INTERNATIONAL STANDARD ISO/IEC 18013-3:2009
TECHNICAL CORRIGENDUM 1

Published 2011-12-01

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ • COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

Information technology — Personal identification — ISO-compliant driving licence —

Part 3:
Access control, authentication and integrity validation

TECHNICAL CORRIGENDUM 1

Technologies de l'information — Identification des personnes — Permis de conduire conforme à l'ISO

Partie 3: Contrôle d'accès, authentification et validation d'intégrité

RECTIFICATIF TECHNIQUE 1

Technical Corrigendum 1 to ISO/IEC 18013-3:2009 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 17, *Cards and personal identification*.

ICS 35.240.15 **Ref. No. ISO/IEC 18013-3:2009/Cor.1:2011(E)**

Published in Switzerland

Page 19, 8.2.4.2

Delete the following sentence:

“The SHA-1 hashing algorithm and trailer option 1 shall be used.”

Page 30, 8.6.3

Add the following sentence before the NOTE beginning with “Both BAP and EAP use the commands GET CHALLENGE …”:

“A SIC that supports EAP shall respond to unauthenticated read attempts of EAP protected data group (including selection of files) with ‘Security status not satisfied’ (0X6982).”

Page 93, C.4.4.5

Add the following sentence before the 5th paragraph beginning with “If the signature verification is successful, access to data groups…” :

“EXTERNAL AUTHENTICATE command can be used only if Path length constraint of the certificate sent previously with PSO_VERIFY command is set to 0. If Path length constraint is >0, the card shall return an ISO checking error and take no further action.”

Page 43, Annex B

Replace Annex B with the following:

Annex B
(normative)

Basic access protection

B.1 Introduction

BAP is a mechanism and protocol to protect identity documents with a SIC against skimming attacks.

This protection is achieved by requiring the establishment of a secure channel using pre-defined key material which should only be revealed by closer physical inspection of the document, before access is granted to information stored in the SIC.

The secure channel protects the integrity and authenticity of authorized communication between an IS and an identity document. If the entropy of the key material is high enough, a certain amount of protection against eavesdropping attacks is also achieved.

NOTE 1 Because the same pre-defined key material is used for all communication sessions with a given document, this protocol does not give forward-secrecy. This means that, if knowledge about the keying material is gained, it can be used to decrypt any past sessions. However, knowledge of a particular session's session keys does not enable the decryption of past or future sessions.

NOTE 2 This protocol is largely based on ICAO's Basic Access Control, which can be viewed as an application of BAP specific to machine-readable travel documents.

B.2 Parameters

IA's implementing BAP shall select:

a) a cryptographic hash function *h*,

b) a block cipher *e* with block length *n* (in bits) and key length *k* (in bits), and

c) a cipher-based message authentication code (MAC) algorithm *m*.

Valid combinations thereof are listed in B.8.

Referring specifications shall specify the following when referencing BAP:

a) The source of K_{doc}.

b) The method(s) by which K_{doc} is entered into the IS.

c) The BAP configuration used.

NOTE ISO/IEC 18013-3 uses information printed on the IDL in machine and/or human-readable form as K_{doc}. The SAI demarcates the information used. The first byte of the input string indicates which BAP configuration is used. For further details, see 8.3 and 8.5.

B.3 Protocol

BAP comprises the following steps:

a) Document basic access keys are established using the key derivation mechanism described in B.4.

b) The IS and the SIC mutually authenticate and derive session keys. The authentication and key establishment protocol described in B.5 is used.

c) After successful authentication, subsequent communication is protected by Secure Messaging as described in B.6. Access shall only be granted as long as secure messaging is active.

B.4 Key derivation mechanism

The following key derivation mechanism is used to derive keys from a key seed (K_{seed}) for both the establishment of the document basic access keys and the establishment of the session keys for secure messaging.

A 32-bit counter c is used to allow for deriving multiple keys from a single seed. Depending on whether a key is used for encryption or MAC computation, the following values shall be used:

- c = 1 (i.e. '00 00 00 01') for encryption,
- c = 2 (i.e. '00 00 00 02') for MAC computation.

The following steps are performed to derive a key K from the seed K_{seed} and c using the selected cryptographic hash function *h*:

1. Let D be the concatenation of K_{seed} and c ($D = K_{seed} \| c$).
2. Using *h*, calculate the hash H of D ($H = h(D)$).
3. The *k* left-most bits of H form the key K.

The document basic access keys K_{enc} and K_{mac} are derived using the mechanism described above, with c = 1 and 2 respectively. In addition, the most significant 16 bytes of the $h(K_{doc})$ is used as the value for K_{seed} . *h* is the selected cryptographic hash function defined the BAP configuration.

K_{doc} should be different for every document and care should be taken to ensure that K_{doc} is sufficiently random for the intended application.

NOTE The entropy of K_{doc} is the upper bound on available entropy for the secure messaging keys. For example, if K_{doc} only provides 30 bits of entropy, the derived keys cannot contain more – even if the key size is larger.

B.5 Authentication and key establishment

Authentication and key establishment is provided by a three pass challenge-response protocol according to ISO/IEC 11770-2 key establishment mechanism 6 using the selected block cipher *e*. A cryptographic checksum according to the selected MAC algorithm *m* is calculated over and appended to the cipher texts. The modes of operation described in B.7 shall be used. Exchanged nonces shall have a size of 64 bits, exchanged keying material shall be *k* bits long. Distinguishing identifiers shall not be used.

In more detail, the IS and SIC perform the following steps[1]:

1. The IS requests a challenge RND.ICC by sending the GET CHALLENGE command.
2. The SIC generates and responds with a random nonce RND.ICC.
3. The IS performs the following operations:

   a) Generate a random nonce RND.IFD and random keying material K.IFD.

[1] The abbreviations IS and SIC are used here are equivalent to IFD and ICC respectively as used in ISO/IEC 7501-1 (ICAO Doc 9303-1).

b) Generate the concatenation S = RND.IFD || RND.ICC || K.IFD.

c) Compute the cryptogram E_IFD = $e[K_{enc}](S)$.

d) Compute the checksum M_IFD = $m[K_{mac}](E\_IFD)$.

e) Send a MUTUAL AUTHENTICATE command using the data E_IFD || M_IFD.

4. The SIC performs the following operations:

a) Check the checksum M_IFD of the cryptogram E_IFD.

b) Decrypt the cryptogram E_IFD.

c) Extract RND.ICC from S and check if the IS returned the correct value.

d) Generate random keying material K.ICC.

e) Generate the concatenation R = RND.ICC || RND.IFD || K.ICC

f) Compute the cryptogram E_ICC = $e[K_{enc}](R)$.

g) Compute the checksum M_ICC = $m[K_{mac}](E\_ICC)$.

h) Send the response using the data E_ICC || M_ICC.

5. The IS performs the following operations:

a) Check the checksum M_ICC of the cryptogram E_ICC.

b) Decrypt the cryptogram E_ICC.

c) Extract RND.IFD from R and check if the SIC returned the correct value.

B.6 Secure messaging

After a successful execution of the authentication protocol, both the IS and the SIC compute session keys KS_{enc} and KS_{mac} using the key derivation mechanism described in B.4 with (K.ICC ⊕ K.IFD) as key seed. All further communication shall be protected by secure messaging (SM) as described in ISO/IEC 7816-4 according to the requirements below. The modes of operation described in B.7 shall be used.

B.6.1 Message structure of SM APDUs

The SM data objects shall be used according to Table B.1 — Usage of SM Data Objects in the following order:

- Command APDU: [DO‘87’] [DO‘97’] DO‘8E’.
- Response APDU: [DO‘87’] DO‘99’ DO‘8E’.

All SM data objects shall be encoded in BER-TLV as specified in ISO/IEC 7816-4. The command header shall be included in the MAC calculation, therefore the class byte CLA shall be ‘0C’.

The actual value of Lc will be modified to Lc’ after application of secure messaging. In the protected command APDU the *new Le* byte shall be set to ‘00’, while the value of the original Le byte may be conveyed in the appropriate data object.

Table B.1 — Usage of SM Data Objects

	DO‘87’	DO‘97’	DO‘99’	DO‘8E’
Content	Padding-content indicator byte (shall be ‘01’) followed by the cryptogram	Le (1 or 2 bytes)	Processing status (SW1-SW2)	Cryptographic checksum (MAC)
Command APDU	Mandatory if data is sent, otherwise absent.	Mandatory if data is requested, otherwise absent.	Not used.	Mandatory.
Response APDU	Mandatory if data is returned, otherwise absent.	Not used.	Mandatory, only absent when a SM error occurs.	Mandatory if DO‘87’ and/or DO‘99’ are present.

Figure B.1 — Computation of a SM command APDUB. shows the transformation of an unprotected command APDU to a protected command APDU in the case that data and Le are available (case 4). If no data is available (case 1 and 2), leave building DO‘87’ out. If Le is not available (case 1 and 3), leave building DO‘97’ out.

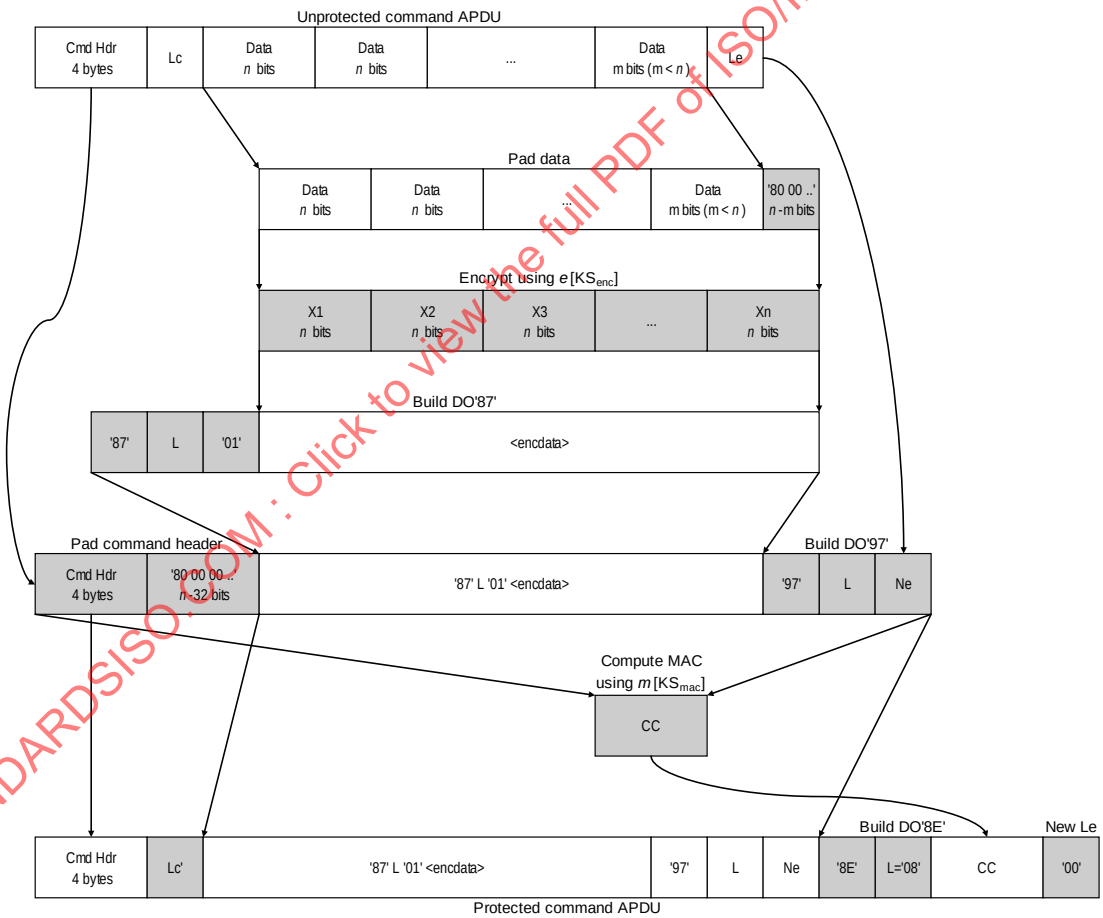

Figure B.1 — Computation of a SM command APDUB.

Figure B.2 shows the transformation of an unprotected response APDU to a protected response APDU in case data is available. If no data is available, leave building DO'87' out.

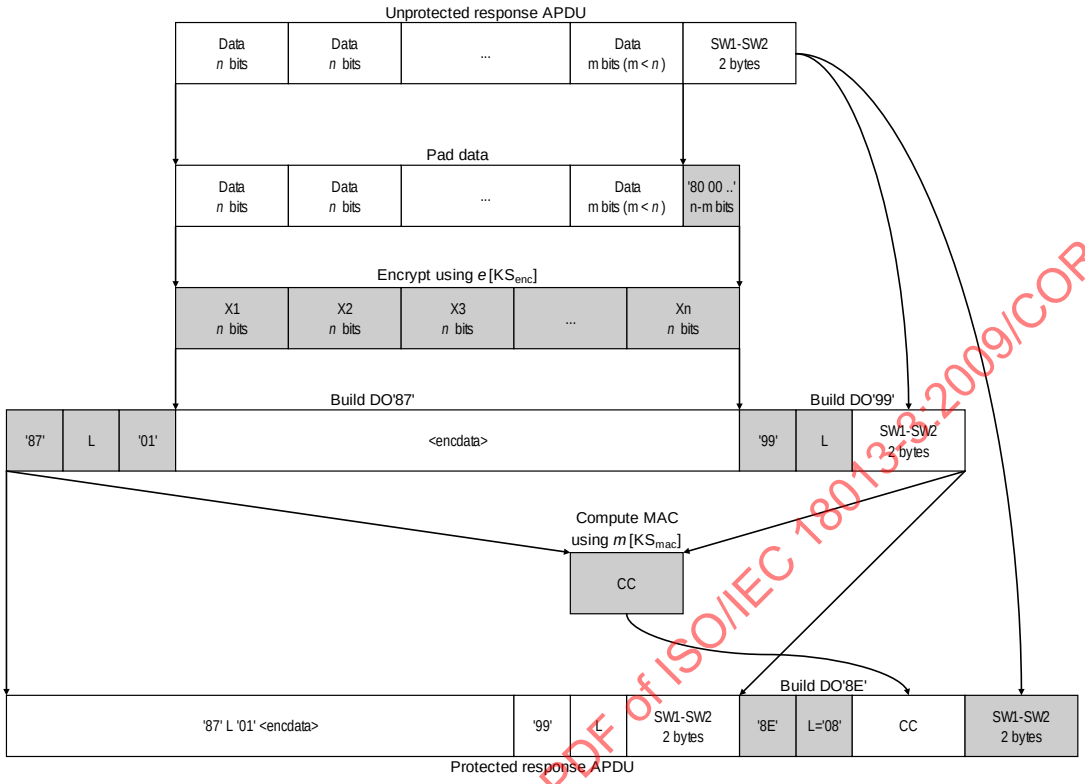

Figure B.2 — Computation of a SM response APDU

B.6.2 SM errors

When the integrated circuit in the document recognizes an SM error while interpreting a command, then the secure messaging session shall be aborted and the status words returned in plain. ISO/IEC 7816-4 defines the following status bytes to indicate SM errors:

a) '6987' Expected SM data objects missing

b) '6988' SM data objects incorrect

B.6.3 Other errors

In the application context, other errors (i.e. status words other than '90 00') may occur that are protected under SM. Under these conditions SM shall not be aborted.

B.7 Modes of operation

B.7.1 Encryption

During encryption, the selected block cipher shall operate in cipher block chaining (CBC) mode with an initialization vector (IV) of *n* '0' bits.

During authentication, the data to be encrypted shall only be padded if it is not a multiple of the block cipher's block length *n*. During the computation of SM APDUs, data shall always be padded.

Padding according to ISO/IEC 9797-1 padding method 2 shall be used.

B.7.2 Message authentication

Cryptographic checksums are calculated using the selected MAC algorithm with an initialization vector (IV) of *n* '0' bits. The MAC length shall be 8 bytes.

After a successful authentication, the datagram to be MACed shall be prepended with an 8 byte send sequence counter (SSC). If the block length n is larger than 64 bits, the SSC shall be prepended by n-64 zero bits to form a full block. The SSC is incremented every time before a MAC is calculated, i.e. if the starting value is x, in the first command the value of SSC is x+1. The value of SSC for the first response is then x+2. The initial value of the SSC is computed by concatenating the four least significant bytes of RND.ICC and RND.IFD, respectively:

SSC = RND.ICC (4 least significant bytes) || RND.IFD (4 least significant bytes)

B.8 Basic access protection configurations

When selecting *h*, *e*, *n*, *k* and *m*, IA's shall pick a valid combination, herein called "configuration", from the choices in this section.

Referring specifications shall either limit the choice of configurations to one, or specify a way to convey the chosen configuration to an IS.

NOTE The criteria for selecting a configuration are typically determined by the application's security, performance and cost requirements.

The following algorithms are used by the configurations:

- SHA-1 and SHA-256 according to ISO/IEC 10118-3;
- TDEA according to ISO/IEC 18033-3 ("Triple DES");
- AES-128, AES-192 and AES-256 according to ISO/IEC 18033-3;
- ISO/IEC 9797-1 MAC algorithm 3;
- CMAC according to NIST SP 800-38B.

Table B.2 — BAP configuration 1

One-byte identifier	'31'
OID	`bap-config-1`
Hash Algorithm *h*	SHA-1
Block Cipher *e*	TDEA using keying option 2. The left-most 64 bits of the 128-bit key form K1, while the right-most 64 bits form K2.
Block Length *n*	64 bits
Key Length *k*	128 bits Note that only 112 bits are effectively used by this block cipher as keying material; certain implementations may require adjustment of the remaining parity bits.
MAC Algorithm *m*	ISO/IEC 9797-1 MAC algorithm 3 with block cipher TDEA and padding method 2. TDEA is used with the keying option that K1=K2=K3 (reduces to DEA). For MAC calculation, the left-most 64 bits of the 128-bit key form K, while the right-most 64 bits form K'. The resulting MAC algorithm is also known as "Retail MAC".

BAP configuration 1 is equivalent to Basic Access Control (BAC) as described in ICAO Doc 9303 (ISO/IEC 7501-1), Annex A, Appendix 5.

Table B.3 — BAP configuration 2

One-byte identifier	'32'
OID	`bap-config-2`
Hash Algorithm *h*	SHA-1
Block Cipher *e*	AES-128
Block Length *n*	128 bits
Key Length *k*	128 bits
MAC Algorithm *m*	CMAC using AES-128

Table B.4 — BAP configuration 3

One-byte identifier	'33'
OID	`bap-config-3`
Hash Algorithm *h*	SHA-256
Block Cipher *e*	AES-192
Block Length *n*	128 bits
Key Length *k*	192 bits
MAC Algorithm *m*	CMAC using AES-192

Due to the explicit exclusion of AES-192 in Suite B of the United States National Security Agency, BAP configuration 3 is not recommended.

Table B.5 — BAP configuration 4

One-byte identifier	'34'
OID	`bap-config-4`
Hash Algorithm *h*	SHA-256
Block Cipher *e*	AES-256
Block Length *n*	128 bits
Key Length *k*	256 bits
MAC Algorithm *m*	CMAC using AES-256

The following ASN.1 object identifiers are used to refer to the different BAP configurations:

```
   bap-config-1 OBJECT IDENTIFIER ::= {
      iso(1) standard(0) driving-licence(18013) part-3(3) security-mechanisms(2) id-sm-
BAP(1) 1
   }

   bap-config-2 OBJECT IDENTIFIER ::= {
      iso(1) standard(0) driving-licence(18013) part-3(3) security-mechanisms(2) id-sm-
BAP(1) 2
   }

   bap-config-3 OBJECT IDENTIFIER ::= {
      iso(1) standard(0) driving-licence(18013) part-3(3) security-mechanisms(2) id-sm-
BAP(1) 3
   }

   bap-config-4 OBJECT IDENTIFIER ::= {
      iso(1) standard(0) driving-licence(18013) part-3(3) security-mechanisms(2) id-sm-
BAP(1) 4
   }
```

B.9 Card Commands

B.9.1 GET CHALLENGE

The GET CHALLENGE command receives a (true) random challenge from the card for authentication in the subsequent MUTUAL AUTHENTICATE command.

Table B.6 — Command APDU: GET CHALLENGE

CLA	As defined in ISO/IEC 7816-4
INS	0x84 GET CHALLENGE
P1	0x00 No information given
P2	0x00 (any other values reserved for future use)
Lc field	Absent
Data field	Absent
Le field	0x08

Table B.7 — Response APDU: GET CHALLENGE

Data field	8-byte random challenge (RND.ICC)
SW1-SW2	'9000' Normal processing Other Operating system dependent error

B.9.2 MUTUAL AUTHENTICATE

The MUTUAL AUTHENTICATE command is used to submit the host cryptogram to the card and receive the card cryptogram for mutual authentication.

Table B.8 — Command APDU: MUTUAL AUTHENTICATE

CLA	As defined in ISO/IEC 7816-4
INS	0x82 MUTUAL AUTHENTICATE
P1	0x00 reference algorithm implicitly known
P2	0x00 qualifier reference implicitly known
Lc field	Length of subsequent data field.
Data field	Host cryptogram including MAC (E_IFD || M_IFD).
Le field	0x28 (configurations 1 and 2) 0x38 (configurations 3 and 4)

Table B.9 — Response APDU: MUTUAL AUTHENTICATE

Data field	Card cryptogram including MAC (E_ICC || M_ICC)
SW1-SW2	'9000' Normal processing '6300' Verification failed. Host cryptogram or MAC verification failed Other Operating system dependent error

B.10 Worked examples (informative)

This section provides worked examples for all configurations of BAP. Note that not all steps are explicitly shown.

B.10.1 Example Using Configuration 1

Static document keying material:

K_{doc} = '31239AB9CB282DAF66231DC5A4DF6BFBAE'

Computation of basic access keys:

Input: $K_{seed} = H_{SHA\text{-}1}(K_{doc})$

K_{seed} = 'BFE25204D0A589510CD9C397C064CC2DAF5E952F'

Encryption Key (K_{enc}) computation:

1. Concatenate K_{seed} and c (c = 1):
   D = 'BFE25204D0A589510CD9C397C064CC2D
   00000001'

2. Calculate the hash of D:
$H_{SHA-1}(D)$ = 'AE161CC6AFB5FB766BD20016CAC3F181 E77D9428'

3. Form key:
K_{enc} = 'AE161CC6AFB5FB766BD20016CAC3F181'

$K_1 = K_3$ = 'AE161CC6AFB5FB76'
K_2 = '6BD20016CAC3F181'

Message Authentication Key (K_{mac}) computation:

4. Concatenate K_{seed} and c (c = 2):
D = 'BFE25204D0A589510CD9C397C064CC2D 00000002'

5. Calculate the hash of D:
$H_{SHA-1}(D)$ = '24F522867731552B72533F5D25CC4806 777D5953'

6. Form key:
K_{mac} = '24F522867731552B72533F5D25CC4806'

K = '24F522867731552B'
K' = '72533F5D25CC4806'

Authentication and Establishment of Session Keys:

IS:

1. Request an 8 byte random challenge from the document's SIC:

Command APDU:

CLA	INS	P1	P2	Le
'00'	'84'	'00'	'00'	'08'

Document SIC:

2. Generate random challenge and return it to IS:
RND.ICC = '4608F91988702212'

Response APDU:

Response Data Field	SW1 SW2
RND.ICC	'90' '00'

IS:

3. Generate an 8-byte random challenge and 16-byte random keying material:
RND.IFD = '781723860C06C226'
K.IFD = '0B795240CB7049B01C19B33E32804F0B'

4. Concatenate RND.IFD, RND.ICC and K.IFD:
S = ‘781723860C06C2264608F91988702212
0B795240CB7049B01C19B33E32804F0B’

5. Encrypt S using TDEA with key K_{enc}:
E_IFD = ‘861D8A36082E38FB1F699FFDFAF7F903
ADF74AA79E8459E50080F43ACB096B52’

6. Compute “Retail MAC” over E_IFD with key K_{mac}:
M_IFD = ‘20498D845BE458C3’

7. Construct command data for MUTUAL AUTHENTICATE and send command to the document’s SIC:
cmd_data = ‘861D8A36082E38FB1F699FFDFAF7F903
ADF74AA79E8459E50080F43ACB096B52
20498D845BE458C3’

Command APDU:

CLA	INS	P1	P2	Lc	Command Data Field	Le
‘00’	‘82’	‘00’	‘00’	‘28’	cmd_data	‘28’

Document SIC:

8. Generate 16-byte random keying material:
K.ICC = ‘0B4F80323EB3191CB04970CB4052790B’

9. Calculate XOR of K.IFD and K.ICC:
K_{seed} = ‘0036D272F5C350ACAC50C3F572D23600’

10. Derive session keys:
KS_{enc} = ‘979EC03B1CBFE9DDD11AB1FED206EBE4’
KS_{mac} = ‘F0CA1E1EB5ADF208816B88DD579CC1F8’

11. Initialize send sequence counter:
SSC = ‘887022120C06C226’

12. Concatenate RND.ICC, RND.IFD and K.ICC:
R = ‘4608F91988702212781723860C06C226
0B4F80323EB3191CB04970CB4052790B’

13. Encrypt R using TDEA with key K_{enc}:
E_ICC = ‘C8F977C50533BE2104E68A844040310A
11362AF11EC09D972CE8AD3FDCB9164B’

14. Compute “Retail MAC” over E_ICC with key K_{mac}:
M_ICC = ‘9E8E43F7B5CEDB06’

15. Construct response data and send response APDU to the IS:
resp_data = ‘C8F977C50533BE2104E68A844040310A
11362AF11EC09D972CE8AD3FDCB9164B
9E8E43F7B5CEDB06’

Response APDU:

Response Data Field	SW1 SW2
resp_data	'90' '00'

IS:

16. Calculate XOR of K.IFD and K.ICC:
K_{seed} = '0036D272F5C350ACAC50C3F572D23600'

17. Derive session keys:
KS_{enc} = '969EC03B1CBFE9DDD11AB1FED206EBE4'
KS_{mac} = 'F0CA1E1EB5ADF208816B88DD579CC1F8'

18. Initialize send sequence counter:
SSC = '887022120C06C226'

Secure Messaging:

IS

1. SELECT EF.COM (file identifier = '01 1E'):

Unprotected command APDU:

CLA	INS	P1	P2	Lc	Command Data Field
'00'	'A4'	'02'	'00'	'02'	'01 1E'

a) Mask class byte and pad command header:
cmd_header = '0CA4020C80000000'

b) Pad data:
p_data = '011E800000000000'

c) Encrypt p_data using TDEA with KS_{enc}:
enc_data = '6375432908C044F6'

d) Build DO'87':
DO87 = '8709016375432908C044F6'

e) Concatenate cmd_header and DO87:
M = '0CA4020C8000000087090163754329 08 C044F6'

f) Compute "Retail MAC" of M with KS_{mac}:
- Increment SSC:
SSC = '887022120C06C227'
- Concatenate SSC and M:
N = '887022120C06C2270CA4020C80000000 8709016375432908C044F6'
- Compute MAC:
CC = 'BF8B92D635FF24F8'

g) Build DO‘8E’:
DO8E = ‘8E08BF8B92D635FF24F8’

h) Construct command data:
cmd_data = ‘8709016375432908C044F68E08BF8B92 D635FF24F8’

Protected command APDU:

CLA	INS	P1	P2	Lc	Command Data Field	Le
‘0C’	‘A4’	‘02’	‘0C’	‘15’	cmd_data	‘00’

Document SIC:

2. Set EF.COM as the currently selected file and send affirmative response to IS:

Unprotected response APDU:

SW1	SW2
‘90’	‘00’

a) Build DO‘99’:
DO99 = ‘99029000’

b) Compute “Retail MAC” of DO99 with KS_{mac}:
- Increment SSC:
SSC = ‘887022120C06C228’
- Concatenate SSC and DO99:
N = ‘887022120C06C22899029000’
- Compute MAC:
CC = ‘FA855A5D4C50A8ED’

c) Build DO‘8E’:
DO8E = ‘8E08FA855A5D4C50A8ED’

d) Construct response data:
resp_data = ‘990290008E08FA855A5D4C50A8ED’

Protected response APDU:

Response Data Field	SW1	SW2
resp_data	‘90’	‘00’

IS:

3. READ BINARY of the first 4 bytes:

Unprotected command APDU:

CLA	INS	P1	P2	Le

'00'	'B0'	'00'	'00'	'04'

a) Mask class byte and pad command header:
cmd_header = '0CB0000080000000'

b) Build DO '97':
DO97 = '970104'

c) Concatenate cmd_header and DO97:
M = '0CB000008000000970104'

d) Compute "Retail MAC" of M with KS_{mac}:
- Increment SSC:
SSC = '887022120C06C229'
- Concatenate SSC and M:
N = '887022120C06C2290CB0000080000000
970104'
- Compute MAC:
CC = 'ED6705417E96BA55'

e) Build DO'8E':
DO8E = '8E08ED6705417E96BA55'

f) Construct command data:
cmd_data = '9701048E08ED6705417E96BA55'

Protected command APDU:

CLA	INS	P1	P2	Lc	Command Data Field	Le
'0C'	'B0'	'00'	'00'	'0D'	cmd_data	'00'

Document SIC:

4. Return 4 bytes of EF.COM starting at offset 0:

data = '600D5F01'

Unprotected response APDU:

Response Data Field	SW1 SW2
data	'90' '00'

a) Pad data:
p_data = '600D5F0180000000'

b) Encrypt p_data using TDEA with KS_{enc}:
enc_data = 'F9435D056E27C52E'

c) Build DO'87':
DO87 = '870901F9435D056E27C52E'

d) Build DO‘99’:
DO99 = ‘99029000’

e) Concatenate DO‘87’ and DO‘99’:
M = ‘870901F9435D056E27C52E99029000’

f) Compute “Retail MAC” of M with KS_{mac}:
- Increment SSC:
SSC = ‘887022120C06C22A’
- Concatenate SSC and M:
N = ‘887022120C06C22A870901F9435D056E
27C52E99029000’
- Compute MAC:
CC = ‘0C15238078E0A4C9’

g) Build DO‘8E’:
DO8E = ‘8E080C15238078E0A4C9’

h) Construct response data:
resp_data = ‘870901F9435D056E27C52E990290008E
080C15238078E0A4C9’

Protected response APDU:

Response Data Field	SW1	SW2
resp_data	‘90’	‘00’

IS:

5. READ BINARY of the remaining 11 bytes:

Unprotected command APDU:

CLA	INS	P1	P2	Le
‘00’	‘B0’	‘00’	‘04’	‘0B’

a) Mask class byte and pad command header:
cmd_header = ‘0CB0000480000000’

b) Build DO ‘97’:
DO97 = ‘97010B’

c) Concatenate cmd_header and DO97:
M = ‘0CB000048000000097010B’

d) Compute “Retail MAC” of M with KS_{mac}:
- Increment SSC:
SSC = ‘887022120C06C22B’
- Concatenate SSC and M:
N = ‘887022120C06C22B0CB0000480000000
97010B’
- Compute MAC:
CC = ‘40900A27C4C390D6’

e) Build DO‘8E’:
DO8E = ‘8E0840900A27C4C390D6’

f) Construct command data:
cmd_data = ‘97010B8E0840900A27C4C390D6’

Protected command APDU:

CLA	INS	P1	P2	Lc	Command Data Field	Le
‘0C’	‘B0’	‘00’	‘04’	‘0D’	cmd_data	‘00’

Document SIC:

6. Return 11 bytes of EF.COM starting at offset 4:

data = ‘04303130305C04616B6567’

Unprotected response APDU:

Response Data Field	SW1 SW2
data	‘90’ ‘00’

a) Pad data:
p_data = ‘04303130305C04616B65678000000000’

b) Encrypt p_data using TDEA with KS_{enc}:
enc_data = ‘B3CD0334417393661AA9B39206EC89CC’

c) Build DO‘87’:
DO87 = ‘871101B3CD0334417393661AA9B39206
EC89CC’

d) Build DO‘99’:
DO99 = ‘99029000’

e) Concatenate DO‘87’ and DO‘99’:
M = ‘871101B3CD0334417393661AA9B39206
EC89CC99029000’

f) Compute “Retail MAC” of M with KS_{mac}:
- Increment SSC:
SSC = ‘887022120C06C22C’
- Concatenate SSC and M:
N = ‘887022120C06C22C871101B3CD033441
7393661AA9B39206EC89CC99029000’
- Compute MAC:
CC = ‘0747E8CEC180EB48’

g) Build DO‘8E’:
DO8E = ‘8E080747E8CEC180EB48’

h) Construct response data:
resp_data = ‘871101B3CD0334417393661AA9B39206
EC89CC990290008E080747E8CEC180EB
48’

Protected response APDU:

Response Data Field	SW1	SW2
resp_data	‘90’	‘00’

B.10.2 Example Using Configuration 2

Static document keying material:
K_{doc} = ‘3255CA83CC52EC6454DE7AFB1D3DA66F4E’

Compute Basic Access Keys

Input: K_{seed} = $H_{SHA-1}(K_{doc})$
K_{seed} = ‘4D9AE55D18986679B8F39740755E53DF48FE152A’

Encryption Key (K_{enc}) computation

1. Concatenate K_{seed} and c (c = 1):
D = ‘4D9AE55D18986679B8F39740755E53DF
00000001’

2. Calculate the hash of D:
$H_{SHA-1}(D)$ = ‘F5C60770CF201E80C091A1564D979235
18876DD8’

3. Form key:
K_{enc} = ‘F5C60770CF201E80C091A1564D979235’

Message Authentication Key (K_{mac}) computation

4. Concatenate K_{seed} and c (c = 2):
D = ‘4D9AE55D18986679B8F39740755E53DF
00000002’

5. Calculate the hash of D:
$H_{SHA-1}(D)$ = ‘B6A66D80D698B5B1C6A0AA8C94CCC39D
8ACE393A’

6. Form key:
K_{mac} = ‘B6A66D80D698B5B1C6A0AA8C94CCC39D’

Authentication and Establishment of Session Keys:

IS

1. Request an 8 byte random challenge from the document’s SIC:

Command APDU:

CLA	INS	P1	P2	Le
'00'	'84'	'00'	'00'	'08'

Document SIC:

2. Generate random challenge and return it to IS:
RND.ICC = 'E72450C17A59DF40'

Response APDU:

Response Data Field	SW1 SW2
RND.ICC	'90' '00'

IS:

3. Generate an 8-byte random challenge and 16-byte random keying material:
RND.IFD = '41C51B949D4FD786'
K.IFD = '766F9E2B2B503AFBEB420BED8D1A73A8'

4. Concatenate RND.IFD, RND.ICC and K.IFD:
S = '41C51B949D4FD786E72450C17A59DF40
766F9E2B2B503AFBEB420BED8D1A73A8'

5. Encrypt S using AES with key K_{enc}:
E_IFD = '8335EB0A44172C8124C82F30F6F0E2E2
192E8881208D50E4EF53744CB460E090'

6. Compute CMAC over E_IFD with key K_{mac}:
M_IFD = '0D612FFE0699676C'

7. Construct command data for MUTUAL AUTHENTICATE and send command to the document's SIC:
cmd_data = '8335EB0A44172C8124C82F30F6F0E2E2
192E8881208D50E4EF53744CB460E090
0D612FFE0699676C'

Command APDU:

CLA	INS	P1	P2	Lc	Command Data Field	Le
'00'	'82'	'00'	'00'	'28'	cmd_data	'28'

Document SIC:

8. Generate 16-byte random keying material:
K.ICC = 'C1655F49E136D12B8522B1C99510E71B'

9. Calculate XOR of K.IFD and K.ICC:
K_{seed} = 'B70AC162CA66EBD06E60BA24180A94B3'

10. Derive session keys:
KS_{enc} = ‘AB9497F5819AB69A25A7798789061CF8’
KS_{mac} = ‘1FBF06A0DE775C473D64B5E9933290D3’

11. Initialize send sequence counter:
SSC = ‘7A59DF409D4FD786’

12. Concatenate RND.ICC, RND.IFD and K.ICC:
R = ‘E72450C17A59DF4041C51B949D4FD786
C1655F49E136D12B8522B1C99510E71B’

13. Encrypt R using AES with key K_{enc}:
E_ICC = ‘D95197A57FB69D9104AD5F9C291FE712
B1FD32BEC3D711743EFEA62EB975251C”

14. Compute CMAC over E_ICC with key K_{mac}:
M_ICC = ‘BD7D520A3E783182’

15. Construct response data and send response APDU to the IS:
resp_data = ‘D95197A57FB69D9104AD5F9C291FE712
B1FD32BEC3D711743EFEA62EB975251C
BD7D520A3E783182’

Response APDU:

Response Data Field	SW1	SW2
resp_data	‘90’	‘00’

IS:

16. Calculate XOR of K.IFD and K.ICC:
K_{seed} = ‘B70AC162CA66EBD06E60BA24180A94B3’

17. Derive session keys:
KS_{enc} = ‘AB9497F5819AB69A25A7798789061CF8’
KS_{mac} = ‘1FBF06A0DE775C473D64B5E9933290D3’

18. Initialize send sequence counter:
SSC = ‘7A59DF409D4FD786’

Secure Messaging:

IS:

1. SELECT EF.COM (file identifier = ’01 1E’):

Unprotected command APDU:

CLA	INS	P1	P2	Lc	Command Data Field
‘00’	‘A4’	‘02’	‘00’	‘02’	’01 1E’

a) Mask class byte and pad command header:
cmd_header = ‘0CA4020C8000000000000000000000’

b) Pad data:
p_data = ‘011E800000000000000000000000000’

c) Encrypt p_data using AES with KS_{enc}:
enc_data = ‘4FF75761BC5C1ECE82AE43F70938D50F’

d) Build DO‘87’:
DO87 = ‘8711014FF75761BC5C1ECE82AE43F709
38D50F’

e) Concatenate cmd_header and DO87:
M = ‘0CA4020C800000000000000000000000
8711014FF75761BC5C1ECE82AE43F709
38D50F’

f) Compute CMAC of M with KS_{mac}:
- Increment SSC:
SSC = ‘7A59DF409D4FD787’
- Concatenate padded SSC and M:
N = ‘00000000000000007A59DF409D4FD787
0CA4020C800000000000000000000000
8711014FF75761BC5C1ECE82AE43F709
38D50F’
- Compute MAC:
CC = ‘C795C566BE469CCA’

g) Build DO‘8E’:
DO8E = ‘8E08C795C566BE469CCA’

h) Construct command data:
cmd_data = ‘8711014FF75761BC5C1ECE82AE43F709
38D50F8E08C795C566BE469CCA’

Protected command APDU:

CLA	INS	P1	P2	Lc	Command Data Field	Le
‘0C’	‘A4’	‘02’	‘0C’	‘1D’	cmd_data	‘00’

Document SIC:

2. Set EF.COM as the currently selected file and send affirmative response to IS:

Unprotected response APDU:

SW1	SW2
‘90’	‘00’

a) Build DO‘99’:
DO99 = ‘99029000’

b) Compute CMAC of DO99 with KS_{mac}:
- Increment SSC:
  SSC = '7A59DF409D4FD788'
- Concatenate padded SSC and DO99:
  N = '00000000000000007A59DF409D4FD788
  99029000'
- Compute MAC:
  CC = '98AB75F63BE65DD8'

c) Build DO'8E':
DO8E = '8E0898AB75F63BE65DD8'

d) Construct response data:
resp_data = '990290008E0898AB75F63BE65DD8'

Protected response APDU:

Response Data Field	SW1 SW2
resp_data	'90' '00'

IS:

3. READ BINARY of the first 4 bytes:

Unprotected command APDU:

CLA	INS	P1	P2	Le
'00'	'B0'	'00'	'00'	'04'

a) Mask class byte and pad command header:
cmd_header = '0CB000008000000000000000000000000'

b) Build DO '97':
DO97 = '970104'

c) Concatenate cmd_header and DO97:
M = '0CB000008000000000000000000000000
970104'

d) Compute CMAC of M with KS_{mac}:
- Increment SSC:
  SSC = '7A59DF409D4FD789'
- Concatenate padded SSC and M:
  N = '00000000000000007A59DF409D4FD789
  0CB000008000000000000000000000000
  970104'
- Compute MAC:
  CC = '67BB878136DD4520'

e) Build DO'8E':
DO8E = '8E0867BB878136DD4520'

f) Construct command data:
cmd_data = ‘9701048E0867BB878136DD4520’

Protected command APDU:

CLA	INS	P1	P2	Lc	Command Data Field	Le
‘0C’	‘B0’	‘00’	‘00’	‘0D’	cmd_data	‘00’

Document SIC

4. Return 4 bytes of EF.COM starting at offset 0:

data = ‘600D5F01’

Unprotected response APDU:

Response Data Field	SW1 SW2
data	‘90’ ‘00’

a) Pad data:
p_data = ‘600D5F0180000000000000000000000000’

b) Encrypt p_data using AES with KS_{enc}:
enc_data = ‘D60D14976646FB2304A0155F6BC6E42D’

c) Build DO‘87’:
DO87 = ‘871101D60D14976646FB2304A0155F6B
C6E42D’

d) Build DO‘99’:
DO99 = ‘99029000’

e) Concatenate DO‘87’ and DO‘99’:
M = ‘871101D60D14976646FB2304A0155F6B
C6E42D99029000’

f) Compute CMAC of M with KS_{mac}:
- Increment SSC:
SSC = ‘7A59DF409D4FD78A’
- Concatenate padded SSC and M:
N = ‘00000000000000007A59DF409D4FD78A
871101D60D14976646FB2304A0155F6B
C6E42D99029000’
- Compute MAC:
CC = ‘834E374C7D5C5A56’

g) Build DO‘8E’:
DO8E = ‘8E08834E374C7D5C5A56’

h) Construct response data:
resp_data = ‘871101D60D14976646FB2304A0155F6B
C6E42D990290008E08834E374C7D5C5A
56

Protected response APDU:

Response Data Field	SW1 SW2
resp_data	‘90’ ‘00’

IS:

5. READ BINARY of the remaining 11 bytes:

Unprotected command APDU:

CLA	INS	P1	P2	Le
‘00’	‘B0’	‘00’	‘04’	‘0B’

a) Mask class byte and pad command header:
cmd_header = ‘0CB000048000000000000000000000000’

b) Build DO ‘97’:
DO97 = ‘97010B’

c) Concatenate cmd_header and DO97:
M = ‘0CB000048000000000000000000000000
97010B’

d) Compute CMAC of M with KS_{mac}:
- Increment SSC:
SSC = ‘7A59DF409D4FD78B’
- Concatenate padded SSC and M:
N = ‘00000000000000007A59DF409D4FD78B
0CB000048000000000000000000000000
97010B’
- Compute MAC:
CC = ‘DC3ACC1CF7CC990C’

e) Build DO‘8E’:
DO8E = ‘8E08DC3ACC1CF7CC990C’

f) Construct command data:
cmd_data = ‘97010B8E08DC3ACC1CF7CC990C’

Protected command APDU:

CLA	INS	P1	P2	Lc	Command Data Field	Le
‘0C’	‘B0’	‘00’	‘04’	‘0D’	cmd_data	‘00’

Document SIC:

6. Return 11 bytes of EF.COM starting at offset 4:

data = ‘04303130305C04616B6567’

Unprotected response APDU:

Response Data Field	SW1 SW2
data	‘90’ ‘00’

a) Pad data:
p_data = ‘04303130305C04616B65678000000000’

b) Encrypt p_data using AES with KS_{enc}:
enc_data = ‘36B83A1FBAC98D89DDDA2235AD29A8BB’

c) Build DO‘87’:
DO87 = ‘87110136B83A1FBAC98D89DDDA2235AD
29A8BB’

d) Build DO‘99’:
DO99 = ‘99029000’

e) Concatenate DO‘87’ and DO‘99’:
M = ‘87110136B83A1FBAC98D89DDDA2235AD
29A8BB99029000’

f) Compute CMAC of M with KS_{mac}:
- Increment SSC:
SSC = ‘7A59DF409D4FD78C’
- Concatenate padded SSC and M:
N = ‘00000000000000007A59DF409D4FD78C87110136B83A1FBA
C98D89DDDA2235AD29A8BB99029000’
- Compute MAC:
CC = ‘FA99B42BF9D1B884’

g) Build DO‘8E’:
DO8E = ‘8E08FA99B42BF9D1B884’

h) Construct response data:
resp_data = ‘87110136B83A1FBAC98D89DDDA2235AD
29A8BB990290008E08FA99B42BF9D1B8
84’

Protected response APDU:

Response Data Field	SW1 SW2
resp_data	‘90’ ‘00’

B.10.3 Example Using Configuration 3

Static document keying material:

K_{doc} = '33B59B332F43A1AC5311DE3C8CE009340F 57B53210E3A7C092'

Compute Basic Access Keys:

Input: $K_{seed} = H_{SHA-256}(K_{doc})$
K_{seed} = ' D409C6CC7292EA567501E127D2A3FDF2 8140F1483C053F1AA1F9D10B05B2F5DF'

Encryption Key (K_{enc}) computation:

1. Concatenate K_{seed} and c (c = 1):
   D = 'D409C6CC7292EA567501E127D2A3FDF2 00000001'

2. Calculate the hash of D:
   $H_{SHA-256}(D)$ = 'BB30BD6D81A16BF71D4C96BEAA8714C2 895E2B010B97832C7A876CA7C371008B'

3. Form key:
   K_{enc} = 'BB30BD6D81A16BF71D4C96BEAA8714C2 895E2B010B97832C'

Message Authentication Key (K_{mac}) computation:

4. Concatenate K_{seed} and c (c = 2):
   D = 'D409C6CC7292EA567501E127D2A3FDF2 00000002'

5. Calculate the hash of D:
   $H_{SHA-256}(D)$ = '35E8880F471F29430BC3FE3376706DF2 18465DC8C5635E8E79D264BEB0A7CA0E'

6. Form key:
   K_{mac} = '35E8880F471F29430BC3FE3376706DF2 18465DC8C5635E8E'

Authentication and Establishment of Session Keys:

IS:

1. Request an 8 byte random challenge from the document's SIC:

   Command APDU:

CLA	INS	P1	P2	Le
'00'	'84'	'00'	'00'	'08'

Document SIC:

2. Generate random challenge and return it to IS:
RND.ICC = ‘A724E1735E5D5B63’

Response APDU:

Response Data Field	SW1	SW2
RND.ICC	‘90’	‘00’

IS:

3. Generate an 8-byte random challenge and 24-byte random keying material:
RND.IFD = ‘DBA0881FC039F4B4’
K.IFD = ‘82D950B0EF5C5B36AF3FB362F7431AC1
A8EC1A4581CAF682’

4. Concatenate RND.IFD, RND.ICC and K.IFD; and add padding:
S = ‘DBA0881FC039F4B4A724E1735E5D5B63
82D950B0EF5C5B36AF3FB362F7431AC1
A8EC1A4581CAF6828000000000000000’

5. Encrypt S using AES with key K_{enc}:
E_IFD = ‘799BEADC2659E2F07668F320D277AD0D
D18F4714BED4B29D340FB8B3B6515EB0
69979518B04DD7A1A0AA54FC8204253E’

6. Compute CMAC over E_IFD with key K_{mac}:
M_IFD = ‘360A2A7E4912A35D’

7. Construct command data for MUTUAL AUTHENTICATE and send command to the document’s SIC:
cmd_data = ‘799BEADC2659E2F07668F320D277AD0D
D18F4714BED4B29D340FB8B3B6515EB0
69979518B04DD7A1A0AA54FC8204253E
360A2A7E4912A35D’

Command APDU:

CLA	INS	P1	P2	Lc	Command Data Field	Le
‘00’	‘82’	‘00’	‘00’	‘38’	cmd_data	‘38’

Document SIC:

8. Generate 16-byte random keying material:
K.ICC = ‘3C2F19D7B42105F25C81B07C78E57BF4
5818DB906CB9360D’

9. Calculate XOR of K.IFD and K.ICC:
K_{seed} = ‘BEF649675B7D5EC4F3BE031E8FA66135
F0F4C1D5ED73C08F’

10. Derive session keys:
KS_{enc} = ‘0E6F93A7699C8EB5FE3B0CF30BC1EAF8

2796411E137058EA'
KS_{mac} = '86D8E8AE5752B1C8268D8566F973BCB6
E1383A083F65181A'

11. Initialize send sequence counter:
SSC = '5E5D5B63C039F4B4'

12. Concatenate RND.ICC, RND.IFD and K.ICC; and add padding:
R = 'A724E1735E5D5B63DBA0881FC039F4B4
3C2F19D7B42105F25C81B07C78E57BF4
5818DB906CB9360D8000000000000000'

13. Encrypt R using AES with key K_{enc}:
E_ICC = '0F6DD1745961AD4B490CEA9A9438BAF2
B346C15373C9762C6D73729DD8042F42
D38BFBC5C2815377DEC2261A7E7091E4

14. Compute CMAC over E_ICC with key K_{mac}:
M_ICC = 'F1A924D6231626A2'

15. Construct response data and send response APDU to the IS:
resp_data = '0F6DD1745961AD4B490CEA9A9438BAF2
B346C15373C9762C6D73729DD8042F42
D38BFBC5C2815377DEC2261A7E7091E4
F1A924D6231626A2'

Response APDU:

Response Data Field	SW1	SW2
resp_data	'90'	'00'

IS:

16. Calculate XOR of K.IFD and K.ICC:
K_{seed} = 'BEF649675B7D5EC4F3BE031E8FA66135
F0F4C1D5ED73C08F'

17. Derive session keys:
KS_{enc} = '0E6F93A7699C8EB5FE3B0CF30BC1EAF8
2796411E137058EA'
KS_{mac} = '86D8E8AE5752B1C8268D8566F973BCB6
E1383A083F65181A'

18. Initialize send sequence counter:
SSC = '5E5D5B63C039F4B4'

Secure Messaging:

IS:

1. SELECT EF.COM (file identifier = '01 1E'):

Unprotected command APDU:

CLA	INS	P1	P2	Lc	Command Data Field
'00'	'A4'	'02'	'00'	'02'	'01 1E'

a) Mask class byte and pad command header:
cmd_header = '0CA4020C8000000000000000000000000'

b) Pad data:
p_data = '011E8000000000000000000000000000'

c) Encrypt p_data using AES with KS_{enc}:
enc_data = '6106F66E07CD676ACAE7E4DCD53140F0'

d) Build DO'87':
DO87 = '8711016106F66E07CD676ACAE7E4DCD5
3140F0'

e) Concatenate cmd_header and DO87:
M = '0CA4020C8000000000000000000000000
8711016106F66E07CD676ACAE7E4DCD5
3140F0'

f) Compute CMAC of M with KS_{mac}:
- Increment SSC:
SSC = '5E5D5B63C039F4B5'
- Concatenate padded SSC and M:
N = '0000000000000005E5D5B63C039F4B5
0CA4020C8000000000000000000000000
8711016106F66E07CD676ACAE7E4DCD5
3140F0'
- Compute MAC:
CC = 'F5F8F24E1A75A67D'

g) Build DO'8E':
DO8E = '8E08F5F8F24E1A75A67D'

h) Construct command data:
cmd_data = '8711016106F66E07CD676ACAE7E4DCD5
3140F08E08F5F8F24E1A75A67D'

Protected command APDU:

CLA	INS	P1	P2	Lc	Command Data Field	Le
'0C'	'A4'	'02'	'0C'	'1D'	cmd_data	'00'

Document SIC:

2. Set EF.COM as the currently selected file and send affirmative response to IS:

Unprotected response APDU:

SW1	SW2
'90'	'00'

a) Build DO'99':
DO99 = '99029000'

b) Compute CMAC of DO99 with KS_{mac}:
- Increment SSC:
SSC = '5E5D5B63C039F4B6'
- Concatenate padded SSC and DO99:
N = '00000000000000005E5D5B63C039F4B6
99029000'
- Compute MAC:
CC = '5ACDD04195E4D582'

c) Build DO'8E':
DO8E = '8E085ACDD04195E4D582'

d) Construct response data:
resp_data = '990290008E085ACDD04195E4D582'

Protected response APDU:

Response Data Field	SW1	SW2
resp_data	'90'	'00'

IS:

3. READ BINARY of the first 4 bytes:

Unprotected command APDU:

CLA	INS	P1	P2	Le
'00'	'B0'	'00'	'00'	'04'

a) Mask class byte and pad command header:
cmd_header = '0CB000008000000000000000000000000'

b) Build DO '97':
DO97 = '970104'

c) Concatenate cmd_header and DO97:
M = '0CB000008000000000000000000000000
970104'

d) Compute CMAC of M with KS_{mac}:
- Increment SSC:
  SSC = '5E5D5B63C039F4B7'
- Concatenate padded SSC and M:
  N = '0000000000000005E5D5B63C039F4B7
  0CB0000080000000000000000000000
  970104'
- Compute MAC:
  CC = '42A89F1958CB432C'

e) Build DO'8E':
DO8E = '8E0842A89F1958CB432C'

f) Construct command data:
cmd_data = '9701048E0842A89F1958CB432C'

Protected command APDU:

CLA	INS	P1	P2	Lc	Command Data Field	Le
'0C'	'B0'	'00'	'00'	'0D'	cmd_data	'00'

Document SIC:

4. Return 4 bytes of EF.COM starting at offset 0:

data = '600D5F01'

Unprotected response APDU:

Response Data Field	SW1	SW2
data	'90'	'00'

a) Pad data:
p_data = '600D5F0180000000000000000000000'

b) Encrypt p_data using AES with KS_{enc}:
enc_data = 'CE7B3391BE1A2270A0772F99A90606CA'

c) Build DO'87':
DO87 = '871101CE7B3391BE1A2270A0772F99A9
0606CA'

d) Build DO'99':
DO99 = '99029000'

e) Concatenate DO'87' and DO'99':
M = '871101CE7B3391BE1A2270A0772F99A9
0606CA99029000'

f) Compute CMAC of M with KS_{mac}:
- Increment SSC:
  SSC = '5E5D5B63C039F4B8'
- Concatenate padded SSC and M:
  N = '0000000000000005E5D5B63C039F4B8

871101CE7B3391BE1A2270A0772F99A9
0606CA99029000'
- Compute MAC:
CC = '47A49E91EDBCC466'

g) Build DO'8E':
DO8E = '8E0847A49E91EDBCC466'

h) Construct response data:
resp_data = '871101CE7B3391BE1A2270A0772F99A9
0606CA990290008E0847A49E91EDBCC4
66'

Protected response APDU:

Response Data Field	SW1	SW2
resp_data	'90'	'00'

IS:

5. READ BINARY of the remaining 11 bytes:

Unprotected command APDU:

CLA	INS	P1	P2	Le
'00'	'B0'	'00'	'04'	'0B'

a) Mask class byte and pad command header:
cmd_header = '0CB000048000000000000000000000000'

b) Build DO '97':
DO97 = '97010B'

c) Concatenate cmd_header and DO97:
M = '0CB000048000000000000000000000000
97010B'

d) Compute CMAC of M with KS_{mac}:
- Increment SSC:
SSC = '5E5D5B63C039F4B9'
- Concatenate padded SSC and M:
N = '00000000000000005E5D5B63C039F4B9
0CB000048000000000000000000000000
97010B'
- Compute MAC:
CC = '2B5D0653A9ED3EBF'

e) Build DO'8E':
DO8E = '8E082B5D0653A9ED3EBF'

f) Construct command data:
cmd_data = '97010B8E082B5D0653A9ED3EBF'

Protected command APDU:

CLA	INS	P1	P2	Lc	Command Data Field	Le
‘0C’	‘B0’	‘00’	‘04’	‘0D’	cmd_data	‘00’

Document SIC:

6. Return 11 bytes of EF.COM starting at offset 4:

data = ‘04303130305C04616B6567’

Unprotected response APDU:

Response Data Field	SW1	SW2
data	‘90’	‘00’

a) Pad data:
p_data = ‘04303130305C04616B656780000000000’

b) Encrypt p_data using AES with KS_{enc}:
enc_data = ‘D3B264B934AB8E7BDA7DCEEEDA79AA39’

c) Build DO‘87’:
DO87 = ‘871101D3B264B934AB8E7BDA7DCEEEDA
79AA39’

d) Build DO‘99’:
DO99 = ‘99029000’

e) Concatenate DO‘87’ and DO‘99’:
M = ‘871101D3B264B934AB8E7BDA7DCEEEDA
79AA3999029000’

f) Compute CMAC of M with KS_{mac}:
- Increment SSC:
SSC = ‘5E5D5B63C039F4BA’
- Concatenate padded SSC and M:
N = ‘00000000000000005E5D5B63C039F4BA
871101D3B264B934AB8E7BDA7DCEEEDA
79AA3999029000’
- Compute MAC:
CC = ‘C7F1FA5FD19FA478’

g) Build DO‘8E’:
DO8E = ‘8E08C7F1FA5FD19FA478’