  OBJECT IDENTIFIER '0 4 0 127 0 7 2 2 4 2 4' -- PACEInfo, id-PACE-ECDH-GM-AES-CBC-CMAC-256
  INTEGER 2 -- version: 2
```

```
INTEGER 16 -- parameterId : BrainpoolP384r1
 }
}
```

5.3 eID application domain

5.3.1 EF.CIAInfo

Description

The CIAInfo file contains generic information about the application as such and its' capabilities. This information includes the serial number, algorithms implemented etc.

Access conditions

Access methods	Access condition
Read	ALW
Update	NEV

Value notation example

ISO 7816-15 interpretation

CIAInfo	Value
version	v2
serialNumber	‘C8C000002D193C7C’
manufacturerID	“ESTeID”
label	“Estonian eID”
cardflags	authRequired, prnGeneration
supportedAlgorithms	
reference	0
algorithm	ckm-ecdsa-sha224
parameters	‘0500’
supportedOperations	compute-signature
objId	"1.2.840.10045.4.3.1"
reference	1
algorithm	ckm-ecdsa-sha256
parameters	‘0500’
supportedOperations	compute-signature
objId	"1.2.840.10045.4.3.2"
reference	2
algorithm	ckm-ecdsa-sha384
parameters	‘0500’
supportedOperations	compute-signature
objId	"1.2.840.10045.4.3.3"
reference	3
algorithm	ckm-ecdsa-sha512
parameters	‘0500’
supportedOperations	compute-signature
objId	"1.2.840.10045.4.3.4"
reference	4
algorithm	ckm-ecdh1-derive
parameters	‘0500’
supportedOperations	derive-key
objId	"1.3.132.1.12"
preferredLanguage	“ee”

ASN.1

```
SEQUENCE SIZE( 168 )
  INTEGER SIZE( 1 )
    0000  01                                                  .
  OCTET-STRING SIZE( 8 )
    0000  42 85 02 53 80 93 04 2A
  UTF8-STRING SIZE( 6 )
    0000  45 53 54 65 49 44                                   ESTeID
  80 [ CONTEXT 0 ] SIZE( 12 )
    0000  45 73 74 6F 6E 69 61 6E 20 65 49 44                 Estonian eID
  BIT-STRING SIZE( 2 )
    0000  05 60                                               `
  A2 [ CONTEXT 2 ] IMPLICIT SEQUENCE SIZE( 123 )
    SEQUENCE SIZE( 23 )
      INTEGER SIZE( 1 )
        0000  00
      INTEGER SIZE( 2 )
```

```
      0000  10 43
    NULL SIZE( 0 )
    BIT-STRING SIZE( 2 )
      0000  06 40
    OBJECT IDENTIFIER = { ecdsa-with-SHA224 }
  SEQUENCE SIZE( 23 )
    INTEGER SIZE( 1 )
      0000  01
    INTEGER SIZE( 2 )
      0000  10 44
    NULL SIZE( 0 )
    BIT-STRING SIZE( 2 )
      0000  06 40
    OBJECT IDENTIFIER = { ecdsa-with-SHA256 }
  SEQUENCE SIZE( 23 )
    INTEGER SIZE( 1 )
      0000  02
    INTEGER SIZE( 2 )
      0000  10 45
    NULL SIZE( 0 )
    BIT-STRING SIZE( 2 )
      0000  06 40
    OBJECT IDENTIFIER = { ecdsa-with-SHA384 }
  SEQUENCE SIZE( 23 )
    INTEGER SIZE( 1 )
      0000  03
    INTEGER SIZE( 2 )
      0000  10 46
    NULL SIZE( 0 )
    BIT-STRING SIZE( 2 )
      0000  06 40
    OBJECT IDENTIFIER = { ecdsa-with-SHA512 }
  SEQUENCE SIZE( 21 )
    INTEGER SIZE( 1 )
      0000  04
    INTEGER SIZE( 2 )
      0000  10 50
    NULL SIZE( 0 )
    BIT-STRING SIZE( 3 )
      0000  07 00 80
    OBJECT IDENTIFIER = { id-ecDH }
PRINTABLE-STRING SIZE( 2 )
  0000  65 65                                             ee
```

5.3.2 EF.OD

Description

The Object Directory (OD) file is a transparent elementary file, which contains pointers to other elementary files (PrKDs, CDs, AODs, DCODs) of the eID application.

An off-card application (middleware software) using the eID application shall use this file to determine how to perform security services with the card.

Access conditions

Access method	Access condition
Read	ALW
Update	NEV

Value notation example

ISO 7816-15 interpretation

CIOChoice ::=	Value
authObjects : path : efidOrPath	'3F005006'
privateKeys : path : efidOrPath	'3F005001'
certificates : path : efidOrPath	'3F005003'
dataContainerObjects : path : efidOrPath	'3F005005'

ASN.1

```
A8 [ CONTEXT 8 ] IMPLICIT SEQUENCE SIZE( 8 )
  SEQUENCE SIZE( 6 )
    OCTET-STRING SIZE( 4 )
      0000  3F 00 50 06
A0 [ CONTEXT 0 ] IMPLICIT SEQUENCE SIZE( 8 )
  SEQUENCE SIZE( 6 )
    OCTET-STRING SIZE( 4 )
      0000  3F 00 50 01
A4 [ CONTEXT 4 ] IMPLICIT SEQUENCE SIZE( 8 )
  SEQUENCE SIZE( 6 )
    OCTET-STRING SIZE( 4 )
      0000  3F 00 50 03
A7 [ CONTEXT 7 ] IMPLICIT SEQUENCE SIZE( 8 )
  SEQUENCE SIZE( 6 )
    OCTET-STRING SIZE( 4 )
      0000  3F 00 50 05
```

5.3.3 EF.AOD

Description

This elementary file (Authentication Object Directory) contains generic authentication object attributes such as allowed characters, length, padding character, etc. The authentication objects are used to control access to other objects such as keys. The contents of this file is according to ISO/IEC 7816-15.

Access conditions

Access method	Access condition
Read	ALW
Update	NEV

Value notation example

ISO 7816-15 interpretation

AuthenticationObjectChoice ::= pwd	PIN1	PIN2	PUK
commonObjectAttributes			
label	”authentication PIN”	“signature PIN”	“PIN unblocking key”
flags	private, modifiale	private, modifiale	private
authId	’03’	’03’	
classAttributes			
authId	’01’	’02’	’03’
typeAttributes			
pwdFlags	case-sensitive, local, initialized, needs-padding, exchangeRefData,	case-sensitive local, initialized, needs-padding, exchangeRefData	case-sensitive, local, change-disabled, unblock-disabled, initialized, needs-padding, unblockingPassword
pwdType	ascii-numeric	ascii-numeric	ascii-numeric
minLength	4	5	8
storedLength	12	12	12
pwdReference	’0081’	’0082’	’0083’
padChar	’00’	’00’	’00’

ASN.1

```
SEQUENCE SIZE( 59 )
  SEQUENCE SIZE( 27 )
    UTF8-STRING SIZE( 18 )
      0000  61 75 74 68 65 6E 74 69 63 61 74 69 6F 6E 20 50  authentication P
      0010  49 4E                                           IN
    BIT-STRING SIZE( 2 )
      0000  06 C0
    OCTET-STRING SIZE( 1 )
      0000  03
  SEQUENCE SIZE( 3 )
    OCTET-STRING SIZE( 1 )
      0000  01
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 23 )
    SEQUENCE SIZE( 21 )
      BIT-STRING SIZE( 3 )
        0000  04 CC 10
      ENUMERATED SIZE( 1 )
```

```
        0000  01
      INTEGER SIZE( 1 )
        0000  04
      INTEGER SIZE( 1 )
        0000  0C
      80 [ CONTEXT 0 ] SIZE( 2 )
        0000  00 81
      OCTET-STRING SIZE( 1 )
        0000  00
SEQUENCE SIZE( 54 )
  SEQUENCE SIZE( 22 )
    UTF8-STRING SIZE( 13 )
      0000  73 69 67 6E 61 74 75 72 65 20 50 49 4E           signature PIN
    BIT-STRING SIZE( 2 )
      0000  06 C0
    OCTET-STRING SIZE( 1 )
      0000  03
  SEQUENCE SIZE( 3 )
    OCTET-STRING SIZE( 1 )
      0000  02
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 23 )
    SEQUENCE SIZE( 21 )
      BIT-STRING SIZE( 3 )
        0000  04 CC 10
      ENUMERATED SIZE( 1 )
        0000  01
      INTEGER SIZE( 1 )
        0000  05
      INTEGER SIZE( 1 )
        0000  0C
      80 [ CONTEXT 0 ] SIZE( 2 )
        0000  00 82
      OCTET-STRING SIZE( 1 )
        0000  00
SEQUENCE SIZE( 55 )
  SEQUENCE SIZE( 24 )
    UTF8-STRING SIZE( 18 )
      0000  50 49 4E 20 75 6E 62 6C 6F 63 6B 69 6E 67 20 6B  PIN unblocking k
      0010  65 79                                            ey
    BIT-STRING SIZE( 2 )
      0000  07 80
  SEQUENCE SIZE( 3 )
    OCTET-STRING SIZE( 1 )
      0000  03
```

```
A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 22 )
  SEQUENCE SIZE( 20 )
    BIT-STRING SIZE( 2 )
      0000  01 FE
    ENUMERATED SIZE( 1 )
      0000  01
    INTEGER SIZE( 1 )
      0000  08
    INTEGER SIZE( 1 )
      0000  0C
    80 [ CONTEXT 0 ] SIZE( 2 )
      0000  00 83
    OCTET-STRING SIZE( 1 )
      0000  00
```

5.3.4 EF.PrKD

Description

This transparent elementary file (Private Key Directory) contains general key attributes such as labels, intended usage, identifiers etc. When applicable, it contains cross-reference pointers to authentication objects used to protect the access to the keys. It also contains references to the actual key objects.

Initially this file contains key information for one authentication and one signature key. Only information for active keys is stored in the file. File content is changed only during the re-key operation.

Depending on the re-key operation type (if EC key algorithm is changed from its initial value) and the post issuance solution implementation values in *id*, *keyReference* and *namedCurve* attributes could be changed during the re-key operation.

When post issuance solution performs re-key operation it takes current key information from this EF.PrKD file and based on this information it finds the correct information for the new key from EF.ContInfo file.

Access conditions

Access method	Access condition
Read	ALW
Update	CHV (PIN 1) + MA+SM

Value notation example

ISO 7816-15 interpretation

PrivateKeyChoice ::= privateECKey	Authentication key *(secp384r1 / brainPoolP512r1)*	Signature key *(secp384r1 / brainPoolP512r1)*
commonObjectAttributes		
label	”authentication key”	“signature key”
flags	private, modifiale	private, modifiale
authId	’01’	’02’
userConsent	-	1
accessControlRules		
accessMode	execute, pso_cds, pso_dec	execute, pso_cds
securityCondition : authId	'01'	'02'
accessMode	update	update
securityCondition : authReference		
authMethod	secureMessaging, extAuthentication, userAuthentication	secureMessaging, extAuthentication, userAuthentication
seIdentifier	1	1
classAttributes		
id	’11’ / ’12’ / ’13’ / ’14’	’ 21’ / ’22’ / ’23’ / ’24’
usage	sign, derive	nonRepudiation
accessFlags	sensitive, alwaysSensitive, neverExtractable, cardGenerated	sensitive, alwaysSensitive, neverExtractable, cardGenerated
keyReference	1 / 2 / 3 / 4	5 / 6 / 7 / 8
algReference	0, 1, 2, 3, 4	0, 1, 2, 3
typeAttributes		
value indirect : path :		
efidOrPath	’’	‘’
keyInfo		
namedCurve	”1.3.132.0.34" / “1.3.36.3.3.2.8.1.1.13”	”1.3.132.0.34" / “1.3.36.3.3.2.8.1.1.13”

ASN.1

```
A0 [ CONTEXT 0 ] IMPLICIT SEQUENCE SIZE( 106 )
  SEQUENCE SIZE( 53 )
    UTF8-STRING SIZE( 18 )
      0000  61 75 74 68 65 6E 74 69 63 61 74 69 6F 6E 20 6B  authentication k
      0010  65 79                                           ey
    BIT-STRING SIZE( 2 )
      0000  06 C0
    OCTET-STRING SIZE( 1 )
      0000  01
    SEQUENCE SIZE( 24 )
      SEQUENCE SIZE( 7 )
        BIT-STRING SIZE( 2 )
          0000  00 25
        OCTET-STRING SIZE( 1 )
          0000  01
      SEQUENCE SIZE( 13 )
```

```
        BIT-STRING SIZE( 2 )
          0000  06 40
        SEQUENCE SIZE( 7 )
          BIT-STRING SIZE( 2 )
            0000  05 E0
          INTEGER SIZE( 1 )
            0000  01
  SEQUENCE SIZE( 32 )
    OCTET-STRING SIZE( 1 )
      0000  11
    BIT-STRING SIZE( 3 )
      0000  07 20 80
    BIT-STRING SIZE( 2 )
      0000  03 B8
    INTEGER SIZE( 1 )
      0000  01
    A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 15 )
      INTEGER SIZE( 1 )
        0000  00
      INTEGER SIZE( 1 )
        0000  01
      INTEGER SIZE( 1 )
        0000  02
      INTEGER SIZE( 1 )
        0000  03
      INTEGER SIZE( 1 )
        0000  04
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 15 )
    SEQUENCE SIZE( 13 )
      SEQUENCE SIZE( 2 )
        OCTET-STRING SIZE( 0 )
      SEQUENCE SIZE( 7 )
        OBJECT IDENTIFIER = { secp384r1 }
A0 [ CONTEXT 0 ] IMPLICIT SEQUENCE SIZE( 101 )
  SEQUENCE SIZE( 51 )
    UTF8-STRING SIZE( 13 )
      0000  73 69 67 6E 61 74 75 72 65 20 6B 65 79           signature key
    BIT-STRING SIZE( 2 )
      0000  06 C0
    OCTET-STRING SIZE( 1 )
      0000  02
    INTEGER SIZE( 1 )
      0000  01
    SEQUENCE SIZE( 24 )
```

```
    SEQUENCE SIZE( 7 )
      BIT-STRING SIZE( 2 )
        0000  02 24
      OCTET-STRING SIZE( 1 )
        0000  02
    SEQUENCE SIZE( 13 )
      BIT-STRING SIZE( 2 )
        0000  06 40
      SEQUENCE SIZE( 7 )
        BIT-STRING SIZE( 2 )
          0000  05 E0
        INTEGER SIZE( 1 )
          0000  01
SEQUENCE SIZE( 29 )
  OCTET-STRING SIZE( 1 )
    0000  21
  BIT-STRING SIZE( 3 )
    0000  06 00 40
  BIT-STRING SIZE( 2 )
    0000  03 B8
  INTEGER SIZE( 1 )
    0000  05
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 12 )
    INTEGER SIZE( 1 )
      0000  00
    INTEGER SIZE( 1 )
      0000  01
    INTEGER SIZE( 1 )
      0000  02
    INTEGER SIZE( 1 )
      0000  03
A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 15 )
  SEQUENCE SIZE( 13 )
    SEQUENCE SIZE( 2 )
      OCTET-STRING SIZE( 0 )
    SEQUENCE SIZE( 7 )
      OBJECT IDENTIFIER = { secp384r1 }
```

5.3.5 EF.CDF

Description

This transparent elementary file contains attributes and pointers to end user certificates. Certificate attributes are such as labels, key identifiers, pointers to certificate files etc.

Initially this file contains certificate information for one authentication and one signature certificate. Only information for active certificates is stored in the file. File content is changed only during the re-key operation.

Depending on the re-key operation type (if EC key algorithm is changed from its initial value) and the post issuance solution implementation values in *id* and *efidOrPath* attributes could be changed during the re-key operation.

File referenced by efidOrPath is the file where the actual certificate is written.

Access conditions

Access method	Access condition
Read	ALW
Update	CHV (PIN 1) + MA+SM

Value notation example

ISO 7816-15 interpretation

CertificateChoice ::= x509Certificate	Authentication certificate	Signature certificate
commonObjectAttributes		
label	”authentication certificate”	“signature certificate”
flags	Modifiale	modifiale
accessControlRules		
accessMode	Read	read
securityCondition : always	NULL	NULL
accessMode	update, delete	update, delete
securityCondition : authReference		
authMethod	secureMessaging, extAuthentication, userAuthentication	secureMessaging, extAuthentication, userAuthentication
seIdentifier	1	1
classAttributes		
id	’11’ / ’12’ / ’13’ / ’14’	’21’ / ’22’ / ’23’ / ’24’
typeAttributes		
value indirect : path :		
efidOrPath	’3F00ADF13411’	’3F00ADF23421’

ASN.1

```
SEQUENCE SIZE( 78 )
  SEQUENCE SIZE( 57 )
    UTF8-STRING SIZE( 26 )
      0000  61 75 74 68 65 6E 74 69 63 61 74 69 6F 6E 20 63  authentication c
      0010  65 72 74 69 66 69 63 61 74 65                    ertificate
    BIT-STRING SIZE( 2 )
      0000  06 40
    SEQUENCE SIZE( 23 )
      SEQUENCE SIZE( 6 )
        BIT-STRING SIZE( 2 )
          0000  07 80
        NULL SIZE( 0 )
      SEQUENCE SIZE( 13 )
        BIT-STRING SIZE( 2 )
```

```
          0000  04 50
        SEQUENCE SIZE( 7 )
          BIT-STRING SIZE( 2 )
            0000  05 E0
          INTEGER SIZE( 1 )
            0000  01
  SEQUENCE SIZE( 3 )
    OCTET-STRING SIZE( 1 )
      0000  11
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 12 )
    SEQUENCE SIZE( 10 )
      SEQUENCE SIZE( 8 )
        OCTET-STRING SIZE( 6 )
          0000  3F 00 AD F1 34 11
SEQUENCE SIZE( 73 )
  SEQUENCE SIZE( 52 )
    UTF8-STRING SIZE( 21 )
      0000  73 69 67 6E 61 74 75 72 65 20 63 65 72 74 69 66  signature certif
      0010  69 63 61 74 65                                  icate
    BIT-STRING SIZE( 2 )
      0000  06 40
    SEQUENCE SIZE( 23 )
      SEQUENCE SIZE( 6 )
        BIT-STRING SIZE( 2 )
          0000  07 80
        NULL SIZE( 0 )
      SEQUENCE SIZE( 13 )
        BIT-STRING SIZE( 2 )
          0000  04 50
        SEQUENCE SIZE( 7 )
          BIT-STRING SIZE( 2 )
            0000  05 E0
          INTEGER SIZE( 1 )
            0000  01
  SEQUENCE SIZE( 3 )
    OCTET-STRING SIZE( 1 )
      0000  21
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 12 )
    SEQUENCE SIZE( 10 )
      SEQUENCE SIZE( 8 )
        OCTET-STRING SIZE( 6 )
          0000  3F 00 AD F2 34 21
```

5.3.6 EF.DCOD

Description

This transparent elementary file contains attributes and pointers to data objects. These data objects are used by Thales middleware in its Smart Card Minidriver/CSP implementation.

Initially this file contains key container information for one authentication key (AT_KEYEXCHANGE) and one signature key (AT_SIGNATURE). Only information for active key containers is stored in the file. File content could be changed only during the re-key operation.

File referenced by efidOrPath is the file where the actual data object data is written.

Access conditions (also for elementary files B101/B102/B103 listed below)

Access method	Access condition
Read	ALW
Update	CHV (PIN 1) + MA+SM

Value notation example

ISO 7816-15 interpretation

DataContainerObjectChoice ::= opaqueDO	Key container referring to authentication key	Key container referring to signature key	Default Key Container
commonObjectAttributes			
label	”812F2422-C7EB-4542-817A-E54A81DE2300”	“48A83664-D45B-4960-8339-71BDC9136ABA”	“Default Key Container”
flags	Modifiale	modifiale	modifiale
accessControlRules			
accessMode	read	read	read
securityCondition : always	NULL	NULL	NULL
accessMode	update,	update	update
securityCondition : authReference			
authMethod	secureMessaging, extAuthentication, userAuthentication	secureMessaging, extAuthentication, userAuthentication	secureMessaging, extAuthentication, userAuthentication
seIdentifier	1	1	1
classAttributes			
applicationName	”CSP”	”CSP”	”CSP”
typeAttributes			
value indirect : path :			
efidOrPath	’3F00B101’	’3F00B102’	’3F00B103’

ASN.1

```
SEQUENCE SIZE( 92 )
  SEQUENCE SIZE( 67 )
    UTF8-STRING SIZE( 36 )
      0000  38 31 32 46 32 34 32 32 2D 43 37 45 42 2D 34 35  812F2422-C7EB-45
      0010  34 32 2D 34 32 38 35 2D 30 32 35 33 38 30 39 33  42-4285-02538093
      0020  30 34 32 41                                      042A
```

```
    BIT-STRING SIZE( 2 )
      0000  06 40
    SEQUENCE SIZE( 23 )
      SEQUENCE SIZE( 6 )
        BIT-STRING SIZE( 2 )
          0000  07 80
        NULL SIZE( 0 )
      SEQUENCE SIZE( 13 )
        BIT-STRING SIZE( 2 )
          0000  06 40
        SEQUENCE SIZE( 7 )
          BIT-STRING SIZE( 2 )
            0000  05 E0
          INTEGER SIZE( 1 )
            0000  01
  SEQUENCE SIZE( 5 )
    UTF8-STRING SIZE( 3 )
      0000  43 53 50                                          CSP
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 14 )
    SEQUENCE SIZE( 12 )
      OCTET-STRING SIZE( 4 )
        0000  3F 00 B1 01
      INTEGER SIZE( 1 )
        0000  00
      80 [ CONTEXT 0 ] SIZE( 1 )
        0000  06
SEQUENCE SIZE( 92 )
  SEQUENCE SIZE( 67 )
    UTF8-STRING SIZE( 36 )
      0000  34 38 41 38 33 36 36 34 2D 44 34 35 42 2D 34 39  48A83664-D45B-49
      0010  36 30 2D 34 32 38 35 2D 30 32 35 33 38 30 39 33  60-4285-02538093
      0020  30 34 32 41                                      042A
    BIT-STRING SIZE( 2 )
      0000  06 40
    SEQUENCE SIZE( 23 )
      SEQUENCE SIZE( 6 )
        BIT-STRING SIZE( 2 )
          0000  07 80
        NULL SIZE( 0 )
      SEQUENCE SIZE( 13 )
        BIT-STRING SIZE( 2 )
          0000  06 40
        SEQUENCE SIZE( 7 )
          BIT-STRING SIZE( 2 )
```

```
            0000  05 E0
          INTEGER SIZE( 1 )
            0000  01
  SEQUENCE SIZE( 5 )
    UTF8-STRING SIZE( 3 )
      0000  43 53 50                                          CSP
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 14 )
    SEQUENCE SIZE( 12 )
      OCTET-STRING SIZE( 4 )
        0000  3F 00 B1 02
      INTEGER SIZE( 1 )
        0000  00
      80 [ CONTEXT 0 ] SIZE( 1 )
        0000  06
SEQUENCE SIZE( 77 )
  SEQUENCE SIZE( 52 )
    UTF8-STRING SIZE( 21 )
      0000  44 65 66 61 75 6C 74 20 4B 65 79 20 43 6F 6E 74  Default Key Cont
      0010  61 69 6E 65 72                                    ainer
    BIT-STRING SIZE( 2 )
      0000  06 40
    SEQUENCE SIZE( 23 )
      SEQUENCE SIZE( 6 )
        BIT-STRING SIZE( 2 )
          0000  07 80
        NULL SIZE( 0 )
      SEQUENCE SIZE( 13 )
        BIT-STRING SIZE( 2 )
          0000  06 40
        SEQUENCE SIZE( 7 )
          BIT-STRING SIZE( 2 )
            0000  05 E0
          INTEGER SIZE( 1 )
            0000  01
  SEQUENCE SIZE( 5 )
    UTF8-STRING SIZE( 3 )
      0000  43 53 50                                          CSP
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 14 )
    SEQUENCE SIZE( 12 )
      OCTET-STRING SIZE( 4 )
        0000  3F 00 B1 03
      INTEGER SIZE( 1 )
        0000  00
      80 [ CONTEXT 0 ] SIZE( 1 )
```

```
0000  24
```

Thales middleware specific data encoding for key containers

File ID	Data	Description
'B101'	01 01 11 02 01 01	Thales proprietary TLV encoding: certificate id: 11 type = 01 (AT_KEYEXCHANGE)
'B102'	01 01 21 02 01 02	certificate id: 21 type = 02 (AT_SIGNATURE)
'B103'	38313246323432322D433745422D3435 34322D383137412D4535344138314445 32333030 *"812F2422-C7EB-4542-817A-E54A81DE2300"*	Defines the default key container of the eID application. Actual data is ascii encodel label of the data object which point to the default key container (certificate id = 11).

5.3.7 EF.ContInfo

Description

This proprietary transparent elementary file contains information about all end user asymmetric key containers created during the personalization phase of the eID application. Key container creation is not possible when the eID application is in operational state (when the card has been delivered to the document holder) which means that key containers needed in post issuance operations (re-key) must be created in advance.

Information for each key container in this file is encoded the same way as in EF.PrKD file. Content of the file cannot be changed. This proprietary file is not part of the ISO 7816-15 standard but internally used by the post issuance solution to find proper parameters for a new asymmetric key pair when the re-key happens (on board asymmetric key pair generation). After successful key pair generation in the used key container the information which refers to this newly generated key pair (and which is stored in this file) will be copied to the EF.PrKD file.

FID = 5034.

Access conditions

Access method	Access condition
Read	ALW
Update	NEV

Value notation example

ISO 7816-15 interpretation (file encoding based on ISO 7816-15 but the file itself is not part of the standard)

PrivateKeyChoice ::= privateECKey	Authentication key containers *(key references #1, #2, #3 and #4)*				Signature key containers *(key references #5, #6, #7 and #8)*			
commonObjectAttributes								
label	”authentication key”				“signature key”			
flags	private, modifiale				private, modifiale			
authId	’01’				’02’			
userConsent	-				1			
accessControlRules								
accessMode	execute, pso_cds, pso_dec				execute, pso_cds			
securityCondition : authId	'01'				'02'			
accessMode	update				update			
securityCondition : authReference								
authMethod	secureMessaging, extAuthentication, userAuthentication				secureMessaging, extAuthentication, userAuthentication			
seIdentifier	1				1			
classAttributes								
Id	’11’	’12’	’13’	’14’	’21’	’22’	’23’	’24’
Usage	sign, derive				nonRepudiation			
accessFlags	sensitive, alwaysSensitive, neverExtractable, cardGenerated				sensitive, alwaysSensitive, neverExtractable, cardGenerated			
keyReference	1	2	3	4	5	6	7	8
algReference	0, 1, 2, 3, 4				0, 1, 2, 3			
typeAttributes								
value indirect : path :								
efidOrPath	’’				‘’			
keyInfo								
namedCurve	”1.3.132.0.34" *(secp384r1)*		“1.3.36.3.3.2.8.1.1.13” *(brainPoolP512r1)*		”1.3.132.0.34" *(secp384r1)*		“1.3.36.3.3.2.8.1.1.13” *(brainPoolP512r1)*	

ASN.1

```
A0 [ CONTEXT 0 ] IMPLICIT SEQUENCE SIZE( 106 )
  SEQUENCE SIZE( 53 )
    UTF8-STRING SIZE( 18 )
      0000  61 75 74 68 65 6E 74 69 63 61 74 69 6F 6E 20 6B  authentication k
      0010  65 79                                           ey
    BIT-STRING SIZE( 2 )
      0000  06 C0
    OCTET-STRING SIZE( 1 )
      0000  01
    SEQUENCE SIZE( 24 )
      SEQUENCE SIZE( 7 )
        BIT-STRING SIZE( 2 )
          0000  00 25
        OCTET-STRING SIZE( 1 )
          0000  01
      SEQUENCE SIZE( 13 )
        BIT-STRING SIZE( 2 )
```

```
          0000  06 40
        SEQUENCE SIZE( 7 )
          BIT-STRING SIZE( 2 )
            0000  05 E0
          INTEGER SIZE( 1 )
            0000  01
  SEQUENCE SIZE( 32 )
    OCTET-STRING SIZE( 1 )
      0000  11
    BIT-STRING SIZE( 3 )
      0000  07 20 80
    BIT-STRING SIZE( 2 )
      0000  03 B8
    INTEGER SIZE( 1 )
      0000  01
    A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 15 )
      INTEGER SIZE( 1 )
        0000  00
      INTEGER SIZE( 1 )
        0000  01
      INTEGER SIZE( 1 )
        0000  02
      INTEGER SIZE( 1 )
        0000  03
      INTEGER SIZE( 1 )
        0000  04
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 15 )
    SEQUENCE SIZE( 13 )
      SEQUENCE SIZE( 2 )
        OCTET-STRING SIZE( 0 )
      SEQUENCE SIZE( 7 )
        OBJECT IDENTIFIER = { secp384r1 }
A0 [ CONTEXT 0 ] IMPLICIT SEQUENCE SIZE( 106 )
  SEQUENCE SIZE( 53 )
    UTF8-STRING SIZE( 18 )
      0000  61 75 74 68 65 6E 74 69 63 61 74 69 6F 6E 20 6B  authentication k
      0010  65 79                                            ey
    BIT-STRING SIZE( 2 )
      0000  06 C0
    OCTET-STRING SIZE( 1 )
      0000  01
    SEQUENCE SIZE( 24 )
      SEQUENCE SIZE( 7 )
        BIT-STRING SIZE( 2 )
```

```
          0000  00 25
        OCTET-STRING SIZE( 1 )
          0000  01
      SEQUENCE SIZE( 13 )
        BIT-STRING SIZE( 2 )
          0000  06 40
        SEQUENCE SIZE( 7 )
          BIT-STRING SIZE( 2 )
            0000  05 E0
          INTEGER SIZE( 1 )
            0000  01
  SEQUENCE SIZE( 32 )
    OCTET-STRING SIZE( 1 )
      0000  12
    BIT-STRING SIZE( 3 )
      0000  07 20 80
    BIT-STRING SIZE( 2 )
      0000  03 B8
    INTEGER SIZE( 1 )
      0000  02
    A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 15 )
      INTEGER SIZE( 1 )
        0000  00
      INTEGER SIZE( 1 )
        0000  01
      INTEGER SIZE( 1 )
        0000  02
      INTEGER SIZE( 1 )
        0000  03
      INTEGER SIZE( 1 )
        0000  04
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 15 )
    SEQUENCE SIZE( 13 )
      SEQUENCE SIZE( 2 )
        OCTET-STRING SIZE( 0 )
      SEQUENCE SIZE( 7 )
        OBJECT IDENTIFIER = { secp384r1 }
A0 [ CONTEXT 0 ] IMPLICIT SEQUENCE SIZE( 110 )
  SEQUENCE SIZE( 53 )
    UTF8-STRING SIZE( 18 )
      0000  61 75 74 68 65 6E 74 69 63 61 74 69 6F 6E 20 6B  authentication k
      0010  65 79                                            ey
    BIT-STRING SIZE( 2 )
      0000  06 C0
```

```
    OCTET-STRING SIZE( 1 )
      0000  01
    SEQUENCE SIZE( 24 )
      SEQUENCE SIZE( 7 )
        BIT-STRING SIZE( 2 )
          0000  00 25
        OCTET-STRING SIZE( 1 )
          0000  01
      SEQUENCE SIZE( 13 )
        BIT-STRING SIZE( 2 )
          0000  06 40
        SEQUENCE SIZE( 7 )
          BIT-STRING SIZE( 2 )
            0000  05 E0
          INTEGER SIZE( 1 )
            0000  01
  SEQUENCE SIZE( 32 )
    OCTET-STRING SIZE( 1 )
      0000  13
    BIT-STRING SIZE( 3 )
      0000  07 20 80
    BIT-STRING SIZE( 2 )
      0000  03 B8
    INTEGER SIZE( 1 )
      0000  03
    A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 15 )
      INTEGER SIZE( 1 )
        0000  00
      INTEGER SIZE( 1 )
        0000  01
      INTEGER SIZE( 1 )
        0000  02
      INTEGER SIZE( 1 )
        0000  03
      INTEGER SIZE( 1 )
        0000  04
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 19 )
    SEQUENCE SIZE( 17 )
      SEQUENCE SIZE( 2 )
        OCTET-STRING SIZE( 0 )
      SEQUENCE SIZE( 11 )
        OBJECT IDENTIFIER = { brainpoolP512r1 }
A0 [ CONTEXT 0 ] IMPLICIT SEQUENCE SIZE( 110 )
  SEQUENCE SIZE( 53 )
```

```
  UTF8-STRING SIZE( 18 )
    0000  61 75 74 68 65 6E 74 69 63 61 74 69 6F 6E 20 6B  authentication k
    0010  65 79                                            ey
  BIT-STRING SIZE( 2 )
    0000  06 C0
  OCTET-STRING SIZE( 1 )
    0000  01
  SEQUENCE SIZE( 24 )
    SEQUENCE SIZE( 7 )
      BIT-STRING SIZE( 2 )
        0000  00 25
      OCTET-STRING SIZE( 1 )
        0000  01
    SEQUENCE SIZE( 13 )
      BIT-STRING SIZE( 2 )
        0000  06 40
      SEQUENCE SIZE( 7 )
        BIT-STRING SIZE( 2 )
          0000  05 E0
        INTEGER SIZE( 1 )
          0000  01
SEQUENCE SIZE( 32 )
  OCTET-STRING SIZE( 1 )
    0000  14
  BIT-STRING SIZE( 3 )
    0000  07 20 80
  BIT-STRING SIZE( 2 )
    0000  03 B8
  INTEGER SIZE( 1 )
    0000  04
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 15 )
    INTEGER SIZE( 1 )
      0000  00
    INTEGER SIZE( 1 )
      0000  01
    INTEGER SIZE( 1 )
      0000  02
    INTEGER SIZE( 1 )
      0000  03
    INTEGER SIZE( 1 )
      0000  04
A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 19 )
  SEQUENCE SIZE( 17 )
    SEQUENCE SIZE( 2 )
```

```
        OCTET-STRING SIZE( 0 )
      SEQUENCE SIZE( 11 )
        OBJECT IDENTIFIER = { brainpoolP512r1 }
A0 [ CONTEXT 0 ] IMPLICIT SEQUENCE SIZE( 101 )
  SEQUENCE SIZE( 51 )
    UTF8-STRING SIZE( 13 )
      0000  73 69 67 6E 61 74 75 72 65 20 6B 65 79           signature key
    BIT-STRING SIZE( 2 )
      0000  06 C0
    OCTET-STRING SIZE( 1 )
      0000  02
    INTEGER SIZE( 1 )
      0000  01
    SEQUENCE SIZE( 24 )
      SEQUENCE SIZE( 7 )
        BIT-STRING SIZE( 2 )
          0000  02 24
        OCTET-STRING SIZE( 1 )
          0000  02
      SEQUENCE SIZE( 13 )
        BIT-STRING SIZE( 2 )
          0000  06 40
        SEQUENCE SIZE( 7 )
          BIT-STRING SIZE( 2 )
            0000  05 E0
          INTEGER SIZE( 1 )
            0000  01
  SEQUENCE SIZE( 29 )
    OCTET-STRING SIZE( 1 )
      0000  21
    BIT-STRING SIZE( 3 )
      0000  06 00 40
    BIT-STRING SIZE( 2 )
      0000  03 B8
    INTEGER SIZE( 1 )
      0000  05
    A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 12 )
      INTEGER SIZE( 1 )
        0000  00
      INTEGER SIZE( 1 )
        0000  01
      INTEGER SIZE( 1 )
        0000  02
      INTEGER SIZE( 1 )
```

```
      0000  03
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 15 )
    SEQUENCE SIZE( 13 )
      SEQUENCE SIZE( 2 )
        OCTET-STRING SIZE( 0 )
      SEQUENCE SIZE( 7 )
        OBJECT IDENTIFIER = { secp384r1 }
A0 [ CONTEXT 0 ] IMPLICIT SEQUENCE SIZE( 101 )
  SEQUENCE SIZE( 51 )
    UTF8-STRING SIZE( 13 )
      0000  73 69 67 6E 61 74 75 72 65 20 6B 65 79           signature key
    BIT-STRING SIZE( 2 )
      0000  06 C0
    OCTET-STRING SIZE( 1 )
      0000  02
    INTEGER SIZE( 1 )
      0000  01
    SEQUENCE SIZE( 24 )
      SEQUENCE SIZE( 7 )
        BIT-STRING SIZE( 2 )
          0000  02 24
        OCTET-STRING SIZE( 1 )
          0000  02
      SEQUENCE SIZE( 13 )
        BIT-STRING SIZE( 2 )
          0000  06 40
        SEQUENCE SIZE( 7 )
          BIT-STRING SIZE( 2 )
            0000  05 E0
          INTEGER SIZE( 1 )
            0000  01
  SEQUENCE SIZE( 29 )
    OCTET-STRING SIZE( 1 )
      0000  22
    BIT-STRING SIZE( 3 )
      0000  06 00 40
    BIT-STRING SIZE( 2 )
      0000  03 B8
    INTEGER SIZE( 1 )
      0000  06
    A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 12 )
      INTEGER SIZE( 1 )
        0000  00
      INTEGER SIZE( 1 )
```

```
        0000  01
      INTEGER SIZE( 1 )
        0000  02
      INTEGER SIZE( 1 )
        0000  03
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 15 )
    SEQUENCE SIZE( 13 )
      SEQUENCE SIZE( 2 )
        OCTET-STRING SIZE( 0 )
      SEQUENCE SIZE( 7 )
        OBJECT IDENTIFIER = { secp384r1 }
A0 [ CONTEXT 0 ] IMPLICIT SEQUENCE SIZE( 105 )
  SEQUENCE SIZE( 51 )
    UTF8-STRING SIZE( 13 )
      0000  73 69 67 6E 61 74 75 72 65 20 6B 65 79           signature key
    BIT-STRING SIZE( 2 )
      0000  06 C0
    OCTET-STRING SIZE( 1 )
      0000  02
    INTEGER SIZE( 1 )
      0000  01
    SEQUENCE SIZE( 24 )
      SEQUENCE SIZE( 7 )
        BIT-STRING SIZE( 2 )
          0000  02 24
        OCTET-STRING SIZE( 1 )
          0000  02
      SEQUENCE SIZE( 13 )
        BIT-STRING SIZE( 2 )
          0000  06 40
        SEQUENCE SIZE( 7 )
          BIT-STRING SIZE( 2 )
            0000  05 E0
          INTEGER SIZE( 1 )
            0000  01
  SEQUENCE SIZE( 29 )
    OCTET-STRING SIZE( 1 )
      0000  23
    BIT-STRING SIZE( 3 )
      0000  06 00 40
    BIT-STRING SIZE( 2 )
      0000  03 B8
    INTEGER SIZE( 1 )
      0000  07
```

```
    A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 12 )
      INTEGER SIZE( 1 )
        0000  00
      INTEGER SIZE( 1 )
        0000  01
      INTEGER SIZE( 1 )
        0000  02
      INTEGER SIZE( 1 )
        0000  03
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 19 )
    SEQUENCE SIZE( 17 )
      SEQUENCE SIZE( 2 )
        OCTET-STRING SIZE( 0 )
      SEQUENCE SIZE( 11 )
        OBJECT IDENTIFIER = { brainpoolP512r1 }
A0 [ CONTEXT 0 ] IMPLICIT SEQUENCE SIZE( 105 )
  SEQUENCE SIZE( 51 )
    UTF8-STRING SIZE( 13 )
      0000  73 69 67 6E 61 74 75 72 65 20 6B 65 79           signature key
    BIT-STRING SIZE( 2 )
      0000  06 C0
    OCTET-STRING SIZE( 1 )
      0000  02
    INTEGER SIZE( 1 )
      0000  01
    SEQUENCE SIZE( 24 )
      SEQUENCE SIZE( 7 )
        BIT-STRING SIZE( 2 )
          0000  02 24
        OCTET-STRING SIZE( 1 )
          0000  02
      SEQUENCE SIZE( 13 )
        BIT-STRING SIZE( 2 )
          0000  06 40
        SEQUENCE SIZE( 7 )
          BIT-STRING SIZE( 2 )
            0000  05 E0
          INTEGER SIZE( 1 )
            0000  01
  SEQUENCE SIZE( 29 )
    OCTET-STRING SIZE( 1 )
      0000  24
    BIT-STRING SIZE( 3 )
      0000  06 00 40
```

```
    BIT-STRING SIZE( 2 )
      0000  03 B8
    INTEGER SIZE( 1 )
      0000  08
    A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 12 )
      INTEGER SIZE( 1 )
        0000  00
      INTEGER SIZE( 1 )
        0000  01
      INTEGER SIZE( 1 )
        0000  02
      INTEGER SIZE( 1 )
        0000  03
  A1 [ CONTEXT 1 ] IMPLICIT SEQUENCE SIZE( 19 )
    SEQUENCE SIZE( 17 )
      SEQUENCE SIZE( 2 )
        OCTET-STRING SIZE( 0 )
      SEQUENCE SIZE( 11 )
        OBJECT IDENTIFIER = { brainpoolP512r1 }
```

5.3.8 EF.CardSN

Description

This proprietary file contains the card (chip) serial number used by Thales middleware/post issuance solution. This value is used for mutual authentication. The EF.CardSN should match what is retrieved from the CPLC data (concatenation of IC Fabrication Date + IC Serial Number + IC Batch ID) of the card.

FID = 0001, size = 8 bytes.

Access conditions

Access method	Access condition
Read	ALW
Update	NEV

5.3.9 DF.AWP

Description

DF.AWP is a subdirectory of the eID application where authentication certificate is stored. AID of this directory is "AWP Application" (415750204170706C69636174696F6E).

Access conditions

DF access method	Access condition
Delete DF (self-deletion)	NEV
Create EF	CHV (PIN 1) + MA+SM

5.3.9.1 Authentication Certificate

Description

This file contains X.509 DER encoded document holder's "*authentication certificate*".

File is referred in EF.CDF directory file where some parameters and the path to this file is defined. During the re-key operation this file will be deleted and re-created and depending on the post issuance solution implementation the file id of this file could be changed.

Access conditions

Access method	Access condition
Read	ALW
Update	CHV (PIN 1) + MA+SM
Delete	CHV (PIN 1) + MA+SM

5.3.10 DF.QSCD

DF.QSCD is a subdirectory of the eID application where signature certificate is stored. AID of this directory is "QSCD Application" (51534344204170706C69636174696F6E).

Access conditions

DF access method	Access condition
Delete	NEV
Create	CHV (PIN 1) + MA+SM

5.3.10.1 Signature certificate

Description

This file contains X.509 DER encoded document holder's "*signature certificate*".

File is referred in EF.CDF directory file where some parameters and the path to this file is defined. During the re-key operation this file will be deleted and re-created and depending on the post issuance solution implementation the file id of this file could be changed.

Access conditions

Access method	Access condition
Read	ALW
Update	CHV (PIN 1) + MA+SM
Delete	CHV (PIN 1) + MA+SM

5.3.11 DF.DocumentData

DF.DocumentData is a subdirectory of the eID application where document data is stored. AID of this directory is "Document Data" (446F63756D656E742044617461).

Access conditions

DF access method	Access condition
Delete	NEV
Create	NEV

5.3.11.1 Document data files

Description

These transparent elementary files contain the same document holder data that is printed on the Document. Each data (personal data) element is stored as a separate file. Whether the personal data element content is empty or not the corresponding file always exists. If personal data element content is empty the corresponding file contains one 0x00 byte (file size is 1 byte), otherwise the file size is adjusted to the personal data element size. UTF-8 character encoding is used for all personal data elements.

Access conditions

Access method	Access condition
Read	ALW
Update	NEV
Delete	NEV

Personal data elements for each Document type

File ID	Personal Data element	Format	ID card	RP card	Digital ID	Diplomatic ID
’5001’	PD1 : Surname		Yes	Yes	Yes	Yes
’5002’	PD2 : Given name		Yes	Yes	Yes	Yes
’5003’	PD3 : Sex	X	Yes	Yes	No	No
’5004’	PD4 : Citizenship	ISO3166 alpha-3	Yes	Yes	No	No
’5005’	PD5 : Date of birth	DD MM YYYY	Yes	Yes	No	Yes
’5006’	PD6 : Personal code	99999999999	Yes	Yes	Yes	Yes
’5007’	PD7 : Document number	XX9999999	Yes	Yes	Yes	Yes
’5008’	PD8 : Date of expiry	DD MM YYYY	Yes	Yes	Yes	Yes
’5009’	PD9 : Date of issue	DD MM YYYY	Yes	Yes	Yes	Yes
’5010’	PD10 : Authority		Yes	Yes	No	Yes
’5011’	PD11 : Place of birth		Yes	Yes	No	No
’5012’	PD12 : Type of permit 1		No	Yes	No	No
’5013’	PD13 : Type of permit 2		No	Opt	No	No
’5014’	PD14 : Type of permit 3		No	Opt	No	No
’5015’	PD15 : Remarks front 1		No	Opt	No	No
’5016’	PD16 : Remarks front 2		No	Opt	No	No
’5017’	PD17 : Remarks front 3		No	Opt	No	No
’5018’	PD18 : Remarks back 1		Opt	Opt	No	No
’5019’	PD19 : Remarks back 2		Opt	Opt	No	No
’5020’	PD20 : Mission 1		No	No	No	Yes
’5021’	PD21 : Mission 2		No	No	No	Opt
’5022’	PD22 : Position 1		No	No	No	Yes
’5023’	PD23 : Position 2		No	No	No	Opt

6 APDU command formats

eID application accepts commands and responses in compliance with the application protocol data unit (APDU) format defined by the ISO 7816-4 standard.

6.1 Protocols

6.1.1 T=0 Protocol

If the card is using the T=0 protocol, this converts APDUs into transmission protocol data units (TPDUs). The reader sends TPDU commands to the card and the card returns TPDU responses to the reader, as described in ISO 7816-3. Refer to this standard if you need more information about converting APDUs into TPDUs.

6.1.2 T=1 Protocol

If the card is using the T=1 protocol, the command and response are mapped directly onto the information field of one I-block or the concatenation of the information fields of successive I–blocks that are chained, as described in ISO 7816-3. Refer to this standard if you need more information about the T=1 protocol.

6.1.3 T=CL Protocol

If the card is using the T=CL protocol, the contactless interface transmits APDUs as described in the ISO/IEC 14443–4 standard.

6.2 Command–Response Pairs

Case 1: No Input/No Output

APDU Command

CLA	INS	P1	P2	---	-----	---

APDU Response

-----	SW1	SW2

Case 2: No Input/Output of Expected Length

APDU Command

CLA	INS	P1	P2	---	-----	Le

APDU Response

Data Field	SW1	SW2

Case 3: Input/No Output

APDU Command

CLA	INS	P1	P2	Lc	Data Field	---

APDU Response

-----	SW1	SW2

Case 4: Input/Output of Expected Length

APDU Command

CLA	INS	P1	P2	Lc	Data Field	Le

APDU Response

Data Field	SW1	SW2

For Input/Output of expected length, when a command with response parameters/data is used, the terminal must send a Get Response command to retrieve the response message.

T=0 Protocol:
When using the T=0 protocol, IAS Classic Applet returns status bytes of SW1 = 61h, SW2 = Licc (the total number of bytes in the response) and the terminal sends the Get Response command automatically.

T=1 and T=CL Protocol:
When using the T=1 or T=CL protocol, when Licc (the total number of bytes in the response) is 256 bytes or less, there is no need for Get Response command, eID application returns APDU response as described in the following table:

Licc	Data Returned	Status Bytes Returned	Sequence Open/Closed
Le > Licc	Licc bytes	90h 00h	Closed
Le = Licc	Licc bytes	90h 00h	Closed
Le < Licc	Le	61h XXh (XXh=Licc–Le)	Open

Table 1. Response to Case 4 Command Licc <= 256 bytes

6.3 Large Outgoing Data Retrieval

If the data for an APDU command is too long to fit in a single command, it can be send using one of the following mechanism defined in ISO/IEC 7816-4:

- Command chaining (supported with T=0, T=1 and T=CL)
- Extended length encoding (supported with T=1 and T=CL)

A command can use only either Extended Length Encoding or Command Chaining, but not both at the same time.

6.4 Command Chaining

If the data for an APDU command is too long to fit in a single command, it is "chained" as described in ISO/IEC 7816-4. Once initiated, a chain must be completed before another command (that is not part of the chain) can be sent. If the chain is interrupted, all the data and processing related to the chained command is lost.

Each command in the chain must have the same header except the last command of a chain which has bit 5 of the CLA byte set to 0.

6.5 Extended Length Encoding

The Extended Length Encoding scheme allows you to send commands of up to 65535 bytes and receive responses of up to 65536 bytes.

In Extended Length Encoding, the fifth byte of the command, P3 (which corresponds to the Lc or Le field in Short Length Encoding) is set to 00h. The Lc and Le values are encoded as 16-bit unsigned integers on 2 bytes each:

- For the Lc field, a value from 0001h to FFFFh encodes a numerical value from 1 to 65535 respectively; a value of 0000h is an invalid.
- For the Le field, a value from 0001h to FFFFh encodes a numerical value from 1 to 65535 respectively; a value of 0000h encodes a value of 65536.

It is recommended to use Le = 0000h on case 2 and case 4 commands.

7 Command APDUs

Command	Standard	Functionality
SELECT	Global Platform, Card Specification, version 2.3.1.	Select an application on the card.
SELECT FILE	ISO/IEC 7816-4	Select a file from the card's file system
GET RESPONSE	ISO/IEC 7816-4	Read response data from the card
READ BINARY	ISO/IEC 7816-4	Read binary data from a transparent (binary) file
VERIFY	ISO/IEC 7816-4	Verify reference data presented by user (e.g. PIN) with the reference data stored inside the card. The current verification status can be also queried with this command.
MANAGE SECURITY ENVIRONMENT: **SET**	ISO/IEC 7816-4	Set the security environment (algorithms, keys) that shall be used in the following PERFORM SECURITY OPERATION commands.
PERFORM SECURITY OPERATION: **HASH**	ISO/IEC 7816-8	Calculate a hash code. The algorithm is specified with the MSE command.
PERFORM SECURITY OPERATION: **COMPUTE DIGITAL SIGNATURE**	ISO/IEC 7816-8	Compute a digital signature with a private key. The algorithm and key are specified with the MSE command.
PERFORM SECURITY OPERATION: **DECIPHER**	ISO/IEC 7816-8	Decrypt data with a private key. The algorithm and key are specified with the MSE command.
CHANGE REFERENCE DATA	ISO/IEC 7816-8	Change the current reference data (e.g. PIN)
RESET RETRY COUNTER	ISO/IEC 7816-8	Unlock locked reference data (e.g. PIN)
GET DATA	ISO/IEC 7816-4	Retrieve the public part of a RSA key

7.1 SELECT

The SELECT command selects an application on the card. All successive commands are handled by the selected application until a new application selection is made.

Byte	Value
CLA	00h 0Ch - secure messaging
INS	A4h
P1	04h – select by name (by Application IDentifier (AID))
P2	00h – select first or only occurrence 02h – select next occurrence
Lc	length of subsequent data field
Data	AID
Le	00h

Table 2. SELECT command APDU

Byte	Value
Data	File Control Information (FCI)
SW1-SW2	Status bytes

Table 3. SELECT response APDU

The content of FCI is described in Annex A.

7.2 SELECT FILE

The SELECT FILE command selects a DF or an EF.

If P1, P2 and Lc are all zero, the command selects the root but returns no data. To select the root and return the FCI information, you must select it by its file ID (3F00h).

Note: Some card implementations may process the command (that is, 0x A4 04 02 00) at the card operating system level before sending it to the selected application. Consequently non nominal cases can result in an unexpected status word. For all the application nominal cases, the behaviour is as described here.

7.2.1 Conditions of Use and Security

7.2.1.1 Usage

The MF and DFs can be selected by their file identifier, by their path or by their name, but not by partial name. EFs can be selected by their file identifier or by their path.

The command cannot be used to select a deactivated file.

7.2.1.2 Security

If the EF to be selected is protected by a security attribute, then this security attribute must be fulfilled in order to perform the SELECT FILE command.

The security attribute specifies if prior mutual authentication is necessary, and if so whether or not secure messaging is also necessary.

Note: For clarity, the command is described here with the data in plain text.

7.2.2 Format

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging
INS	A4h
P1	00h - select EF, DF or MF by file identifier 02h - select EF by file identifier under current DF 04h - select DF or MF by name 08h - select file by absolute path from MF
P2	00h - FCI returned in response 04h - FCP returned in response 0Ch - no response

Lc	Empty or length of subsequent data field
Data	P1 = 00h - EF, DF or MF file identifier (or empty = MF) P1 = 02h - EF file identifier P1 = 04h - MF or DF name P1 = 08h - absolute path from MF without the identifier of MF (3F00h)
Le	Empty or maximum length of data expected in response

Table 4. SELECT FILE command APDU

Note: When P1 is 00h, (file selection by file ID), but not under the current DF, all the files in the application domain will be searched in the order of creation. It selects the first file that matches the file ID specified in the data field, which might not be the file intended.

7.2.3 Response

Byte	Value
Data	File Control Information (FCI), File Control Parameters or empty
SW1-SW2	Status bytes

Table 5. SELECT FILE response APDU

The card returns the data requested by the P2 reference control parameter, followed by the SW1, SW2 status codes. The response data is either in the FCI template, the FCP template or does not exist (P2 = 0Ch).

The FCP template contains the file control parameters that were specified when creating the file. The content of FCP is described in Annex A.

The FCI information returned by the command (P2 = 00h) is shown in Annex A.

Caution: When the command is sent with secure messaging, the tag (62h for FCP, 6Fh for FCI) and length of the FCP or FCI template are absent. Two trailer bytes with the value 00h are appended to the end of the FCP/FCI data. This is true for DF and EF selection.

Possible values for the SW1, SW2 status codes are as follows:

SW1	SW2	Description
90h	00h	Command processed without error. No FCI data returned
62h	83h	Deactivated file
64h	00h	Execution error, file descriptor checksum error
67h	00h	Incorrect length Lc

69h	82h	Security status not satisfied, error during secure messaging
69h	99h	Select application failed PACE authentication not open
6Ah	81h	Function not supported (cannot select MF by path)
6Ah	82h	File not found
6Ah	86h	Incorrect P1, P2 parameter

Table 6. SELECT FILE, possible SW1 and SW2 status codes

7.3 GET RESPONSE

The GET RESPONSE command returns response data from the card for case 4 APDU commands.

For example, this command is used in to get response data from commands

- SELECT FILE,
- PERFORM SECURITY OPERATION: COMPUTE DIGITAL SIGNATURE and
- PERFORM SECURITY OPERATION: DECIPHER.

7.3.1 Conditions of Use and Security

7.3.1.1 Usage

The GET RESPONSE command must be used only after command case 4.

7.3.1.2 Security

The GET RESPONSE command can be accessed freely.

7.3.2 Format

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging
INS	C0h
P1	00h
P2	00h
Lc	Empty
Data	Empty
Le	Maximum length of data expected in response

Table 7. GET RESPONSE command APDU

7.3.3 Response

Byte	Value
Data	Value of the response
SW1-SW2	Status bytes

Table 8. GET RESPONSE response APDU

Most status byte values for this command are managed by the card platform. The eID application can return only the following codes:

SW1	SW2	Description
68h	84h	The chaining mechanism is not available for the command.
90h	00h	Successful execution of the command (Le = Licc)

Table 9. GET RESPONSE, possible SW1 and SW2 status codes (eID application)

Once taken over, the card platform can also return the following codes:

SW1	SW2	Description
61h	XXh	Le < Licc, XXh is the remaining number of available data bytes in the card
69h	82h	Security status not satisfied
6Ch	Licc	if Le > Licc

Table 10. GET RESPONSE, possible SW1 and SW2 status codes (card platform)

7.4 READ BINARY

The READ BINARY command reads all or part of a transparent EF.

There are two ways of referencing the EF:

- Implicit selection. In this case, the EF is under the current DF, and is specified by its short file identifier (SFI). P1 specifies the SFI and P2 specifies the start address for the read (called the offset). The command cannot be used for files with an SFI of 1Fh.
- Direct selection. In this case the EF must have been previously selected using the SELECT FILE command. The data is read directly from the current EF. In this case the offset is specified over P1 and P2.

The command reads the number of bytes specified in Le from the offset, up to a maximum of 255 bytes (including secure messaging data if used).

7.4.1 Conditions of Use and Security

7.4.1.1 Usage

The command can be used on any activated EF.

In the case of implicit selection, the EF becomes the current EF after execution of the command.

7.4.1.2 Security

If the EF containing the data to be read is protected by a security attribute, then this security attribute must be fulfilled in order to perform the READ BINARY command.

The security attribute specifies if prior mutual authentication is necessary, and if so whether or not secure messaging is also necessary.

Note: For clarity, the command is described here with the data in plain text.

7.4.2 Format

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging
INS	B0h
P1	See table below
P2	See table below
Lc	Empty
Data	Empty
Le	Number of bytes to read. If empty or zero, the command reads until the end of the file, up to a maximum of 255 bytes (including data for secure messaging if SM specified).

Table 11. READ BINARY command APDU

Coding of P1 and P2									
b8	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**Hex**	**Meaning**
0	-	-	-	-	-	-	-	-	P1-P2 specifies a 15-bit offset of the data to be read
1	-	-	-	-	-	-	-	-	P1 specifies a short FID and P2 specifies an 8-bit offset of the data to be read
1	0	0	x	x	x	x	x	-	– short FID (value domain 1 – 30)

Table 12. READ BINARY command APDU

7.4.3 Response

Byte	Value
Data	Data read from the file
SW1-SW2	Status bytes

Table 13. READ BINARY, coding of P1 and P2

Possible values for the SW1, SW2 status codes are as follows:

SW1	SW2	Description
90h	00h	Command processed without error
62h	82h	End of file reached (offset + Le is higher than the end of the file)
64h	00h	Execution error, file descriptor checksum error.
69h	82h	Security status not satisfied, for example: • The security attributes are not satisfied • Error during secure messaging
69h	85h	Conditions of use not satisfied, for example: Security attribute for the current contact/contactless interface is absent, zero or not BER-TLV coded.
69h	86h	Command not allowed (no current EF)
6Ah	82h	File not found (b8 of P1 = 1 indicating implicit selection, but no SFI given)
6Ah	86h	Incorrect P1, P2 (check that the offset is inside the EF and that the SFI is not 1Fh)
6Bh	00h	Incorrect P1, P2 (offset outside EF)

Table 14. READ BINARY, possible SW1 and SW2 status codes

7.5 VERIFY

The VERIFY command authenticates a user by comparing the PIN entered in the command (the verification PIN) with the reference PIN.

If the VERIFY command is successful, the following actions take place:

- The reference PIN Try Counter is set to PIN Try Limit.
- The reference PIN validated flag is set to true.
- The reference PIN usage counter is decremented by 1 unless its value is FFh (no limit to the number of times the PIN can be used) or 00h (already been used maximum allowed number of times).
- The reference PINs Credentials Counter can be used.

If the Verify command fails, the following actions take place:

- The reference PIN Try Counter is decremented by 1. If the reference PIN Try Counter reaches zero as a result, the command returns 6983h.
- The reference PIN validated flag is set to false.
- The reference PIN usage counter is unchanged.

In "Unverify" Mode:

If the command is used to "unverify" a PIN, a successful execution of the command sets the reference PIN to "unverified".

7.5.1 Conditions of Use and Security

7.5.1.1 Usage

The length of the entered PIN must be equal to length of the PIN in the card, otherwise the VERIFY command fails.

Sending the command with Lc = 00h has a special meaning. It enables you to know if the PIN has already been successfully presented during the current session. If the PIN has been successfully presented already, then the VERIFY command returns a status code of 9000h. If the PIN has not been successfully presented already, the

VERIFY command returns a status code of 63 Cxh, where x is the number of remaining presentation attempts allowed.

7.5.1.2 Security

The PIN Try Counter and the Usage Counter for the reference PIN must not be zero.

The use of secure messaging is determined by the security attribute of the PIN. If secure messaging is specified, the verification PIN provided by the terminal must be of the same length as the reference PIN and its value must match.

Note: For clarity, the command is described with the data in plain text.

7.5.2 Format

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging
INS	20h
P1	00h – Verify mode FFh – Unverify mode
P2	11h – global PIN 81h-8Fh - local PIN
Lc	Empty or length of subsequent data field
Data	Empty or verification data (padded to the correct length). Padding is done according to ISO/IEC 7816-15.
Le	Empty

Table 15. VERIFY command APDU

7.5.3 Response

Byte	Value
Data	Empty
SW1-SW2	Status bytes If Lc = 00h, the command can be used to retrieve the number X of further allowed retries (SW1-SW2 = 63CXh), or to check whether the verification is not required (SW1-SW2 = 9000h).

Table 16. VERIFY response APDU

Possible values for the SW1, SW2 status codes are as follows:

SW1	SW2	Description
90h	00h	Command processed without error
63h	Cxh	Reference PIN not verified.”x” attempts remaining
67h	00h	Incorrect length, Lc.

69h	82h	Security conditions not satisfied (error during secure messaging)
69h	83h	Authentication method blocked (reference PIN blocked)
69h	84h	PIN Try Counter or PIN Usage Counter for reference PIN has reached zero
69h	85h	Conditions of use not satisfied, for example: • Security attribute for the current contact/contactless interface is absent, zero or not BER-TLV coded. (Possible in "Verify" mode but not "Unverify" mode).
69h	99h	PACE authentication not open
6Ah	86h	Incorrect P1, P2
6Ah	88h	Referenced data not found

Table 17. VERIFY, possible SW1 and SW2 status codes

7.6 MANAGE SECURITY ENVIRONMENT: SET

The MANAGE SECURITY ENVIRONMENT: SET (MSE: SET) command updates a CRT in the current SE. The value of the CRT in the original SE remains unchanged.

7.6.1 Conditions of Use and Security

7.6.1.1 Usage

The command applies to the current SE only.

7.6.1.2 Security

The MANAGE SECURITY ENVIRONMENT: SET command can be accessed freely.

Note: For clarity, the command is described here with the data in plain text.

7.6.2 Format

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging 10h - transmit first portion of data in chaining mode with no secure messaging and wait for final portion
INS	22h
P1	01000001b = 41h – computation and decipherment
P2	P1 = SET - P2 = AAh hash template (HT) - P2 = B6h, value of DST in data field - P2 = B8h, value of CT in data field
Lc	Empty or length of subsequent data field
Data	Concatenation of CRDOs. CRDOs are the items contained in the CRT. The data does not contain the tag and length of the CRT.

Le	Empty

Table 18. MSE: SET command APDU

7.6.3 Response

Byte	Value
Data	Empty
SW1-SW2	Status bytes

Table 19 MSE: SET response APDU

Possible values for the SW1, SW2 status codes are as follows:

SW1	SW2	Description
90h	00h	Command processed without error
67h	00h	Incorrect length. Lc > 30
69h	82h	Security status not satisfied, error during secure messaging
69h	85h	Conditions of use not satisfied, for example wrong order of operations or context of use not respected.
6Ah	80h	Incorrect data, for example no algorithm ID referenced
6Ah	86h	Incorrect P1, P2

Table 20. MSE: SET, possible SW1 and SW2 status codes

The table below describes the Control Reference Data Objects (CRDO) that are supported in Digital Signature Templates (DST) and Confidentiality Templates (CT).

Tag	Value	DST	CT
80h	Algorithm reference	+	+
84h	Key reference	+	+

Table 21. Control Reference Data Objects (CRDO)

Supported combinations of P1-P2				
P1	**P2**	**Meaning**	**CRDO in data field**	**Data field contents**
'41'	'AA'	SET SE for hashing	HT	' 80 01 xx '
'41'	'B6'	SET SE for digital signature	DST	' 80 01 xx 84 01 xx '
'41'	'B8'	SET SE for decipherment (RSA use)	CT	' 80 01 xx 84 01 xx ‘
'41'	'B8'	SET SE for decipherment (ELC use)	CT	' 84 01 xx ‘

Table 22. MSE: SET supported P1-P2 combinations

The supported values for the CRDO algorithm reference (tag 80h) are specified in the tables below.

Algorithm reference	Details
0Xh	No hash indicated. Either the hash function is defined implicitly or is not applicable.
1Xh	SHA-1
3Xh	SHA-224
4Xh	SHA–256
5Xh	SHA-384
6Xh	SHA-512
X1h	RSA with padding according to ISO 9796–2.
X2h	**RSASSA-PKCS1-v1_5 signature scheme** (according to PKCS#1 v2.2 with RSA algorithm, compatible with PKCS#1 v1.5)
X3h	RSA with padding according to RFC 2409.
X4h	ECDSA.
X5h	RSA PSS.
XXh	All other values are RFU.

Table 23. Values for the algorithm reference used in Digital Signature Template

The high nibble of the algorithm reference specifies the hash algorithm used (if hashing is relevant for the algorithm). The low nibble specifies the rest of the details about the algorithm.

Algorithm reference	Details
1Ah	RSASSA PKCS#1_v1.5
1Dh	RSAES OAEP SHA-1 (160 bits)
3Dh	RSAES OAEP SHA-224
4Dh	RSAES OAEP SHA–256
5Dh	RSAES OAEP SHA-384
6Dh	RSAES OAEP SHA-512

Table 24. Values for the algorithm reference used in Confidentiality Template

7.7 PERFORM SECURITY OPERATION: HASH

The PERFORM SECURITY OPERATION: HASH (PSO: HASH)provides the hashed message to be used as input for the computation of a digital signature. There are three different cases:

- The hash is performed entirely by card
- The hash is performed externally but card performs the final round of hashing
- The hash is performed entirely outside the card and the PERFORM SECURITY OPERATION: HASH command is used to "set" the data in preparation for the PERFORM SECURITY OPERATION: COMPUTE DIGITAL SIGNATURE command

For information, the following table summarizes the various hash algorithm constants:

Hash	Length of intermediate hash	Length of the counter indicating the number of bits already hashed (bytes) 1	Block length of hash algorithm	Length of message digest result
SHA-1	20 bytes	8 bytes	64 bytes	20 bytes
SHA-224	32 bytes	8 bytes	64 bytes	28 bytes
SHA-256	32 bytes	8 bytes	64 bytes	32 bytes
SHA-384	64 bytes	16 bytes	128 bytes	48 bytes
SHA-512	64 bytes	16 bytes	128 bytes	64 bytes

Table 25. Hash Algorithm Constants

For the sake of simplicity, the counters indicating the number of bits already hashed can be coded on 8 or 16 bytes for all supported hash algorithms. However, when coded on 16 bytes, the 8 MSB (leftmost) bytes must be null. Thus a length coded on 8 or 16 bytes is equivalent.

7.7.1 Conditions of Use and Security

7.7.1.1 Usage

If the data is to be hashed entirely in the card and is more than 64 bytes, divide it into blocks of 64 bytes or less and perform the PERFORM SECURITY OPERATION: HASH command for each block. The final command has a different format (see "APDU Format to Hash the Final Block"). If the data is 64 bytes or less, use the format described for the final command. If a command other than PERFORM SECURITY OPERATION: HASH is sent, the hash session ends and previously hashed data is lost.

The result of the PERFORM SECURITY OPERATION: HASH is available until one of the following occurs:

- The eID application is deselected or reselected
- A further PERFORM SECURITY OPERATION: HASH command is issued
- A MANAGE SECURITY ENVIRONMENT: SET command is issued

7.7.1.2 Security

The current SE must contain a DST template or an HT template and the algorithm ID in this template must be consistent with the format of the command data.

If a MANAGE SECURITY ENVIRONMENT: SET command is sent during a hash sequence which references a different DST key, the hash sequence is aborted and the internal "hash integrity flag" is set to "unverified".

Note: For clarity, the command is described here with the data in plain text.

7.7.2 Format

This command format varies according to where the hash is performed. Each case is shown separately.

7.7.2.1 Hashing Performed Entirely by the Card

If the data is more than one block length, the command must be sent for each block. The block length is 64 bytes or 128 bytes according to the SHA used (see the "block length" column in Hash Algorithm Constants table).

The APDU format for all the blocks except the last one is as follows:

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging

INS	2Ah
P1	90h
P2	80h
Lc	Length of subsequent data field
Data	Data to be hashed (no more than one block length as shown in “block length” column in Hash Algorithm Constants table
Le	Empty.

Table 26. PSO: HASH command APDU for all the blocks except the last one

The APDU format for the last block or the only block if the data is not more than one block length is as follows:

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging
INS	2Ah
P1	90h
P2	A0h
Lc	Length of subsequent data field
Data	Data to be hashed (last block, in TLV format as follows (L is the length of the last data block): 80h || L || last data block).
Le	Length of the hash result. This is optional and is used if you want to return the final value of the hash in the command.

Table 27. PSO: HASH command APDU for the last block or the only block

7.7.2.2 Hashing Performed Partially by the Card

In this case, all the data except the final block is hashed outside the card but the card hashes the final block. The APDU format is as follows:

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging
INS	2Ah
P1	90h
P2	A0h
Lc	Length of subsequent data field
Data	Data is the concatenation of the immediate hash value and the final data block according to the hash algorithm used, as shown in Input Hash Lengths for Different Algorithms (SHAs) table.
Le	Empty.

Table 28. PSO: HASH command APDU for the final block

Tag	Length	Value
90h	XXh	NN–byte intermediate hash value || 8–byte or 16–byte counter (number of bits already hashed). The length of the counter depends on the SHA algorithm as shown in in Input Hash Lengths for Different Algorithms (SHAs) table.
80h	L	Final data block Where: L is the length of the final data block, which must be between 0 and the block lengths as indicated in in Hash Algorithm Constants table.

Table 29. PSO: HASH Input Data (Hash Performed Partially by the Card)

Table 30. Input Hash Lengths for Different Algorithms (SHAs)

Hash	XX	Length of Intermediate Hash (NN bytes) and Counter
SHA-1	1Ch	20 + 8 bytes
SHA-224	28h	32 + 8 bytes
SHA-256	28h	32 + 8 bytes
SHA-384	48h or 50h	64 + 8 bytes or 64 + 16 bytes
SHA-512	48h or 50h	64 + 8 bytes or 64 + 16 bytes

Table 31. Input Hash Lengths for Different Algorithms (SHAs)

7.7.2.3 Hashing Performed Entirely Externally

In this case, all the data is hashed outside the card. The APDU format is as follows:

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging
INS	2Ah
P1	90h
P2	A0h
Lc	Length of subsequent data field
Data	Hash value, in TLV format as follows (L is the length of the hash value according to the algorithm ID): 90h || L || hash value). **Algorithm ID** — **Length of hash value** 01h or 02h — 1–36 bytes 03h — 16–36 bytes 11h or 12h — 20 bytes 32h — 28 bytes 41h or 42h — 32 bytes 52h — 48 bytes 62h — 64 bytes
Le	Empty.

Table 32. PSO: HASH command APDU when all the data is hashed outside the card

7.7.3 Response

Byte	Value
Data	Empty
SW1-SW2	Status bytes

Table 33. PSO: HASH response APDU when all the data is hashed outside the card

Possible values for the SW1, SW2 status codes are as follows:

SW1	SW2	Description
90h	00h	Command processed without error
64h	00h	Incorrect checksum for hash value
68h	84h	The chaining mechanism is not available for the command.
69h	82h	Error during secure messaging
69h	85h	Conditions of use not satisfied, for example: • Algorithm ID must be one of those in Values for the algorithm reference used in Digital Signature Template table • Length of input hash data not valid for algorithm ID

		• Security attribute for the current contact/contactless interface is absent, zero or not BER-TLV coded.
69h	86h	Command not allowed (use of SHA-1 has been prohibited by the cryptographic environment parameters data object)
6Ah	80h	Incorrect data. Input template in incorrect format

Table 34. PSO: HASH, possible SW1 and SW2 status codes

7.8 PERFORM SECURITY OPERATION: COMPUTE DIGITAL SIGNATURE

The PERFORM SECURITY OPERATION: COMPUTE DIGITAL SIGNATURE (PSO: CDS) command is used to compute a digital signature (from either a message or a hash computation).

A message is always hashed before signing. This hash can be performed by in one of three ways:

- entirely by card
- entirely externally
- partially externally and partially by card. In this case, the data is hashed externally but the last block is hashed by the card.

In all cases, card pads the hash to create Digital Signature Input (DSI) and signs this DSI by using the PERFORM SECURITY OPERATION: COMPUTE DIGITAL SIGNATURE command.

The current SE indicates the keys and algorithms to use for the signature and, if applicable, the hash algorithm.

7.8.1 Conditions of Use and Security

7.8.1.1 Usage

The command must be preceded by a PERFORM SECURITY OPERATION: HASH command which provides the hash or sets the hashed data if the hash is performed entirely externally.

None of the following actions must have taken place in between the execution of the PERFORM SECURITY OPERATION: HASH command and the PERFORM SECURITY OPERATION: COMPUTE DIGITAL SIGNATURE command otherwise the hash data is lost:

- Select eID application
- Another completed hash computation
- A MANAGE SECURITY ENVIRONMENT: SET command

The current SE must contain a DST which refers to a private key. This key must be a signature key and must have been initialized.

Caution: The key modulus value must be greater than the value of the padded message to be signed. Remember that messages padded with ISO 9796–2 padding begin with a padding byte of 6Ah, so any modulus value beginning with a value less than 6Ah is not allowed. In such a case, the PERFORM SECURITY OPERATION: COMPUTE DIGITAL SIGNATURE does not return an error code in its status bytes, but generates a signature that has no meaning.

7.8.1.2 Security

In the application phase any security attributes set for the private key regarding the PERFORM SECURITY OPERATION: COMPUTE DIGITAL SIGNATURE must be fulfilled.

Note: For clarity, the command is described here with the data in plain text.

If the signature key has the optional non-repudiation flag activated (value other than 00h or absent), the applet sets the PIN(s) protecting the key to "unverified" after the command (so the PIN needs to be presented again if the key is used again).

Note: If more than one PIN is protecting the signature key and the PINs are to be set to unverified afterwards, the PERFORM SECURITY OPERATION: COMPUTE DIGITAL SIGNATURE command unverifies ONE PIN only (the one that first granted access).

If the signature key has the optional non-repudiation flag deactivated (flag is present and value is 00h), the applet sets the PIN(s) protecting the key to "verified" after the command.

If the signature key is protected by more than one PIN defined in the security attributes as OR, then a correct presentation of any one of the PINs is valid for the PERFORM SECURITY OPERATION: COMPUTE DIGITAL SIGNATURE command. Thus, if a PIN had already been verified, the second PIN could be used to authorize another PERFORM SECURITY OPERATION: COMPUTE DIGITAL SIGNATURE command, without the need to re-verify the first PIN. If the security attributes specified an AND condition, then all the specified PINs need to be verified for ONE use of the PERFORM SECURITY OPERATION: COMPUTE DIGITAL SIGNATURE command.

If the signature was not completed because the APDU processing was interrupted due to the context not being properly established (that is, the applet returns status bytes of 69 82h) the PIN Presentation flag remains untouched.

If the signature key is protected by one or more PINs, and the "Change PIN before first use" option is active, then these PINs must have been changed at least once since personalization for the signature key to be available. In other words, the "changed" flag for the PIN must be true.

If the signature key has a signature counter associated with it, the counter is incremented when the command is performed successfully. If the incremented counter exceeds the maximum allowed value, the command fails.

7.8.2 Format

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging
INS	2Ah
P1	9Eh - digital signature data object is returned in response
P2	9Ah - data field contains data to be signed
Lc	Empty
Data	Empty
Le	Maximum length of the expected length of the response, that is, the length of the signature with no tag or length bytes.

Table 35. PSO: CDS command APDU

Note: If the response is greater than 256 bytes including secure messaging (for example with key lengths of 2,048 bits), the card opens a retrieval sequence. The sequence differs according to the communication protocol in use (T=0 or T=1/T=CL). For details see Annex B.

7.8.3 Response

Byte	Value
Data	Digital signature
SW1-SW2	Status bytes

Table 36. PSO: CDS response APDU

The card returns the value of the digital signature, followed by the SW1, SW2 status codes. RSA signatures are unformatted. ECDSA signatures are in the format r || s, where r and s are integers and are the co-ordinates of the point on the elliptic curve that correspond to the signature value.

Possible values for the SW1, SW2 status codes are as follows:

SW1	SW2	Description
90h	00h	Command processed without error
69h	82h	Security conditions not satisfied, for example: • Security attribute not fulfilled for private key • Error during secure messaging • PIN protecting the signature key has not been presented (it is valid for one signature only) • Integrity of DTBS check failed
69h	85h	Conditions of use not satisfied, for example: • No hashed message available from PSO–Hash command. • No algorithm or private key in the current SE's DST. • Algorithm not recognized. • Private key in current SE's DST is not a digital signature key. • Private key in current SE's DST has not been initialized, that is, at least one of the elements has not been initialized using the Put Data command. • PIN protecting the signature key has not been changed since personalization, and the "Change PIN before first use" option is active. • Security attribute for the current contact/contactless interface is absent, zero or not BER-TLV coded.
69h	86h	Command not allowed (use of SHA-1 has been prohibited by the cryptographic environment parameters data object)
6Ah	84h	Signature counter exceeds maximum allowed value.
6Ah	86h	Incorrect P1, P2. Must be 9Eh 9Ah

Table 37. PSO: CDS, possible SW1 and SW2 status codes

7.9 PERFORM SECURITY OPERATION: DECIPHER (RSA USE)

The PERFORM SECURITY OPERATION: DECIPHER (PSO: DECIPHER) command decrypts a message using a decipher key stored in the card. It returns the message in plain data.

The PERFORM SECURITY OPERATION: DECIPHER command can be used in "chaining" mode. This means that if the data is too large to be sent in a single PERFORM SECURITY OPERATION: DECIPHER command you can send it in two PERFORM SECURITY OPERATION: DECIPHER commands. The first command is sent with CLA = 1Xh. This tells the card to wait for the remaining data. Then issue another PERFORM SECURITY OPERATION: DECIPHER command with CLA= 0Xh. The card concatenates the data from the first command with that of the second and then performs the deciphering operation. You will need to use chaining mode to decipher data that was encrypted using a 2,048–bit key or longer.

7.9.1 Conditions of Use and Security

7.9.1.1 Usage

The private key used to decipher the data must be a decipher key and must be stored in the confidentiality template (CT - tag B8h) of the current SE. The encrypted message must be the same length as this key.

If you send the command with CLA = 1Xh to indicate chaining mode, the next command you issue must be another PERFORM SECURITY OPERATION: DECIPHER command with CLA = 0Xh, otherwise the chaining mechanism ends and the data is lost.

The total amount of data that can be sent in "chaining" mode is 742 bytes. If this figure is exceeded, the chaining session is broken and the card returns the error 6700h.

The message to be decrypted must have been encrypted using one of the following RSAES (RSA encryption schemes):

- RSA with either PKCS #1 v1.5 padding (block type 2 format)
- RSA with OAEP padding. The format of OAEP padding is outside the scope of this document. For more information please refer to the RSA PKCS#1 v2.1 standard.

Caution: OAEP has an optional label (to be associated with the message to encrypt), this label is hashed and concatenated with a padding and the message to encrypt. When deciphering the encoded data, the service uses the same label and verifies its hash (retrieved from the deciphered data). The hash of the label is used as an additional verification token to certify the encrypted data. However it is not available in the JavaCard 2.2.2 API (this is also true for JavaCard 3.0.1).

The PKCS #1 V2.1 RSAES OAEP contains inner parameters. The following parameters have been chosen:

- The inner hash is the same as the external hash
- The Label is an empty string
- The MGF function is the MGF1() defined in PKCS #1 V2.1

7.9.1.2 Security

If specified, the "Decipher" security attribute for the private key referenced in the CT of the current SE must be fulfilled.

Note: For clarity, the command is described here with the data in plain text.

The algorithm ID must be one of those listed in Values for the algorithm reference used in Confidentiality Template table.

7.9.2 Format

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging 10h - transmit first portion of data in chaining mode with no secure messaging and wait for final portion 1Ch - transmit first portion of data in chaining mode with secure messaging and wait for final portion
INS	2Ah
P1	80h - decrypted value is returned in response
P2	86h - cryptogram in data field
Lc	Length of subsequent data field
Data	Cryptogram = PI || encrypted message. The encrypted message must be in the correct format (please refer to the PKCS#1 standard). PI=81h.
Le	Empty or maximum length of data expected in response

Table 38. PSO: DECIPHER command APDU

Note: The total length of the encrypted message (not including the 81h byte and secure messaging) must be the same length as the key used to decipher it.

Byte	Value
Data	Decrypted cryptogram (padding is removed by the card and only the actual data is returned).
SW1-SW2	Status bytes

Table 39. PSO: DECIPHER response APDU

Possible values for the SW1, SW2 status codes are as follows:

SW1	SW2	Description
90h	00h	Command processed without error
67h	00h	Incorrect length, Lc, for example: • Data is not the same length as the private key + 1 for padding indicator (PI) • Total data in chaining mode exceeds 742 bytes
69h	82h	Security status not satisfied: • Security attribute not fulfilled for private key referenced in CT of current SE • Error during secure messaging
69h	84h	Referenced data (private key) invalidated or does not exist
69h	85h	Conditions of use not satisfied, for example: • No algorithm or private key in the current SE's CT • Algorithm ID must be 1A, 1D, 3D, 4D, 5D or 6Dh • Private key in current SE's CT is not a decipher key (that is, bit 2 of the private key usage byte is not set to 1) • Private key in current SE's CT has not been initialized, that is, not all the elements have been updated using the Put Data command • Security attribute for the current contact/contactless interface is absent, zero or not BER-TLV coded.
69h	86h	Command not allowed (Algorithm ID indicates the use of RSAES OAEP algorithm but without a supported related hash method (i.e. algoID is 'xD' with x outside the values [1,3,4,5,6])
6Ah	80h	Incorrect data, for example: • PI must be 81h • The message before encryption and padding was longer than key length - 11 bytes • The message before encryption and padding was not in the format described by Table 121 on page 235 or the correct OAEP format defined in PKCS#1 V2.1 standard. • Incorrect tag or tag length (including TLVs used in secure messaging) • Mismatch between algoID used to cipher the message and the algoID relevant for Decipher

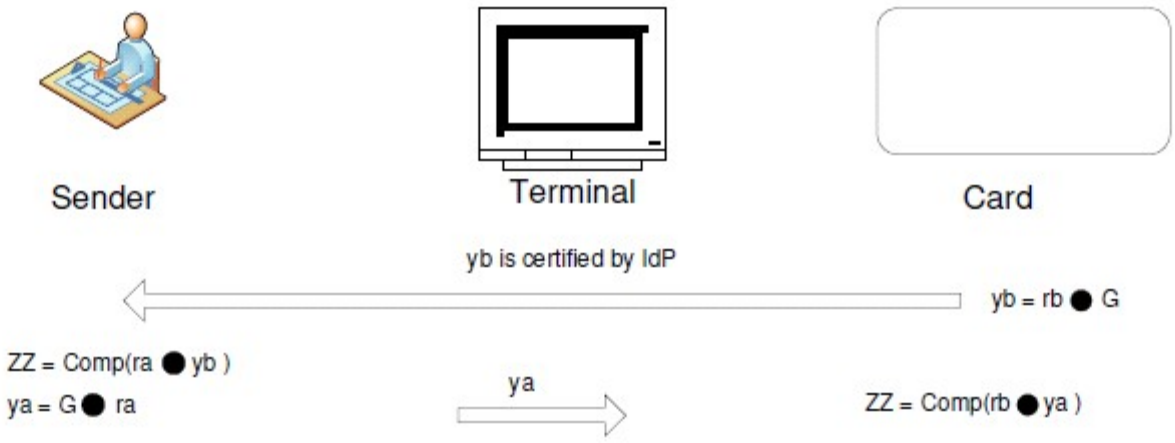
6Ah	86h	Incorrect P1, P2. Must be 80h 86h

Table 40. PSO: DECIPHER, possible SW1 and SW2 status codes

7.10 PERFORM SECURITY OPERATION: DECIPHER (ELC USE)

The command cannot be used to decipher a message encrypted with ELC keys – it is not mathematically possible. However the command is defined in EN 419212 (former EN 14890) as being able to generate a shared key ZZ.

The following figure illustrates how the shared key is generated.



Where:

The terminal knows the Domain (G point)

y_a is the sender's public key (in uncompressed format) made up of $y_a = G \bullet r_a$

y_b is the ICC's certified public key made of $y_b = G ?r_b$

r_a is the IFD's ephemeral private (EC)DH key (randomly generated for freshness)

r_b is the ICC's static private (EC)DH key (generated once)

● is the scalar multiplication over the curve

Note: y_b does not necessarily need to exist in the card after creation. The card keeps r_b to allow the creation of ZZ.

---

with $ZZ = Comp (r_a \bullet y_b) = Comp (r_b \bullet y_a) = Comp ((r_a \bullet r_b) \bullet G)$

Processing:

Phase 1 - Key Certification

1 The card's key token y_b is generated once. y_b is exported and certified and is made public.

Phase 2 - Deciphering of encrypted key

1 The sender provides an encrypted element (encrypted with key ZZ) and y_a its public key

2 The IFD sends the y_a to the ICC in order to recover the decrypted ZZ

3 The IFD uses ZZ to decrypt the element sent by the sender

7.10.1 Conditions of Use and Security

7.10.1.1 Usage

The private key used to generate the symmetric key must be an ELC decipher key and must be stored in the confidentiality template (CT - tag B8h) of the current SE. The CT in the SE must not contain any other key when used with this command. The encrypted message must be the same length as this key.

If you send the command with CLA = 1Xh to indicate chaining mode, the next command you issue must be another PERFORM SECURITY OPERATION: DECIPHER command with CLA = 0Xh, otherwise the chaining mechanism ends and the data is lost.

The total amount of data that can be sent in "chaining" mode is 742 bytes. If this figure is exceeded, the chaining session is broken and the card returns the error 6700h.

7.10.1.2 Security

If specified, the "Decipher" security attribute for the private key referenced in the CT of the current SE must be fulfilled.

Note: For clarity, the command is described here with the data in plain text.

7.10.2 Format

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging 10h - transmit first portion of data in chaining mode with no secure messaging and wait for final portion 1Ch - transmit first portion of data in chaining mode with secure messaging and wait for final portion
INS	2Ah
P1	80h - decrypted value is returned in response
P2	86h - cryptogram in data field
Lc	Length of subsequent data field
Data	Cryptogram = PI || cryptogram. Cryptogram = sender's public key, y_a, in uncompressed format, that is, 04 || X_P || Y_P, where X_P and Y_P represent the points on the elliptic curve. PI=00h (meaning that no padding information given).
Le	The length of the response message, ZZ

Table 41. PSO: DECIPHER (ELC use) command APDU

Note: The total length of the encrypted message (not including the 00h byte and secure messaging) must be the same length as the key used to decipher it.

7.10.3 Response

Byte	Value
Data	Response.
SW1-SW2	Status bytes

Table 42. PSO: DECIPHER (ELC use) response APDU

Possible values for the SW1, SW2 status codes are as follows:

SW1	SW2	Description
90h	00h	Command processed without error
67h	00h	Incorrect length, Lc, for example: • Data is not the same length as the uncompressed ELC private key + 1 for padding indicator
69h	82h	Security status not satisfied: • Security attribute not fulfilled for private key referenced in CT of current SE • Error during secure messaging
69h	84h	Referenced data (private key) invalidated or does not exist
69h	85h	Conditions of use not satisfied, for example: • No algorithm or private key in the current SE's CT • Private key in current SE's CT is not a decipher key (that is, bit 2 of the private key usage byte is not set to 1) • Private key in current SE's CT has not been initialized • Security attribute for the current contact/contactless interface is absent, zero or not BER-TLV coded
69h	86h	Command not allowed
6Ah	80h	Incorrect data, for example: • PI is not 00h
6Ah	86h	Incorrect P1, P2. Must be 80h 86h

Table 43. PSO: DECIPHER (ELC use), SW1 and SW2 status codes

7.11 CHANGE REFERENCE DATA

The CHANGE REFERENCE DATA command replaces the value of the reference PIN by a new value.

The current value of the reference PIN must be presented as part of the CHANGE REFERENCE DATA command.

If the CHANGE REFERENCE DATA command is successful, the following actions take place:

- The reference PIN Try Counter is set to PIN Try Limit.
- The reference PIN validated flag is set to false (because the new value has not yet been verified).
- The reference PIN value is updated with the value sent in CHANGE REFERENCE DATA command.
- The reference PIN's "PIN changed" flag is set to "true".
- The reference PIN usage counter is unchanged.

.

If the CHANGE REFERENCE DATA command fails, the following actions take place:

- The reference PIN Try Counter is decremented by 1.
- The reference PIN validated flag is set to false.
- The reference PIN usage counter is unchanged.

7.11.1 Conditions of Use and Security

7.11.1.1 Usage

7.11.1.2 Security

The PIN Try Counter and the Usage Counter for both the reference PIN and the change PIN must not be zero.

Note: For clarity, the command is described with the data in plain text.

7.11.2 Format

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging
INS	2Ch
P1	00h - reset retry counter and set new verification data
P2	Reference of the PIN: - 11h for global PIN - 81h–8Fh for local PIN
Lc	Length of subsequent data field
Data	Resetting code (padded to the correct length: 8–16 bytes (local PIN), 8-12 bytes (global PIN)) followed by new reference data (padded to the correct length 8–16 bytes (local PIN), 8-12 bytes (global PIN)). Padding is done according to ISO/IEC 7816-15.
Le	Empty

Table 44. CHANGE REFERENCE DATA command APDU

7.11.3 Response

Byte	Value
Data	Empty
SW1-SW2	Status bytes

Table 45. CHANGE REFERENCE DATA response APDU

Possible values for the SW1, SW2 status codes are as follows:

SW1	SW2	Description
90h	00h	Command processed without error
63h	Cxh	Reference PIN not verified.”x” attempts remaining for change PIN
67h	00h	Incorrect length for example: • Lc incorrect • Length of new PIN value is not the same as length of existing value
69h	82h	Security conditions not satisfied (error during secure messaging or security attribute of reference PIN for Change Reference Data is never)
69h	83h	Authentication method blocked (Change PIN blocked)
69h	84h	Referenced data invalidated (PIN Try Counter or PIN Usage Counter for either the reference PIN or the change PIN has reached zero)
69h	85h	Conditions of use not satisfied, for example: • Security attribute for the current contact/contactless interface is absent, zero or not BER-TLV coded.
6Ah	86h	Incorrect P1, P2
6Ah	88h	Referenced data not found

Table 46. CHANGE REFERENCE DATA, possible SW1 and SW2 status codes

7.12 RESET RETRY COUNTER

The RESET RETRY COUNTER command is used when a PIN code has been locked due to too many consecutive unsuccessful verifications. Unlocking a PIN requires a resetting code (a.k.a. PIN Unlocking Key, PUK) to be presented to the card by the user.

If the RESET RETRY COUNTER command is successfully executed, the following actions take place:

- The reference PIN Try Counter is set to PIN Try Limit.
- The reference PIN validated flag is set to false (because the PIN is not yet verified)
- The reference PIN unblocking counter is decremented by 1, unless its value is A5h (no limit to the number of times reference PIN can be unblocked) or 00h (already been unblocked maximum allowed number of times).
- The reference PIN usage counter is unchanged.
- The PUK Try Counter is set to unblock PUK Try Limit
- The PUK validated flag is set to false
- The PUK usage counter is decremented by 1 unless its value is FFh (no limit to the number of times unblock PIN can be used) or 00h (already been used maximum allowed number of times).
- If a new value for the reference PIN is specified, the reference PIN value is updated with the value sent in the RESET RETRY COUNTER command
- If a new value for the reference PIN is specified, the reference PIN's "PIN changed" flag is set to "true"

If the RESET RETRY COUNTER command fails, the following actions take place:

- The PUK Try Counter is decremented by 1
- The PUK validated flag is set to false

7.12.1 Conditions of Use and Security

7.12.1.1 Usage

When used to unblock a PIN, a new value can be given to the reference PIN.

7.12.1.2 Security

None of the following counters must be zero:

- The Try Counter of the PUK
- The Unblocking Counter of the Reference PIN
- The Usage Counters for both the reference PIN and the PUK.

Note: For clarity, the command is described with the data in plain text.

7.12.2 Format

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging
INS	0Ch
P1	00h - reset retry counter and set new verification data 01h – only reset retry counter

P2	Reference of the PIN: - 11h for global PIN - 81h–8Fh for local PIN
Lc	Empty or length of subsequent data field
Data	Empty, or resetting code (padded to the correct length) followed by new reference data (padded to the correct length), or resetting code (padded to the correct length Padding is done according to ISO/IEC 7816-15.
Le	Empty

Table 47. RESET RETRY COUNTER command APDU

7.12.3 Response

Byte	Value
Data	Empty
SW1-SW2	Status bytes If Lc = 00h, status bytes indicate the number X of further allowed retries (SW1-SW2 = 63CXh).

Table 48. RESET RETRY COUNTER response APDU

Possible values for the SW1, SW2 status codes are as follows:

SW1	SW2	Description
90h	00h	Command processed without error
63h	Cxh	PUK not verified.”x” attempts remaining for PUK
67h	00h	Incorrect length, Lc.
69h	82h	Security conditions not satisfied (error during secure messaging or security attribute for the reference PIN is never)
69h	83h	Authentication method blocked (the FP_PUK needed to unblock the reference PIN is blocked itself)
69h	84h	Referenced data invalidated (PIN Unblocking Counter for reference PIN or PIN Try Counter or PIN Usage Counter for FP_PUK has reached zero)
69h	85h	Conditions of use not satisfied, for example: • Security attribute for the current contact/contactless interface is absent, zero or not BER-TLV coded.
6Ah	86h	Incorrect P1, P2
6Ah	88h	Referenced data not found

Table 49. RESET RETRY COUNTER, possible SW1 and SW2 status codes

1.1 GET DATA

The GET DATA command is used to retrieve following BER-TLV objects:

- Public key elements
- PIN information

The data in the response is in TLV format.

7.12.4 Conditions of Use and Security

1.1.1.1 Usage

When retrieving ELC public key elements, the elements must be retrieved individually, that is, only one element per GET DATA command. For RSA public key elements, the elements can be retrieved individually or the command can retrieve ALL the elements in the one command.

1.1.1.2 Security

The use of the GET DATA command depends on the security attributes of the object whose value is to be retrieved.

Note: For clarity, the command is described here with the data in plain text.

7.12.5 Format

Byte	Value
CLA	00h - no secure messaging 0Ch - secure messaging
INS	CBh
P1	00h
P2	FFh

Lc	Length of subsequent data field
Data	See tables below.
Le	Number of bytes expected in response

Table 50. GET DATA command APDU

Tag	Length	Value			Description	No. of Bytes
B6h	03h				Tag and length of DST Template	2
		83h	01h	KeyID	TLV of public key	3
7Fh 49h	02h				Tag and length of public key template	3
		Tag	Length			
		81h	00h		Tag and length of modulus	2
					TOTAL BYTES	**10**

Table 51. Get Data Coding Structure – Public Key Element (the modulus in this example)

Note: The length of 00h means that all the element is recovered.

Tag	Description
81h	p: prime modulus according to curve type
82h	a: 1st coefficient of curve
83h	b: 2nd coefficient of curve
84h	G: coordinates X and Y in F; defining a curve point G of order n. DO is formatted as follows (uncompressed format): 04h || XG || YG
85h	n: order of the base point (positive prime integer)
86h	Q - coordinates X and Y in F defining the public key point Q in E. DO is formatted as follows (uncompressed format): 04h || XQ || YQ
87h	h: cofactor

Table 52. Tags for ELC public key elements

Tag	Length	Value			Description	No. of Bytes
B6h	03h				Tag and length of DST Template	2
		83h	01h	KeyID	TLV of public key	3
7Fh 49h					Tag of public key template	2
		Tag				
		80h			Tag of entire public key	1
					TOTAL BYTES	**8**

Table 53. Get Data Coding Structure – entire RSA public key (modulus and exponent elements)

Caution: Remember that this would not be possible for ELC public keys.

Tag	Length	Value			Description	No. of Bytes
A0h	03h	Tag	Length	Value	Tag and length of FP_PIN Template	2
		83h	01h	xxh	xxh is FP_PIN Reference as follows: 81h-8Fh for local PIN 11h for global PIN	3

Table 54. Get Data Coding Structure – PIN Container

The "PIN changed" flag is returned only if the "Return PIN changed flag" parameter is set.

Caution: This is not ISO-compliant since the tag A0h means constructed tag, which this isn't. However, if you require your application to be ISO-compliant, you can safely add additional tags to the A0h template as this will not affect the response data (all the data for the PIN is returned).

7.12.6 Response

Byte	Value
Data	Value of retrieved Data Object. See examples below.
SW1-SW2	Status bytes

Table 55. GET DATA response APDU

Tag	Length	Value			Description	No. of Bytes
B6h	03h				Tag and length of DST Template	2
		83h	01h	KeyID	TLV of public key	3
7Fh 49h	8183h				Tag and length of public key template	4
		Tag	Length	Value		
		81h	8180h	ValMod	TLV of modulus (value coded on 128 bytes)	131
					TOTAL BYTES	**140**

Table 56. Get Data Response Structure – Public Key Element (the modulus in this example)

Tag	Length	Value			Description	No. of Bytes
B6h	03h				Tag and length of DST Template	2
		83h	01h	KeyID	TLV of public key	3
7Fh 49h	81 8Dh				Tag and length of public key template	4
		Tag	Length	Value		
		81h	81h 80h	ValMod	TLV of modulus (value coded on 128 bytes)	131
		82h	LEXP	ValEXP	TLV of exponent (value coded on 8 bytes)	10
					TOTAL BYTES	**150**

Table 57. Get Data Response Structure – entire RSA public key (modulus and exponent elements)

Tag	Length	Value			Description	No. of Bytes
A0h	LA0	Tag	Length	Value	Tag and length of PIN Template	2
		83h	01h	xxh	xxh is PIN Reference byte (81h-8Fh for local PIN, 11h for global PIN)	3
		8Ch	L8C		Security Attributes - contact interface	Var
		9Ch	L8C		Security Attributes - contactless interface	Var
		DFh 21h	04h		PIN attributes	7
		DFh 27h	02h		PIN credentials counter	5
		DFh 28h	01h		PIN length	4
		DFh 2Fh	01h		PIN changed flag (1 means PIN has been changed)	3
					TOTAL BYTES	**Var**

Table 58. Get Data Response Structure – PIN Information

If the PIN being returned is the global PIN, and the global PIN has been initialized outside eID application, the PIN length value is returned as 00h. If however, the global PIN has been initialized and is managed by the eID application, then the PIN length will be accurate.

The content of PIN attributes are

Field	Length	Description
#1	1 byte	PIN Try Counter. Stores the number of remaining attempts that can be made to present the correct PIN before it is blocked.
#2	1 byte	Usage counter (range 01h – FFh; FFh means no limit).
#3	1 byte	Unblocking counter (range 00h – 0Fh or A5h for no limit)
#4	1 byte	Unblocking PIN method. See table below.

Meaning	b8	b7	b6	b5	b4	b3	b2	b1
Unblock PIN not allowed (neither by unblock PIN nor by MS 3DES3 External Authentication	0	0	0	0	0	0	0	0
MS 3DES3 Ext Auth grants unblock PIN		1						
MS 3DES3 Ext Auth does not grant unblock PIN		0						
FP_PUK Ref to be used	X		X	X	X	X	X	X
Not checked	1	1	1	1	1	1	1	1

Table 59. Unblocking PIN method byte coding

The range of values for the FP_PUK reference are 01h – 0Fh for a local PIN, 11h for the global PIN.

The unblocking PIN method byte is coded as an OR condition between the MS 3DES3 External Authentication and the FP_PUK reference. Here are some example values:

- 84h: Unblocking possible with PIN#4 (PIN ref is 84h)
- 40h: Unblocking possible only if MS 3DES3 Ex Auth has taken place.
- C4h: Unblocking possible with PIN#4 or if MS 3DES3 Ex Auth has taken place.
- 00h: NEVER

Note: b7 is taken into account EXCEPT when the byte is FFh.

Credentials Counter

Certain objects may be protected by security attributes that specify that user authentication (PIN) is necessary before access is granted to a particular command. The security attribute therefore checks the PIN status when the command is issued.

The aim of the credentials counter is to limit the number of times that such checks are made, either for the current session or for the life of the applet. The counter comes into operation after a successful PIN verification.

The counter can exist for both local PINs and the global PIN.

Caution: In the case of global PINs, the counter is internal to the eID application. If used by the eID application, the counter is correctly reset in RAM after a successful global PIN presentation. However it is NOT reset if the global PIN was successfully verified by another applet. Consequently, the mechanism is not triggered in such a case and so access requiring the global ¨PIN is not granted, even though the global PIN has been successfully verified.

The first byte of the counter is the counter type and can be "RAM" (00h) or "EEPROM" (80h). The two types of counter work differently.

RAM Type

The counter limits the number of times that the PIN status is checked for the current session. A session starts for each successful Verify on a given PIN, and ends when the credential counter reaches 0. It works as follows:

1. When the PIN is successfully verified, the "current credential counter value" (stored in RAM) is reset to the credentials counter value.
2. Each time a security attribute or any action checks the PIN status (such as an access condition or a PIN used as an FP_PUK), the "current credential counter" decrements by 1. The credentials counter value in the PIN is NOT decremented.
3. If the current credential counter reaches zero, operations that test if this PIN access is granted will fail.

EEPROM Type

The counter limits the number of times that the PIN status is checked for the life of the applet. It works as follows:

1. When the PIN is successfully verified, the counter comes into operation.
2. Each time a security attribute or any action checks the PIN status (such as an access condition or a PIN used as an FP_PUK), the credentials counter value in the PIN decrements by 1.
3. If the credentials counter value reaches zero, it remains as zero, meaning that the PIN can no longer be used for access rights. However, the PIN can still be used with the CHANGE REFERENCE DATA and RESET RETRY COUNTER commands.

The Credentials Counter is a two-byte value coded as follows:

Counter Type	00h	RAM
	80h	EEPROM
Counter Value	00h to FEh	• If RAM type, the current credentials counter in RAM is set to this value when the PIN is verified. It is the RAM value that is decremented when the PIN status is checked – NOT the Counter Value in the PIN. • If EEPROM type, this value is decremented each time the PIN status is checked
	FFh	There is no limit to the number of PIN status checks. This is true for both types of Credentials Counter.

Table 60. Credentials Counter Coding

The "PIN changed" flag is returned only if the "Return PIN changed flag" parameter was set to true when the applet was personalized.

Possible values for the SW1, SW2 status codes are as follows:

SW1	SW2	Description
90h	00h	Command processed without error
69h	82h	Security status not satisfied, error during secure messaging
69h	85h	Conditions of use not satisfied, for example: • Security attribute for the current contact/contactless interface is absent, zero or not BER-TLV coded.
6Ah	80h	Incorrect FP_PIN reference or incorrect data field when retrieving PIN information
6Ah	86h	Incorrect P1, P2.

Table 61. GET DATA, possible SW1 and SW2 status codes

8 Implementation guidelines for software developer

8.1 Application/File selection

8.1.1 CIA application

CIA (Cryptographic Information Application) application is selected using following Application Identifier (AID):

A0 00 00 00 63 50 4B 43 53 2D 31 35

Selection by Application identifier:

Command	CLA	INS	P1	P2	Lc	Data	Le
SELECT	00	A4	04	00	0C	A0 00 00 00 63 50 4B 43 53 2D 31 35	-

8.2 Path

CIA uses Path ASN.1 structure to reference various files. The Path.efidOrPath octet string contains:

- a file identifier if the length of the octet string is two bytes
- an absolute path if the octet string is longer that two bytes and starts with the file identifier of MF = 3F 00
- a relative path if the octet string is longer than two bytes and starts with the file identifier of the DF (which is not 3F 00)

8.3 Authentication objects

In CIA all objects (private keys, certificates etc.) can be protected with authentication objects (i.e. PINs). Each object may contain a pointer to an authentication object e.g. a private key object may contain a pointer to a PIN object. This means that the private key operation (decrypt or sign) can be done only after successful verification of the PIN code.

The verification status of a PIN may be dropped automatically to state 'not verified' by the card operating system after performing e.g. a private key operation (Compute Digital Signature operation). This is indicated by the **userConsent** element of the private key object. E.g. **userConsent** value set to one for a private key object indicates that the card holder must manually enter the PIN for each Compute Digital Signature operation.

8.3.1 Verify PIN

Padding is done according to PassWordAttributes (storedLength, padChar). The P2 value is taken from PassWordAttributes.pwdReference.

Verify PIN1:

Command	CLA	INS	P1	P2	Lc	Data	Le
VERIFY	00	20	00	81	0C	31 32 33 34 00 00 00 00 00 00 00 00 (PIN = 1234 in ASCII with 00 padding)	-

Verify PIN2:

Command	CLA	INS	P1	P2	Lc	Data	Le
VERIFY	00	20	00	82	0C	31 32 33 34 35 00 00 00 00 00 00 00 (PIN = 12345 in ASCII with 00 padding)	-

8.3.2 Change PIN

Change PIN1:

Command	CLA	INS	P1	P2	Lc	Data	Le
CHANGE REFERENCE DATA	00	24	00	81	18	31 32 33 34 00 00 00 00 00 00 00 00 35 36 37 38 00 00 00 00 00 00 00 00 (old PIN = 1234 in ASCII with 00 padding | new PIN = 5678 in ASCII with 00 padding)	-

Change PIN2:

Command	CLA	INS	P1	P2	Lc	Data	Le
CHANGE REFERENCE DATA	00	24	00	82	18	31 32 33 34 35 00 00 00 00 00 00 00 34 35 36 37 38 00 00 00 00 00 00 00 (old PIN = 12345 in ASCII with 00 padding | new PIN = 45678 in ASCII with 00 padding)	-

8.3.3 Unblocking blocked PIN

Unblock PIN1:

Command	CLA	INS	P1	P2	Lc	Data
RESET RETRY COUNTER	00	2C	00	81	18	31 32 33 34 35 36 37 38 00 00 00 00 35 36 37 38 00 00 00 00 00 00 00 00 (PUK = 12345678 in ASCII with 00 padding | new PIN = 5678 in ASCII with 00 padding)

Unblock PIN2:

Command	CLA	INS	P1	P2	Lc	Data
RESET RETRY COUNTER	00	2C	00	82	18	31 32 33 34 35 36 37 38 00 00 00 00 35 36 37 38 31 00 00 00 00 00 00 00 (PUK = 12345678 in ASCII with 00 padding | new PIN = 56781 in ASCII with 00 padding)

8.4 Private key operations

There may be multiple private keys in the same CIA application. The host application must first determine which one of these private keys to use. This can be done e.g. based on the information inside card holder certificates according to application specific criteria (e.g. key usage bits and CA policy OIDs). Each certificate contains a pointer to the corresponding private key object.

8.4.1 Signature operation

It is assumed that corresponding PIN verification has already been done.

Set the following properties into the SE Digital Signature Template:

- algorithm reference (= 54 i.e. SHA384-ECDSA, card does padding and DigestInfo encapsulating of the hash)
- key reference (= 01 derived from PrivateKeyAttributes.keyReference)

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE: SET	00	22	41	B6 DST in data field	06	80 01 54 (algorithm reference = 54) 84 01 01 (private key reference)	-

The hash is performed entirely by the card, only one block to hash:

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO: HASH	00	2A	90	A0	16	80 14 4B 52 16 5B 4A B6 54 C3 E5 4F 64 B5 F1 EE A6 45 D4 6B 65 C8	-

Sign the hash calculated by the card:

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO: COMPUTE DIGITAL SIGNATURE	00	2A	9E	9A	-	-	00

8.5 Certificate objects

The eID contains two certificates; one for card holder authentication and key agreement operations and the second for card holder digital signature operations. The certificates have been stored on different eID application subdirectories (as defined in the EF.CDF file).

- authentication certificate: file with FID = 3411 in DF.AWP subdirectory (FID = ADF1)
- signature certificate: file with FID = 3421 in DF.QSCD subdirectory (FID = ADF2)

Certificate files are transparent binary files and can be read with the READ BINARY command.

Below is shown procedure to read certificates after eID application selection:

1. Select the certificate file using SELECT FILE command:
   - authentication certificate:

Command	CLA	INS	P1	P2	Lc	Data	Le
SELECT FILE	00	A4	08	0C	0C	AD F1 34 11	-

   - signature certificate:

Command	CLA	INS	P1	P2	Lc	Data	Le
SELECT FILE	00	A4	08	0C	0C	AD F2 34 21	-

2. Send READ BINARY commands after the end of the file has been reached (card returns SW bytes=6B00 (Wrong parameter P1-P2) indicating that our offset is outside file size). The offset in the next READ BINARY commands (P1-P2 parameters) shall be changed according to amount of data read in previous READ BINARY commands.
   - First command:

Command	CLA	INS	P1	P2	Lc	Data	Le
READ BINARY	00	B0	00	00	-	-	00

   - First response:

Data	SW1	SW2
First 256 bytes of certificate	90	00

- In next commands the offset shall be increased according to amount of data read in previous ones. in this case the offset of the next commands shall be increased by value 256 until card returns SW bytes=6B00, indicating that end of file has been reached. Below is shown the second READ BINARY command as an example:

Command	CLA	INS	P1	P2	Lc	Data	Le
READ BINARY	00	B0	01	00	-	-	00

Annex A (Informative): example APDU trace for main functionalities

Below is shown APDU trace for following main functionalities:

1. Select eID application
2. Read authentication certificate.
3. Verify PIN1.
4. Do signing operation using authentication key.

```
B0 C: 00 A4 04 04 - SELECT Lc=12
      0005  A0 00 00 00 63 50 4B 43 53 2D 31 35              ....cPKCS-15
      Le=0
   R: SW1/SW2=9000 (Normal processing: No error) Lr=0
B0 C: 00 A4 08 00 - SELECT Lc=4
      0005  AD F1 34 11                                     ..4.
      Le=0
   R: SW1/SW2=9000 (Normal processing: No error) Lr=22
      0000  6F 14 81 02 04 0B 82 01 01 83 02 34 11 8A 01 05  o..........4....
      0010  8C 04 43 F1 F1 00                               ..C...
B0 C: 00 B0 00 00 - READ BINARY Lc=1
      0005  00                                              .
   R: SW1/SW2=9000 (Normal processing: No error) Lr=256
      0000  30 82 04 07 30 82 03 8D A0 03 02 01 02 02 14 26  0...0..........&
      0010  01 32 59 7F 7F CA 10 75 AB B1 DD 87 A6 FE 72 7F  .2Y....u......r.
      0020  D1 F2 96 30 0A 06 08 2A 86 48 CE 3D 04 03 03 30  ...0...*.H.=...0
      0030  5C 31 18 30 16 06 03 55 04 03 0C 0F 54 65 73 74  \1.0...U....Test
      0040  20 45 53 54 45 49 44 32 30 32 35 31 17 30 15 06   ESTEID20251.0..
      0050  03 55 04 61 0C 0E 4E 54 52 45 45 2D 31 37 30 36  .U.a..NTREE-1706
      0060  36 30 34 39 31 1A 30 18 06 03 55 04 0A 0C 11 5A  60491.0...U....Z
      0070  65 74 65 73 20 45 73 74 6F 6E 69 61 20 4F C3 9C  etes Estonia O..
      0080  31 0B 30 09 06 03 55 04 06 13 02 45 45 30 1E 17  1.0...U....EE0..
      0090  0D 32 34 31 32 31 37 31 32 33 33 33 32 5A 17 0D  .2412171233332Z..
      00A0  32 39 31 32 30 39 32 30 35 39 33 32 5A 30 7D 31  291209205932Z0}1
      00B0  29 30 27 06 03 55 04 03 0C 20 45 52 D2 AA 45 54  )0'..U... ER..ET
      00C0  49 4E 2C 4D C3 82 4E 49 20 C3 96 5A 45 52 2C 33  IN,M..NI ..ZER,3
      00D0  36 39 30 38 32 30 39 39 39 33 31 1A 30 18 06 03  69082099931.0...
      00E0  55 04 05 13 11 50 4E 4F 45 45 2D 33 36 39 30 38  U....PNOEE-36908
      00F0  32 30 39 39 39 33 31 14 30 12 06 03 55 04 2A 0C  2099931.0...U.*.
B0 C: 00 B0 01 00 - READ BINARY Lc=1
      0005  00                                              .
   R: SW1/SW2=9000 (Normal processing: No error) Lr=256
      0000  0B 4D C3 82 4E 49 20 C3 96 5A 45 52 31 11 30 0F  .M..NI ..ZER1.0.
      0010  06 03 55 04 04 0C 08 45 52 D2 AA 45 54 49 4E 31  ..U....ER..ETIN1
      0020  0B 30 09 06 03 55 04 06 13 02 45 45 30 76 30 10  .0...U....EE0v0.
```

```
      0030  06 07 2A 86 48 CE 3D 02 01 06 05 2B 81 04 00 22  ..*.H.=....+..."
      0040  03 62 00 04 13 76 FB 18 2B 1C 49 8C 07 6C 6C 35  .b...v..+.I..ll5
      0050  F6 0B 83 84 6E 3D D9 1C BE 65 02 55 05 FD 4E 80  ....n=...e.U..N.
      0060  6D 43 6E 2D CF A7 8F F4 B2 BD 9B 45 56 EE 9A F9  mCn-.......EV...
      0070  20 39 B5 14 40 8A DB 84 DF DD A0 3E 8E 43 70 FB   9..@......>.Cp.
      0080  89 6F A1 08 C6 13 E4 48 04 CA 09 C3 C2 94 F4 94  .o.....H........
      0090  0A 1F 0A B0 36 A0 E4 04 96 E2 75 18 3E D1 AA 21  ....6.....u.>..!
      00A0  FE 43 3F 05 A3 82 01 ED 30 82 01 E9 30 09 06 03  .C?.....0...0...
      00B0  55 1D 13 04 02 30 00 30 1F 06 03 55 1D 23 04 18  U....0.0...U.#..
      00C0  30 16 80 14 EE F2 95 3F 8C B2 FC 51 9E 84 E6 E6  0......?...Q....
      00D0  5E 84 11 7E 42 BA 20 36 30 70 06 08 2B 06 01 05  ^..~B. 60p..+...
      00E0  05 07 01 01 04 64 30 62 30 38 06 08 2B 06 01 05  .....d0b08..+...
      00F0  05 07 30 02 86 2C 68 74 74 70 3A 2F 2F 63 72 74  ..0..,http://crt
B0 C: 00 B0 02 00 - READ BINARY Lc=1
      0005  00                                              .
   R: SW1/SW2=9000 (Normal processing: No error) Lr=256
      0000  2D 74 65 73 74 2E 65 69 64 70 6B 69 2E 65 65 2F  -test.eidpki.ee/
      0010  74 65 73 74 45 53 54 45 49 44 32 30 32 35 2E 63  testESTEID2025.c
      0020  72 74 30 26 06 08 2B 06 01 05 05 07 30 01 86 1A  rt0&..+.....0...
      0030  68 74 74 70 3A 2F 2F 6F 63 73 70 2D 74 65 73 74  http://ocsp-test
      0040  2E 65 69 64 70 6B 69 2E 65 65 30 1F 06 03 55 1D  .eidpki.ee0...U.
      0050  11 04 18 30 16 81 14 33 36 39 30 38 32 30 39 39  ...0...369082099
      0060  39 33 40 65 65 73 74 69 2E 65 65 30 56 06 03 55  93@eesti.ee0V..U
      0070  1D 20 04 4F 30 4D 30 08 06 06 04 00 8F 7A 01 02  . .O0M0......z..
      0080  30 41 06 0E 88 37 01 03 06 01 04 01 83 91 21 02  0A...7........!.
      0090  01 06 30 2F 30 2D 06 08 2B 06 01 05 05 07 02 01  ..0/0-..+.......
      00A0  16 21 68 74 74 70 73 3A 2F 2F 72 65 70 6F 73 69  .!https://reposi
      00B0  74 6F 72 79 2D 74 65 73 74 2E 65 69 64 70 6B 69  tory-test.eidpki
      00C0  2E 65 65 30 1D 06 03 55 1D 25 04 16 30 14 06 08  .ee0...U.%..0...
      00D0  2B 06 01 05 05 07 03 02 06 08 2B 06 01 05 05 07  +.........+.....
      00E0  03 04 30 43 06 08 2B 06 01 05 05 07 01 03 04 37  ..0C..+........7
      00F0  30 35 30 33 06 06 04 00 8E 46 01 05 30 29 30 27  0503.....F..0)0'
B0 C: 00 B0 03 00 - READ BINARY Lc=1
      0005  00                                              .
   R: SW1/SW2=9000 (Normal processing: No error) Lr=256
      0000  16 21 68 74 74 70 73 3A 2F 2F 72 65 70 6F 73 69  .!https://reposi
      0010  74 6F 72 79 2D 74 65 73 74 2E 65 69 64 70 6B 69  tory-test.eidpki
      0020  2E 65 65 13 02 65 6E 30 3D 06 03 55 1D 1F 04 36  .ee..en0=..U...6
      0030  30 34 30 32 A0 30 A0 2E 86 2C 68 74 74 70 3A 2F  0402.0...,http:/
      0040  2F 63 72 6C 2D 74 65 73 74 2E 65 69 64 70 6B 69  /crl-test.eidpki
      0050  2E 65 65 2F 74 65 73 74 45 53 54 45 49 44 32 30  .ee/testESTEID20
      0060  32 35 2E 63 72 6C 30 1D 06 03 55 1D 0E 04 16 04  25.crl0...U.....
      0070  14 2A C2 72 64 60 CB 40 8B 3E 66 FC CA 65 BF 23  .*.rd`.@.>f..e.#
      0080  71 5D CD 50 77 30 0E 06 03 55 1D 0F 01 01 FF 04  q].Pw0...U......
```

```
      0090  04 03 02 03 88 30 0A 06 08 2A 86 48 CE 3D 04 03  .....0...*.H.=..
      00A0  03 03 68 00 30 65 02 30 41 5C 30 15 BC 0B 48 09  ..h.0e.0A\0...H.
      00B0  F8 0A F8 B8 D7 A4 EF 33 D8 E5 2B 8B 33 53 9F 2F  .......3..+.3S./
      00C0  54 5E 73 8C 6B D2 BF 22 42 86 50 9A 75 04 B8 CB  T^s.k.."B.P.u...
      00D0  A2 3A 12 01 8D C8 FF E2 02 31 00 C4 87 73 62 4D  .:.......1...sbM
      00E0  78 D4 91 5B 36 7D F2 6B 1F EF F7 D6 EF E0 7C C1  x..[6}.k......|.
      00F0  BB F5 99 BD 85 BB 97 10 F0 B2 07 92 59 DD 2F 15  ............Y./.
B0 C: 00 B0 04 00 - READ BINARY Lc=1
      0005  00                                              .
   R: SW1/SW2=9000 (Normal processing: No error) Lr=11
      0000  24 08 FF 8D BF CE B0 13 F6 9F 02                 $..........
B0 C: 00 B0 04 0B - READ BINARY Lc=1
      0005  00                                              .
   R: SW1/SW2=6B00 (Checking error: Wrong parameter P1-P2) Lr=0
B0 C: 00 20 00 81 - VERIFY Lc=12
      0005  31 32 33 34 00 00 00 00 00 00 00 00              1234........
      Le=0
   R: SW1/SW2=9000 (Normal processing: No error) Lr=0
B0 C: 00 22 41 B6 - MANAGE SECURITY ENVIRONMENT Lc=6
      0005  80 01 54 84 01 01                                ..T...
      Le=0
   R: SW1/SW2=9000 (Normal processing: No error) Lr=0
B0 C: 00 2A 90 A0 - PSO: HASH Lc=50
      0005  90 30 99 51 43 29 18 6B 2F 6A E4 A1 32 9E 7E E6  .0.QC).k/j..2.~.
      0015  C6 10 A7 29 63 63 35 17 4A C6 B7 40 F9 02 83 96  ...)cc5.J..@....
      0025  FC C8 03 D0 E9 38 63 A7 C3 D9 0F 86 BE EE 78 2F  .....8c.......x/
      0035  4F 3F                                           O?
      Le=0
   R: SW1/SW2=9000 (Normal processing: No error) Lr=48
      0000  99 51 43 29 18 6B 2F 6A E4 A1 32 9E 7E E6 C6 10  .QC).k/j..2.~...
      0010  A7 29 63 63 35 17 4A C6 B7 40 F9 02 83 96 FC C8  .)cc5.J..@......
      0020  03 D0 E9 38 63 A7 C3 D9 0F 86 BE EE 78 2F 4F 3F  ...8c.......x/O?
B0 C: 00 2A 9E 9A - PSO: COMPUTE DIGITAL SIGNATURE      Le=0
   R: SW1/SW2=9000 (Normal processing: No error) Lr=96
      0000  9F FD 8C 32 08 C4 FE 32 1E FE F6 14 39 C7 E3 BA  ...2...2....9...
      0010  2B B0 B1 86 A3 E0 58 1F 55 4C EE 3B 04 68 46 4D  +.....X.UL.;.hFM
      0020  20 D1 7D 35 6C F5 C8 D5 3C 59 FC 3E E1 29 F4 45   .}5l...<Y.>.).E
      0030  E2 9B 4E 3E C4 FD 6C 18 98 B4 5D 2A 45 69 4C 01  ..N>..l...]*EiL.
      0040  E3 33 47 F1 BA B7 52 94 9A 85 82 7D D4 7F 8F B2  .3G...R....}....
      0050  25 B8 17 90 9D CF 3F 74 0F AF 9F 6A 8A 8F 70 C3  %.....?t...j..p.
```

Annex B (Informative): coding of the File Control Parameters and File Control Information templates

The FCI template returned by SELECT and SELECT FILE commands is a TLV (Tag-Length-Value) coded data structure.

FCI for DFs

Offset	Data	Description
0	6Fh	Tag of FCI template
1	L	Length of FCI data
2	83h	Tag of File ID
3	02h	Length of File ID
4–5	File ID	Value of File ID
6	8Ch	Tag of security attributes
7	L	Length of security attributes
8	AMB	Access mode byte
9–(8+X)	SCB	Security condition bytes (X)
9+X	84h	Tag of DF Name
10+X	L	Length of DF Name
11+X –	DF name	Value of DF name (up to 16 bytes)

Note: In the example above, the contact interface security attribute is given (tag 8Ch). This could alternatively be replaced by the contactless interface security attribute, (tag 9Ch) or both interfaces could be present.

FCI for EFs

Offset	Data	Description
0	6Fh	Tag of FCI template
1	L	Length of FCI data
2	81h	Tag of File Size
3	02h	Length of File Size
4–5	File Size	Value of File Size.
6	82h	Tag of FDB
7	01h	Length of FDB
8	FDB	Value of FDB
9	83h	Tag of File ID
10	02h	Length of File ID
11–12	File ID	Value of File ID
13	8Ah	Tag of Life Cycle Status byte for file
14	01h	Length of Life Cycle Status byte for file
15	Var.	Value of Life Cycle Status byte for file
16	8Ch	Tag of security attributes
17	L	Length of security attributes
18	AMB	Access mode byte
19–(18+X)	SCBs	Security condition bytes (X)

Note: In the example above, the contact interface security attribute is given (tag 8Ch). This could alternatively be replaced by the contactless interface security attribute, (tag 9Ch) or both interfaces could be present.

FCP for DFs

Tag	Length	Tag	Length	Value	Description
62h	0Bh–25h				Tag and length of FCP template
		82h	01h	38h	TLV of FDB
		83h	02h	Var.	TLV of File ID
		8Ch	02h–04h	Var.	TLV of Security attributes (access mode and security condition bytes) - contact interface
		9Ch	02h–04h	Var.	TLV of Security attributes (access mode and security condition bytes) - contactless interface
		84h	01-10h	Var.	TLV of DF name

The DF name is optional and if used must be the final parameter. The FDB and file ID are mandatory.

Only one security attribute is mandatory (it is possible to have 8Ch or 9Ch or both).

The structure of the FCP of a dedicated file is as follows:

Field	Length	Description
FDB	1 byte	The file descriptor byte (FDB) is set to 38h when a DF is created.
File ID	2 bytes	This is allocated when the file is created.
Security Attribute	2–4 bytes	The security attributes are made up of one Access Mode byte which indicates the commands to be "controlled", followed by 1–3 security condition bytes which indicate the conditions for each command indicated in the Access Mode byte. The security attributes must be present for at least one of the interfaces (contact or contactless). Both interfaces can be present if required. The coding is identical for the contact and contactless interfaces.
DF Name	1–16 bytes	This contains the name of the DF up to a maximum of 16 bytes. It can be used to reference a file in a command

FCP for EFs

Tag	Length	Tag	Length	Value	Description

62h	12h–1Eh				Tag and length of FCP template
		81h	02h	Var.	TLV of File size (in bytes)
		82h	01h	01h	TLV of FDB
		83h	02h	Var.	TLV of File ID
		8Ah	01h	Var.	TLV of Life cycle status
		8Ch	02h–06h	Var.	TLV of Security attributes (access mode and security condition bytes) - contact interface
		9Ch	02h–06h	Var.	TLV of Security attributes - contactless interface

All these elements are mandatory. However only one security attribute is mandatory (it is possible to have 8Ch or 9Ch or both).

The structure of the FCP of an elementary file is as follows:

Field	**Length**	**Description**
File Size	2 bytes	This specifies the size of the file.
FDB	1 byte	The file descriptor byte (FDB) is set to 01h, meaning transparent file, when an EF is created.
Identifier (FID)	2 bytes	This is allocated when the file is created. The short file identifier corresponds to the 5 least significant bits of the file identifier. It is used to reference a file in a command.
Life Cycle Status	1 byte	The life cycle status shows the status of the EF. The LCS byte is coded as described in the table below.
Security Attribute	2-6 bytes	The security attributes are made up of one Access Mode byte which indicates the commands to be “controlled”, followed by 1–5 security condition bytes which indicate the conditions for each command indicated in the Access Mode byte. The security attributes must be present for at least one of the interfaces (contact or contactless). Both interfaces can be present if required. The coding is identical for the contact and contactless interfaces.

Table 62. Life Cycle Status byte coding

b8…b5	b4	b3	b2	b1	State
0..0	0	0	0	1	CREATED
0..0	0	0	1	1	INITIALIZED
0..0	0	1	–	1	OPERATIONAL (ACTIVATED)
0..0	0	1	–	0	OPERATIONAL (DEACTIVATED)

EF Types

All EFs managed by eID application are transparent EFs. These have an FDB value of 01h. A transparent file consists of an unstructured sequence of bytes that can be accessed by specifying an offset relative to the start of the EF. The offset size is given in bytes. The first byte of a transparent EF has the relative address 00h.

Security Attributes

Files and data objects are protected by security attributes, that must be fulfilled before access is granted to the file or data object. These security attributes are defined at the time of file creation.

Each security attribute is made up of:

- One access mode byte (AMB), that defines the command(s) to be protected against
- One security condition byte (SCB) for each bit set to 1 in the AMB

Each SCB defines the conditions that must be fulfilled in order to allow the corresponding command to be performed.

Note:

1 Security attributes are checked by commands only after personalization.

2 Security Environments do not have security attributes because they cannot be modified.

The security attributes are coded in TLV format.

Contact Vs. Contactless interface

Each file or data object can have one security attribute for the contact interface (tag 8Ch) and/or one security attribute for the contactless interface (tag 9Ch). At least one of these must be present.

Reminder: The security attributes coding (AMB and SCBs) applies only to the application phase.

Access Mode Byte

The access mode byte determines the commands that are to be controlled. Its coding has a different meaning for files and data objects since the commands that are used with them are different.

Caution: In both cases, if a bit in the AMB is set to zero, that is there is no corresponding SCB, then the security attribute for the corresponding command is NEVER.

The AMB is coded as follows:

Meaning	b8	b7	b6	b5	b4	b3	b2	b1
	0		0	0	0			0

Delete DF (self deletion)		1						
Create DF (applies only to the root)						1		
Create EF							1	

Table 63. Access Mode Byte coding – for DFs

As DFs can be created under the root only, the Create DF condition is used only by the root.

Meaning	**b8**	**b7**	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**
	0		0			0		
Delete EF (current EF)		1						
Activate File				1				
Deactivate File					1			
Update Binary, **Erase Binary**							1	
Read Binary								1

Table 64. Access Mode Byte coding – for EFs

Bit 8 must be zero for EFs. Bits b7 to b1 are independent of each other, that is, more than one bit can be set to 1.

Meaning	Type of Data Object	b8	b7	b6	b5	b4	b3	b2	b1
	—	1					0		
Reset Retry Counter	PIN / FP		1						
Change Reference Data	PIN			1					
Verify or **PSO-Hash** and **PSO Compute Digital Signature (DTBS)**	PIN / FP or Private Key				1				
PSO Compute Digital Signature, **PSO: Decipher**	Private Key					1			
Put Data (Update) **Generate PK Pair**	Secret/Private/ Public Key Private Key DH Key Parameters Private Key							1	
Get Data	DH Key Parameters Public Key								1

Table 65. Access Mode Byte coding – for data objects

Bit 8 must be 1 for data objects, which are for secret keys, private and public keys and PINs. Bits b4 and b2 are independent of each other, that is, both can be set to 1.

When bit 5 is used for private keys, it is intended to protect the DSI before performing a signature.

Note: The **Put Data (Update)** and **Generate PK Pair** commands share the same bit, b2. It is possible upon SDO creation to define which of these commands (or both), the access mode bit refers to.

Security Condition Byte

There is one security condition byte (SCB) for each bit set to one in the AMB. Each SCB is coded as follows:

Meaning	b8	b7	b6	b5	b4	b3	b2	b1
No condition (ALWAYS)	0	0	0	0	0	0	0	0
Never	1	1	1	1	1	1	1	1
Conditions	x	x	x	x				

At least one condition (OR) (b7 to b5)	0							
All conditions (AND) (b7 to b5)	1							
Secure Messaging (command & response)		1						
Mutual Authentication			1					
User Authentication (PIN, FP or MS 3DES3 Auth)				1				
Security Environment Reference					x	x	x	x
No SE referenced					0	0	0	0
SE# 0001–1110					—	—	—	—
RFU					1	1	1	1

Bits 8–5 indicate the conditions that must be fulfilled to access the file or data object.

Bit 7 set to 1 indicates that the command and the response must be sent with secure messaging.

Security Attributes Example

The AMB and SCBs are encapsulated together in TLV format under the tag 8Ch for the contact interface security attribute and the tag 9Ch for the contactless interface security attribute. The length depends on the number of SM bytes specified in the AMB. The following table shows an example of how a security attribute could be coded:

Tag	Length (bytes)	Value	Meaning
8Ch	06h		Tag and length of the security attribute for contact interface (AMB + 4 SCBs)
		5Bh	AMBs. All five of the SCB for files are present.
		12h	SCB for **Delete File** command. PIN protection defined in SE#2.
		14h	SCB for **Activate File** command. PIN protection defined in SE#4.
		35h	SCB for **Deactivate File** command. **Mutual Authentication** OR PIN protection (using the PIN referenced in SE#5).
		B3h	SCB for **Update Binary** and **Erase Binary** commands. **Mutual Authentication** AND PIN protection (using the PIN referenced in SE#3).
		00h	SCB for **Read Binary** command. No condition.
9Ch	04h		Tag and length of the security attribute for contactless interface (AMB + 3 SCBs)
		19h	AMB. Three SCBs for files are present.
		14h	SCB for **Activate File** command. PIN protection defined in SE#4.
		35h	SCB for **Deactivate File** command. **Mutual Authentication** OR PIN protection (using the PIN referenced in SE#5).
		00h	SCB for **Read Binary** command. No condition.

Table 66. Security Attributes coding example for an EF with both interfaces

---

Caution: Care should be taken when coding security attributes, particularly when they involve security environments and linking command usage to life cycle status. To illustrate the potential problems, here is an example.

A security attribute is coded so that in SE#4, the ACTIVATE FILE command can be performed only when the life cycle status of the file is INITIALIZED. If the file is later deactivated by a DEACTIVATE FILE command, you will never be able to reactivate it in SE#4 because the status cannot revert to INITIALIZED.

- END OF DOCUMENT -