EstEID v. 3.5

Estonian Electronic ID–card application specification

Document information	
Date of Creation	21.05.2013
Recipient	Police and Border Guard Board, Republic of Estonia
Publisher	AS Sertifitseerimiskeskus
Author	Trüb Baltic AS
Version	1.20

Introduction

The aim of this document is to specify the functions, the data content and the interface of the security chip along with the smart card application of the Estonian national public key infrastructure (EstEID).

The reader of this document is expected to be familiar with smart card and chip application related topics. Also the reader should have a fluent knowledge of the Estonian PKI.

Prerequisites to the Smart Card

The Estonian Electronic ID-card application is designed to run on an integrated circuit, a chip with Java Global Platform operating system and a contact based interface. It is assumed that the chip provides EEPROM capacity of 80 kilobytes or more.

The operating system and the chip shall be certified EAL4+ against Common Criteria protection profiles 002 and 035.

Document Scope

This specification describes the intended function and behaviour of the application being the default selectable one on the chip. The scope of this specification is strictly limited to the use within the frame of the Estonian PKI.

The reaction and behaviour of the application and the card to experiments, tests and attack attempts is out of scope of this document.

In this document the following topics are covered:

- Usage;
- Personalisation;
- Configuration; and
- Maintenance.

The usage of the chip application shall be fully covered by this document. The personalisation, configuration and maintenance can involve the specification of the chip OS, of the chip hardware and also the PKI policy.

Differentiation to previous versions of EstEID card application

ID	previous versions	version 3.5	comments
drop i	CMK3	-	Java card has its own card manager. No need to misuse the EstEID application to manage the potential loading of applications
drop ii	RootCA certificate	-	RootCA certificate is always checked online. There's no need to keep a place for offline checks.
drop iii	Passphrase (DESkeys)	-	Only PIN shall be used for authentication, signing and decryption function.
drop iv	Decryption function for signature keypair	-	The signature key and certificate shall be exclusively used for qualified digital signatures.
add α	-	Introduction of AID	An application identifier tag has been introduced.
add β	-	Extension of version ID	The version ID value has been extended from 2 to 3 bytes.
add γ	-	Option to support SHA-2	Beside SHA-1 also SHA-2 support has been added.
add δ	-	Enhanced management of life cycles	Running on a java card the life cycle of the card and the applet (the EstEID application) are clearly to be distinguished and managed separately.

Not released EstEID chip card application versions are marked by using the FCI (File Control Information). Release versions will not carry any FCI.

Document legend

 Tips

 Notes

 Important information or warnings

Operators

|| – Operator marking the concatenation operation.

+ – Operator marking the adding operation.

HEX – Marks that the number is presented in hexadecimal format.

DEC – Marks that the number is presented in decimal format.

BIT – Marks that the number is presented in binary format.

0x – Marks that the following number is presented in hexadecimal format.

Table of contents

1. Chip and card application

1.1. Answer to reset

Every contact card respond to reset with sequence of bytes called Answer To Reset [1](ATR).

The ATR gives information about the electrical communication protocol and the chip itself. It is mainly linked with the underlying integrated circuit (chip) and also the Java operating system on it. There is a block of historical bytes that can be used to indicate the purpose of the chip card.

The ATR can be different depending on if the reset is the first since power-up (Cold ATR) or not (Warm ATR). The meaningful info of ATR can be read from historical bytes of Cold ATR. In ANSI encoding they can be represented as “eID / PKI”. Together with a category indicator byte the historical bytes form a string of 10 bytes with the value "FE 65 49 44 20 2F 20 50 4B 49_{hex}".

The ATR can also be different when the chip, that carries the EstEID chip application, gets replaced.

The ATR cannot be used to identify the EstEID chip card application. The identification procedure is described in chapter 1.3 Identifying the card application.

1.2. Card application

EstEID card application is implemented on top of the JavaCard framework version 2.2.2. It enables operations which are required for PKI procedures. Card application has 3 different internal lifecycle states:

- Blank – Clean card application
- Personalised – Card application with cardholder personal information, PINs and CMKs.
- Live – Card application also has PKI objects.

Current specification concerns only the Live lifecycle state of card application.

1.3. Identifying the card application

EstEID application is the default selected application on the card. The card application can only be identified by selecting the card application with its AID and then identifying the card application version number from the card. The procedure for identifying the application version number is described in chapter 2.6.1 Reading EstEID application version. The card should return version 3.5.1 or higher.

Card application identification should be performed with following operations:

1) Application selection should be performed by executing following SELECT FILE command:

[1] See ISO 7816-3: Identification cards — Integrated circuit cards — Part 3: Cards with contacts — Electrical interface and transmission protocols

CLA	INS	P1	P2	Lc	Data (AID)
00_{hex}	$A4_{hex}$	04_{hex}	$0C_{hex}$	$0F_{hex}$	$D233000000457374454944207633 35_{hex}$

The card should respond with status word 9000_{hex}.

The application that gets selected is an application that is already selected by default on the card. Previous command is only a procedure for double checking that correct application is selected.

a2) Card application version identification should be performed as defined in chapter 2.6.1 Reading EstEID application version.

A proper and biunique identification of the card requires reading both the AID and also the version ID. The AID specifies the application and its features. The third digit of the version ID defines the implementation on a specific card environment, the Java card operating system and the silicon chip itself.

1.4. Card application file system structure

EstEID application derives file system attributes and functionalities from ISO 7816-4. The structure of the file system can be seen on the figure on the right.

Card application specific information is held in DF FID $EEEE_{hex}$. The heart data of card application can be considered the files which hold certificates (EF FID $AACE_{hex}$ & $DDCE_{hex}$) and card holder personal data (EF FID 5044_{hex}). The meaning of other files is to hold information about card application internal counters and references.

The reading operation, of files seen on the figure on the right, is specified in subchapters of chapter 2 Card application objects, their details and general operations.

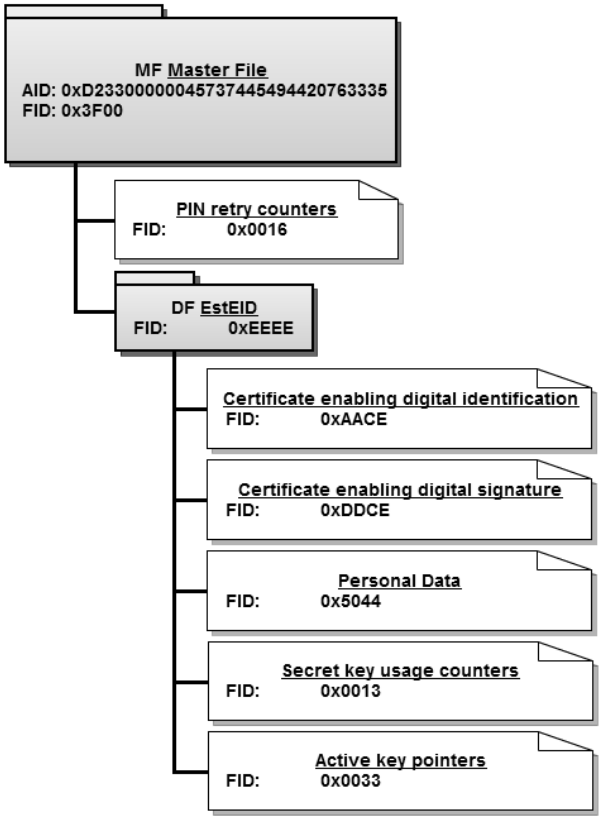

Figure 1-1 EstEID filesystem diagram

1.5. Objects in the card application

Table 1-1 The functions of EstEID security chip objects in the card application

Object	Function description
PIN1	The authorisation of the cardholder: 1) for getting access to the authentication key procedures. 2) for the execution of the following operations: a) the generation of new key pairs b) the loading of certificates
PIN2	For getting access to signature key.
PUK	The unblocking of PIN codes when they have been blocked after number of allowed consecutive incorrect entries.
Authentication certificate	The certificate for cardholder identification.
Signature certificate	The certificate calculating and checking the cardholder's electronic signature.
Authentication key pair (x2)	▪ Key pair that is actively used for cardholder authentication procedures. ▪ Optional idle key pair for a potential future replacement of active authentication key pair.
Signature key pair (x2)	▪ Key pair that is actively used for digital signing procedures. ▪ Optional idle key pair for a potential future replacement of active signature key pair.
Cardholder personal data file	Includes the cardholder's personal data.
CMK_PIN	3DES key which is used to secure the PIN code replacement procedure.
CMK_KEY	3DES key which is used to authorise the new key pair generation.
CMK_CERT	3DES key which is used to form the secure command series to load the user certificates.

1.6. Card application principles

Card application is realised on Java chip platform. It implements functionalities derived from ISO 7816-4 and ISO 7816-8 as much it is required for electronic identification and signing operations. Therefore many of the functionalities we know from ISO 7816 are not implemented. This kind of minimalistic implementation should make chip using easier and clearer for software developers.

Card application supports both T0 and T1 transport protocol.

DES cryptographic operations are performed using ISO 9797-1 padding method 2.

Card application uses for PKI operations 2048 bit RSA with method PKCS#1 version 1.5.

Card application has three different use cases for different card application interfaces, which are inside the card implemented as environments:

- Public environment – Reading card application data objects and changing PIN/PUK codes.
- PKI environment – For performing PKI operations in card application.
- Card application authority environment – Replacing PIN/PUK and certificate objects. Generating new key pairs for cardholder.

All operations with RSA key pairs are authorised by verifying the cardholder with PIN1 or PIN2.

2. Card application objects, their details and general operations

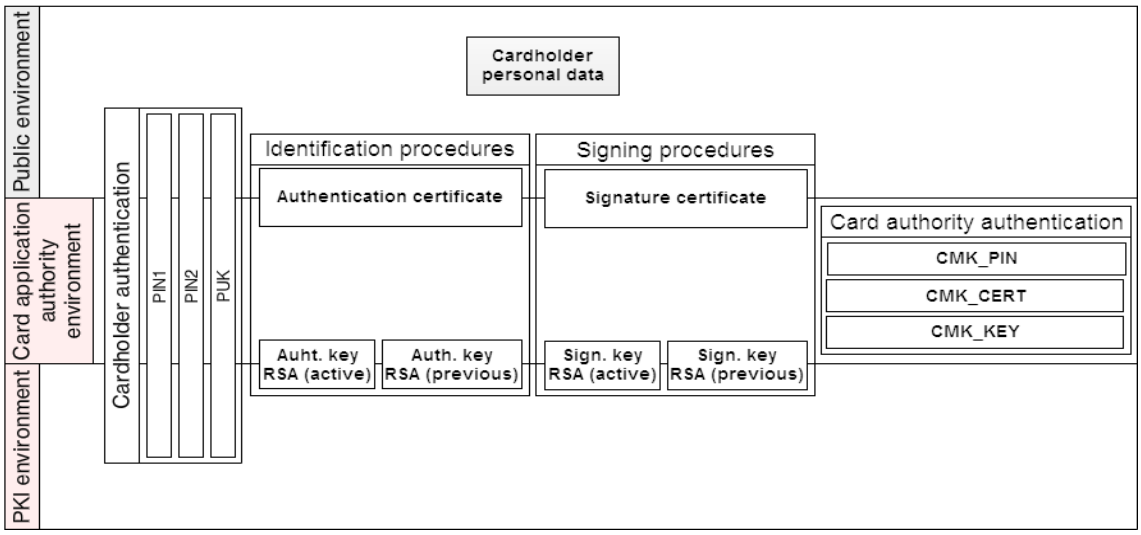

Figure 2-1 Card application objects

Card application contains objects which are used for different procedures. These procedures can be for public, PKI or card application authority environment use. Figure above shows in which environment objects are used. Following sub-chapters describe the objects and their operations.

2.1. Personal data file

EstEID card has on the card personal information of the cardholder. The same information is also included digitally on the card application (except photo and signature images). Cardholder personal data is held in file with File Identifier (FID) 5044_{hex} or 'PD' as ASCII.

The cardholder personal data file includes the records given below. It is a variable-length formatted file where every record in it can differ in length. Records which have marked with maximum length can contain data with various lengths. All the data in these records are coded according to ANSI code page 1252. The records that have the maximum length marked in Table 2-1 Personal Data file contents, could have various length of data up to maximum marked. If given records do not contain meaningful data they will be filled with a placeholder, one space character (20_{hex} as ANSI).

Table 2-1 Personal Data file contents

Record no.	Content	Length or maximum length
1	Surname	max 28_{dec} bytes

Table 2-1 Personal Data file contents		
Record no.	Content	Length or maximum length
2	First name line 1	max 15_{dec} bytes
3	First name line 2	max 15_{dec} bytes
4	Sex Values: 'M' – Male 'N' – Female	1_{dec} bytes
5	Nationality (3 letters) According to ISO 3166-1 alpha-3.	3_{dec} bytes
6	Birth date (dd.mm.yyyy)	10_{dec} bytes
7	Personal identification code	11_{dec} bytes
8	Document number	9_{dec} bytes
9	Expiry date (dd.mm.yyyy)	10_{dec} bytes
10	Place of birth	max 35_{dec} bytes
11	Date of issuance (dd.mm.yyyy)	10_{dec} bytes
12	Type of residence permit	max 50_{dec} bytes
13	Notes line 1	max 50_{dec} bytes
14	Notes line 2	max 50_{dec} bytes
15	Notes line 3	max 50_{dec} bytes
16	Notes line 4	max 50_{dec} bytes

2.1.1. Reading contents of Personal Data file

To read the personal information of the cardholder following operations must be performed:

1) Always is good to select the root file of the card application called Master File (MF) with command SELECT FILE. This is only if you do not know currently selected file on the card application. After card reset or insertion to card reader this file is initially selected and has no actual need to perform this operation.

CLA	INS	P1	P2	Le
00_{hex}	$A4_{hex}$	00_{hex}	$0C_{hex}$	00_{hex}

2) Select directory file $EEEE_{hex}$ by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
00_{hex}	$A4_{hex}$	01_{hex}	04_{hex}	02_{hex}	$EEEE_{hex}$

3) Select Personal Data file with FID 5044_{hex} by using command SELECT FILE once again.

CLA	INS	P1	P2	Lc	Data (FID)
00_{hex}	$A4_{hex}$	02_{hex}	04_{hex}	02_{hex}	5044_{hex}

After given command the Personal Data file is selected and it is possible to read the contents of the file.

4) Records that the Personal Data file contains are specified in table Personal Data file contents. For this example let's read record no. 7 „Personal identification code", with command READ RECORD.

For current example let the record no. 7 value be '47101010033'.

CLA	INS	P1 (record no)	P2	Le
00_{hex}	$B2_{hex}$	07_{hex}	04_{hex}	00_{hex}

In case Le is set to 00_{hex} the length of the record is not known and as a result card application will respond with the data of the record in R-APDU. The Le field can as well have the exact length value for record or value larger than record. For the last case the data from the next record(s) will be also returned in R-APDU.

For successful read card application responds with R-APDU:

Data	SW1	SW2
$34373130313031303033333_{hex}$	90_{hex}	00_{hex}

It must be considered that if protocol T0 is used and Le has value 00_{hex} or it is absent, then the card will respond with status $61XX_{hex}$. How to operate in given situation is described in chapter 7.1 Card possible response in case of protocol T0.

2.2. PIN1, PIN2 and PUK code

Verification by card application can be done with 3 different methods. These methods are called PIN1, PIN2 and PUK code. All these methods have their own card application operational purpose. Details about these verification methods are specified in the table below.

Table 2-2 PIN1, PIN2 and PUK

Method	Length in bytes			Definition
	Min	Max	Initial	
PIN1	4_{dec}	12_{dec}	4_{dec}	For operations with Authentication key.
PIN2	5_{dec}	12_{dec}	5_{dec}	For operations with Digital Signature key.
PUK	8_{dec}	12_{dec}	8_{dec}	For unblocking PIN1 or PIN2 codes.

Card application does not allow using PIN and PUK codes with lengths which are outside of the limits range. Initial lengths of the application are given in the table above. The length of PIN and PUK codes

may vary if cardholder has changed the default value. The host applications communicating with the card application must support all lengths of PIN and PUK codes marked in the table above.

All verification codes consist of ASCII digits from '0' to '9'. As well, all codes must be transmitted to card application in ASCII format.

Card application supports as well all other ASCII characters for verification. However other characters than digits are not supported by the use cases.

PIN and PUK codes cannot be read from the card application. These codes can only be verified by the card application.

The cardholder authentication with PIN1, PIN2 or PUK is conducted by the command VERIFY. The given operation may be executed without restrictions.

All codes have separate retry counter which are initially and reset to value 3 after successful verification. Every unsuccessful verification results in a decrease of the corresponding code's retry counter. If retry counter has reached to 0, given verification method is blocked. Blocked PIN codes can be unblocked using RESET RETRY COUNTER command. However if PUK code retry counter reaches to 0, card application PIN and PUK codes can be unblocked an changed only by the card application authority.

2.2.1. Verify PIN1, PIN2 or PUK code

Cardholder authentication is done by verifying PIN and PUK codes. After successful verification procedure application general operations will be accessible.

It is strongly advised to perform cardholder authentication with PIN or PUK verification on external pin pad device.

Keep in mind that unsuccessful operation of command VERIFY decreases pin retry counter value by one.

For PIN and PUK examples below let's use following PIN and PUK code values:

- PIN1 – '1234' as ASCII or 31323334_{hex}
- PIN2 – '12345' as ASCII or 3132333435_{hex}
- PUK – '12345678' as ASCII or 3132333435363738_{hex}

To verify PIN1 it is needed to execute the following command VERIFY:

CLA	INS	P1	P2	Lc	Data (PIN1 as ASCII)
00_{hex}	20_{hex}	00_{hex}	01_{hex}	04_{hex}	31323334_{hex}

To verify PIN2 it is needed to execute the following command VERIFY:

CLA	INS	P1	P2	Lc	Data (PIN2 as ASCII)
00_{hex}	20_{hex}	00_{hex}	02_{hex}	05_{hex}	3132333435_{hex}

To verify PUK it is needed to execute the following command VERIFY:

CLA	INS	P1	P2	Lc	Data (PUK as ASCII)
00_{hex}	20_{hex}	00_{hex}	00_{hex}	08_{hex}	3132333435363738_{hex}

For successful operation card application will respond with status 9000_{hex} and verification in the application is flagged as verified. For other possible R-APDU statuses look chapter 7.2.7 VERIFY.

2.2.2. Changing PIN1, PIN2 or PUK code

The values of PIN1, PIN2 and PUK codes can be replaced with the command CHANGE REFERENCE DATA if given code is not blocked.

The values of PIN1 and PIN2 codes can be replaced also with PUK verification using command RESET RETRY COUNTER. For examples for given command see chapter 2.2.3 Unblocking PIN1 or PIN2 code.

The values of previous PIN or PUK code and new PIN or PUK code have to be different when changing the code's value.

As a result for successful operation of commands above the code will be replaced and retry counter reset for given operable code.

For given example let the PIN and PUK codes have the same values as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

To replace PIN1 code with '54321' it is needed to executed the following command CHANGE REFERENCE DATA:

CLA	INS	P1	P2	Lc	Data
00_{hex}	24_{hex}	00_{hex}	01_{hex}	09_{hex}	313233343534333231_{hex}

To replace PIN2 code with '654321' it is needed to executed the following command CHANGE REFERENCE DATA:

CLA	INS	P1	P2	Lc	Data
00_{hex}	24_{hex}	00_{hex}	02_{hex}	$0B_{hex}$	$31323334353635343333231_{hex}$

To replace PUK code with '987654321' it is needed to executed the following command CHANGE REFERENCE DATA:

CLA	INS	P1	P2	Lc	Data
00_{hex}	24_{hex}	00_{hex}	00_{hex}	11_{hex}	$313233343536373839383736353433323 1_{hex}$

For successful operation card application will respond with status 9000_{hex}. For other possible R-APDU statuses look chapter 7.2.8 CHANGE REFERENCE DATA.

2.2.3. Unblocking PIN1 or PIN2 code

When PIN codes retry counter values has decremented to value 0 and get blocked, it is possible to unblock them by resetting the retry counter. This operation is possible with command RESET RETRY COUNTER.

For given example let the PUK code have the same values as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

1) To unblock PIN codes with PUK code verification it is needed to do following:
First it is needed to verify the PUK code, with executing the following command VERIFY:

CLA	INS	P1	P2	Lc	Data (PUK as ASCII)
00_{hex}	20_{hex}	00_{hex}	00_{hex}	08_{hex}	3132333435363738_{hex}

After given command is executed and successful response returned it is possible to reset retry counter of PIN codes.

PIN1 and PIN2 can only be unblocked if they are in blocked state.

a) To unblock and reset retry counter of PIN1 with pre-verified PUK it is needed to executed the following command RESET RETRY COUNTER:

CLA	INS	P1	P2	Le
00_{hex}	$2C_{hex}$	03_{hex}	01_{hex}	00_{hex}

b) Or to unblock and reset retry counter of PIN2 with pre-verified PUK it is needed to executed the following command RESET RETRY COUNTER:

CLA	INS	P1	P2	Le
00_{hex}	$2C_{hex}$	03_{hex}	02_{hex}	00_{hex}

After successful execution of previous commands PIN codes get unblocked and retry counters reset.

2) To unblock by changing the PIN codes with PUK code verification it is needed to do following:

Keep in mind that unsuccessful operation of following command RESET RETRY COUNTER decreases PUK retry counter value by one.

The values of previous internal PIN code and new PIN code can have the same values.

a) To unblock and reset retry counter of PIN1 with pre-verified PUK it is needed to executed the following command RESET RETRY COUNTER:

CLA	INS	P1	P2	Lc	Data (PUK || new PIN1 '4321')
00_{hex}	$2C_{hex}$	00_{hex}	01_{hex}	$0C_{hex}$	3132333435363738_{hex} || 34333221_{hex}

b) Or to unblock and reset retry counter of PIN2 with pre-verified PUK it is needed to executed the following command RESET RETRY COUNTER:

CLA	INS	P1	P2	Lc	Data (PUK || new PIN2 '54321')
00_{hex}	$2C_{hex}$	00_{hex}	02_{hex}	$0D_{hex}$	3132333435363738_{hex} || 3534333221_{hex}

2.2.4. Reading PIN1, PIN2, or PUK code counter

The current retry counter values of PIN and PUK codes can be read from the card application from PIN retry counters file with FID 0016_{hex}. It is variable-length formatted file containing 3 records in following corresponding sequence for PIN1, PIN2 and PUK code.

Table 2-3 EF FID 0016_{hex} contents

Record no.	Verification method	Description
1	PIN1	TLV data containing maximum and current retry counter value and unblock reference.
2	PIN2	TLV data containing maximum and current retry counter value and unblock reference.
3	PUK	TLV data containing maximum and current retry counter value.

Before the PIN retry counters file can be read it must be selected:

a) First it must be sure that the MF is selected. Execute MF selection with command SELECT FILE:

CLA	INS	P1	P2	Le
00_{hex}	$A4_{hex}$	00_{hex}	$0C_{hex}$	00_{hex}

b) Secondly execute selection for EF FID 0016_{hex} with command SELECT FILE:

CLA	INS	P1	P2	Lc	Data
00_{hex}	$A4_{hex}$	02_{hex}	$0C_{hex}$	02_{hex}	0016_{hex}

When the EF FID 0016 is selected there can be performed reading operations with command READ RECORD to get the PIN and PUK codes counters:

a) To read retry counter of PIN1 it is needed to executed the following command READ RECORD:

CLA	INS	P1 (record no)	P2	Le
00_{hex}	$B2_{hex}$	01_{hex}	04_{hex}	00_{hex}

For successful reading operation chip responds with following R-APDU where X marks remaining tries for PIN1:

Data						SW1	SW2
Max. tries		Remaining tries		Unblock reference			
Tag || Length	Value	Tag || Length	Value	Tag || Length	Value		
80_{hex} || 01_{hex}	03_{hex}	90_{hex} || 01_{hex}	$0X_{hex}$	83_{hex} || 02_{hex}	0000_{hex}	90_{hex}	00_{hex}

b) To read retry counter of PIN2 it is needed to executed the following command READ RECORD:

CLA	INS	P1 (record no)	P2	Le
00_{hex}	$B2_{hex}$	02_{hex}	04_{hex}	00_{hex}

For successful reading operation chip responds with following R-APDU where X marks remaining tries for PIN2:

Data						SW1	SW2
Max. tries		Remaining tries		Unblock reference			
Tag || Length	Value	Tag || Length	Value	Tag || Length	Value		
80_{hex} || 01_{hex}	03_{hex}	90_{hex} || 01_{hex}	$0X_{hex}$	83_{hex} || 02_{hex}	0000_{hex}	90_{hex}	00_{hex}

c) To read retry counter of PUK it is needed to executed the following command READ RECORD:

CLA	INS	P1 (record no)	P2	Le
00_{hex}	$B2_{hex}$	03_{hex}	04_{hex}	00_{hex}

For successful reading operation chip responds with following R-APDU where X marks remaining tries for PUK:

Data				SW1	SW2
Max. tries		Remaining tries			
Tag || Length	Value	Tag || Length	Value		
80_{hex} || 01_{hex}	03_{hex}	90_{hex} || 01_{hex}	$0X_{hex}$	90_{hex}	00_{hex}

2.3. Certificates

EstEID application contains 2 certificates in the card:

- Certificate for cardholder authentication operations.
- Certificate for cardholder digital signing operations.

Certificate files in the card application are located in DF with FID $EEEE_{hex}$. Authentication certificate file has FID $AACE_{hex}$ and digital signing certificate file has FID $DDCE_{hex}$. Certificate files are transparent files and can be read with command READ BINARY. In the card application certificate files have fixed length memory allocated as there may be a need to write new certificates which have slightly different length. Certificate files are formatted as ASN.1 DER.

To fit the certificate into the file, certificate data is padded according to ISO 9797-1 padding method 2. The padding must be appended to a whole number of bytes long data:

600_{hex} bytes	
Certificate bytes	Padding start indicator || Padding zero bytes until the end of file
3082_{hex} … XX_{hex}	80_{hex} || 00_{hex} ... 00_{hex}

Padding example for certificate with length $5F9_{hex}$ bytes:

Certificate data ($5F9_{hex}$ bytes)	Padding (7_{dec} bytes)
3082_{hex} … XX_{hex}	80000000000000_{hex}

2.3.1. Reading certificate files

Both certificate files from the card application can be read in the similar way. There are 2 ways to read the entire certificate file from the card. The reading of the file can be done by using command READ BINARY. The easiest way to read the file is to use extended length APDU for it. Another way is to use sequence of READ BINARY command by increasing the file reading offset for every next command until the whole file data is returned.

Certificate files are located in DF $EEEE_{hex}$. First navigate to given directory by:

a) selecting the application MF with command SELECT FILE:

CLA	INS	P1	P2	Le
00_{hex}	$A4_{hex}$	00_{hex}	$0C_{hex}$	00_{hex}

b) and selecting directory file $EEEE_{hex}$ by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
00_{hex}	$A4_{hex}$	01_{hex}	04_{hex}	02_{hex}	$EEEE_{hex}$

Next step is to select desired certificate file for reading by using command SELECT FILE once again:

a) For selecting Authentication certificate file use:

CLA	INS	P1	P2	Lc	Data (FID)
00_{hex}	$A4_{hex}$	02_{hex}	04_{hex}	02_{hex}	$AACE_{hex}$

a) For selecting Digital Signature certificate file use:

CLA	INS	P1	P2	Lc	Data (FID)
00_{hex}	$A4_{hex}$	02_{hex}	04_{hex}	02_{hex}	$DDCE_{hex}$

Now certificate file is ready for reading operations:

This are just examples showing one possible way of doing it. Please refer to ISO7816-4 to exploit other methods and solutions.

1) First method for reading the file is using extended APDU. For this it is needed to send command READ BINARY in following format:

CLA	INS	P1	P2	Ext. APDU indicator	Le (length of certificate file)
00_{hex}	$B0_{hex}$	00_{hex}	00_{hex}	00_{hex}	0600_{hex}

2) Second method for reading the file is by sending multiple READ BINARY commands by changing the file reading offset for every next command until the whole file has been read. Data of the result has to be concatenated together.

Keep in mind that certificate file has length of 600_{hex} bytes but certificate file data is actually shorter in length. The actual length of the certificate can be derived from the first response of the READ BINARY command because certificate data is in ASN.1 DER encoded format. 3rd and 4th byte of the 1st response hold the actual length of certificate data.

Actual length of the file can be calculated as 4_{dec} + (3rd byte || 4th byte)$_{hex}$ from the response bytes.

a) First READ BINARY command in sequence:

CLA	INS	P1	P2	Le
00_{hex}	$B0_{hex}$	00_{hex}	00_{hex}	00_{hex} (actually expects 256_{dec})

Command response:

Data			SW1	SW2
Tag	Length	Value		
30_{hex}	82_{hex} || $04A5_{hex}$	252_{dec} bytes of data	90_{hex}	00_{hex}

3rd and 4th bytes in 1st response have values 04_{hex} and $A5_{hex}$. From this we can calculate the length of certificate data:

$4_{dec} + (04_{hex} \,||\, A5_{hex}) = 04A9_{hex}$

256_{dec} bytes have already been read. There are $03A9_{hex}$ bytes still waiting to be read.

b) 2nd, 3rd and 4th time read another 256_{dec} bytes from the chip with READ BINARY command by increasing file reading offset every time with 256_{dec}:

CLA	INS	P1	P2	Le
00_{hex}	$B0_{hex}$	01_{hex}	00_{hex}	00_{hex} (actually expects 256_{dec})

CLA	INS	P1	P2	Le
00_{hex}	$B0_{hex}$	02_{hex}	00_{hex}	00_{hex} (actually expects 256_{dec})

CLA	INS	P1	P2	Le
00_{hex}	$B0_{hex}$	03_{hex}	00_{hex}	00_{hex} (actually expects 256_{dec})

c) For 5th, the last reading, there are $A9_{hex}$ bytes of certificate to read from the card from file offset 400_{hex}:

CLA	INS	P1	P2	Le
00_{hex}	$B0_{hex}$	04_{hex}	00_{hex}	$A9_{hex}$

For other possible response messages see chapter 7.2.3 READ BINARY.

2.4. Cardholder secret keys

EstEID application has two PKI key pairs that enable cardholder authentication and digital signing operations. The length of the key is 2048_{dec} bits. The public exponent for the keys is 40000081_{hex} if supported by the platform or 00010001_{hex} if the underlying platform is not capable of supporting arbitrary exponents. Initial remaining use counter for every key is $FFFFFF_{hex}$. This counter will be decreased by one after every successful operation executed with the key. After the use counter has decreased to 0_{dec}, the key can no longer be used.

Cardholder key pair cannot be read from the card. Key pairs are generated in the card on two occasions:

- Card application personalisation
- New key pair generation procedure

Information about the key can be read from EF with FID 0013_{hex} in DF with FID $EEEE_{hex}$. Given EF is variable-length formatted file and has 4 records containing information about key pairs as specified in following table:

Table 2-4 EF FID 0013_{hex} key records and references of secret keys		
Key description	Record no.	Reference no. KID || KV
Signature key (actively used)	01_{hex}	0100_{hex}
Signature key (can be absent)	02_{hex}	0200_{hex}
Authentication key (actively used)	03_{hex}	1100_{hex}
Authentication key (can be absent)	04_{hex}	1200_{hex}

The active keys could be changed during the life of the card and should be checked from EF FID 0033_{hex} with procedure specified in chapter 2.4.2 Reading secret key information.

Table 2-5 EF FID 0013_{hex} key record description			
Bytes		Data	Description
Position	Count		
0 .. 1	2	8304_{hex}	Tag & Length for key reference
2 .. 3	2	$XXXX_{hex}$	Key reference value
4 .. 5	2	0000_{hex}	Fixed value
6 .. 7	1	$C002_{hex}$	Tag & Length for public key data
8	1	81_{hex} initialised key	Public key indicator
		00_{hex} not initialised key	No key attached
9	1	FF_{hex} initialised key	Public key size 2048_{bits}
		00_{hex} not initialised key	No key attached
10 .. 11	2	9103_{hex}	Tag & Length for key use counter
12 .. 14	3	$XXXXXX_{hex}$	Key use counter value

The information about the currently active key pair can be read from the EF with FID 0033_{hex} in DF with FID $EEEE_{hex}$. Given EF is variable-length formatted file, but contains only 1 record. The structure of the only record of the file is specified in the following table:

Table 2-6 EF FID 0033_{hex} key record description			
Bytes		Data	Description
Position	Count		
0	1	00_{hex}	Security environment (SE) indicator. Applies to all SEs.
1 .. 2	2	$A408_{hex}$	Tag & Length of active authentication key info

Table 2-6 EF FID 0033_{hex} key record description			
Bytes		Data	Description
Position	Count		
3 .. 4	2	9501_{hex}	Tag & Length of authentication key usage qualifier
5	1	40_{hex}	Value for authentication key usage qualifier.
6 .. 7	2	8303_{hex}	Tag & Length of authentication key accessing info
8	1	80_{hex}	Value for authentication key search type (KST)
9 .. 10	2	$XXXX_{hex}$	Value for active authentication key reference: Key ID (KID)
11 .. 12	2	$B608_{hex}$	Tag & Length of active signature key info
13 .. 14	2	9501_{hex}	Tag & Length of signature key usage qualifier
15	1	40_{hex}	Value for signature key usage qualifier
16 .. 17	2	8303_{hex}	Tag & Length of signature key accessing info
18	1	80_{hex}	Value for signature key search type (KST).
19 .. 20	2	$XXXX_{hex}$	Value for active signature key reference: Key ID (KID)

Value 40_{hex} of usage qualifier returned in response is specified in ISO 7816-9. It indicates the given key to be used for data authentication, data confidentiality, internal and mutual authentication.

Value 80_{hex} of key search type indicates that the key is usable only in specific DF.

2.4.1. Reading public key of cardholder secret key

There is only one way to obtain the public key from the card application, which is reading the certificate file and divide the public key from it.

This specification does not specify the procedure how to use or derive the public key from the certificate. This functionality must be searched and used by some cryptography API.

2.4.2. Reading secret key information

The secret key information can be obtained by reading records from EF with FID 0013_{hex}. The structure of given EF is specified in table EF FID 0013hex key records and references of secret keys.

EF FID 0013_{hex} file is located in DF FID $EEEE_{hex}$. First navigate to given directory by:

a) selecting the application MF with command SELECT FILE:

CLA	INS	P1	P2	Le
00_{hex}	$A4_{hex}$	00_{hex}	$0C_{hex}$	00_{hex}

b) and selecting directory file $EEEE_{hex}$ by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
00_{hex}	$A4_{hex}$	01_{hex}	04_{hex}	02_{hex}	$EEEE_{hex}$

Next step is to select EF FID 0013_{hex} for reading by using command SELECT FILE once again:

CLA	INS	P1	P2	Lc	Data (FID)
00_{hex}	$A4_{hex}$	02_{hex}	04_{hex}	02_{hex}	0013_{hex}

Now EF is ready for reading operations:

1) To read information for actively in use signature key from record no 1:

CLA	INS	P1 (record no)	P2	Le
00_{hex}	$B2_{hex}$	01_{hex}	04_{hex}	00_{hex}

2) To read information for secondary signature key from record no 2:

CLA	INS	P1 (record no)	P2	Le
00_{hex}	$B2_{hex}$	02_{hex}	04_{hex}	00_{hex}

3) To read information for actively in use authentication key from record no 3:

CLA	INS	P1 (record no)	P2	Le
00_{hex}	$B2_{hex}$	03_{hex}	04_{hex}	00_{hex}

4) To read information for secondary authentication key from record no 4:

CLA	INS	P1 (record no)	P2	Le
00_{hex}	$B2_{hex}$	04_{hex}	04_{hex}	00_{hex}

For successful operation, card application will respond to these commands in data field as described in table EF FID 0013hex key record description. For other possible R-APDUs see chapter 7.2.2 READ RECORD.

2.4.3. Reading key references for active keys

Key references for active keys can be read from EF FID 0033_{hex} in record no 1.

Before given EF can be read it is needed to be selected. First navigate to given directory by:

1) selecting the application MF with command SELECT FILE:

CLA	INS	P1	P2	Le
00_{hex}	$A4_{hex}$	00_{hex}	$0C_{hex}$	00_{hex}

2) and selecting directory file $EEEE_{hex}$ by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
00_{hex}	$A4_{hex}$	01_{hex}	04_{hex}	02_{hex}	$EEEE_{hex}$

Next step is to select EF FID 0033_{hex} for reading by using command SELECT FILE once again:

CLA	INS	P1	P2	Lc	Data (FID)
00_{hex}	$A4_{hex}$	02_{hex}	04_{hex}	02_{hex}	0033_{hex}

Now EF is ready for reading operations. Read record no 1 from selected file:

CLA	INS	P1 (record no)	P2	Le
00_{hex}	$B2_{hex}$	01_{hex}	04_{hex}	00_{hex}

For successful operation, card application will respond to these commands in data field as described in table EF FID 0033hex key record description. For other possible R-APDUs see chapter 7.2.2 READ RECORD.

The references for active signature and authentication keys EF FID 0033_{hex} may be changed in the course of a new key pair generation operation (see chapter 4.4 New RSA key pair generation.

2.5. Card application management keys: CMK_PIN, CMK_CERT & CMK_KEY

The security of EstEID card application management, for after personalisation process, is based on three secret 3DES keys:

- CMK_PIN – Cardholder authentication objects replacement key. Used to get authorisation for executing command SECURE REPLACE PINS.
- CMK_CERT – Cardholder certificate objects replacement key. Used to get authorisation for executing command SECURE REPLACE CERTIFICATE.
- CMK_KEY – Cardholder new secret keys generation key. Used to get authorisation for executing command SECURE GENERATE KEY.

Each of these keys is derived from the corresponding master 3DES key by the card personalisation service provider in a secure environment. The given derived keys are loaded on the card application during card application personalisation phase and cannot be read from the card or modified afterwards.

The example keys and cardholder ID used in the present document are as follows:

- Master CMK_PIN = $A1A1A1A1A1A1A1A1A2A2A2A2A2A2A2A2_{hex}$
- Master CMK_CERT = $C1C1C1C1C1C1C1C1C2C2C2C2C2C2C2C2_{hex}$
- Master CMK_KEY = $B1B1B1B1B1B1B1B1B2B2B2B2B2B2B2B2_{hex}$
- Cardholder ID = “47101010033”

2.5.1. Deriving card application management keys

Each CMK key on card application is derived from corresponding master CMK key which are maintained by the card centre. Current chapter describes the procedure to derive CMK keys.

The CMK derivation procedure is as follows:

1) Calculate the SHA-1 hash of the cardholder's personal identification number.
2) Take 16 leftmost bytes of calculated SHA-1.
3) Encrypt these bytes with master CMK key, which correspond with desired one, by using 3DES algorithm, in CBC mode with IV value 0000000000000000_{hex}.
4) Set the MSB of each byte off so that each one would have odd value

For following example let the cardholder identification number be '01234567890'. For example calculation of the CMK there will be used master CMK given in chapter 2.5 Card application management keys: CMK_PIN, CMK_CERT & CMK_.

Example of calculating CMK_PIN:

1) SHA-1 hash for given cardholder identification number is:
   $7ED10E4A589C87F9E6A85C22E4B0C38ECF5F5059_{hex}$
2) 16 leftmost bytes of given hash:
   $7ED10E4A589C87F9E6A85C22E4B0C38E_{hex}$
3) Encrypt bytes above with master CMK_PIN in 3DEC CMC mode with IV value 0000000000000000_{hex}:
   $41F9AE3548536F19B93FED4EF890C93B_{hex}$
4) Set the MSB bit of each byte off so that each one would have odd value:

   Bytes as bit array:

   ```
   01000001 11111001 10101110 00110101 01001000 01010011 01101111
   00011001 10111001 00111111 11101101 01001110 11111000 10010000
   11001001 00111011
   ```

   Bytes as bit array with MSB set off:

   ```
   01000000 11111000 10101110 00110100 01001000 01010010 01101110
   00011000 10111000 00111110 11101100 01001110 11111000 10010000
   11001000 00111010
   ```

   Bytes with MSB set off:

   CMK_PIN = **40F8**AE**34**48**526E18B83EEC**4EF890**C83A**$_{hex}$

As well derivations for CMK_CERT and CMK_KEY:

CMK_CERT = $3A8ABC9A981E28AAB20C961464284262_{hex}$

CMK_KEY = $88FA5C9AB082D096AA125EBE70DEFC86_{hex}$

2.6. Miscellaneous information

From the application there can be read information that is not directly related to application data objects. There are 3 types of information available to read:

- EstEID card application version
- CPLC data
- Chip available memory

This information can be accessed with executing command GET DATA. Given information is not related to application file system and does not need navigation to data files.

For following commands Le field is optional if T1 protocol is used.

For protocol T0 Le field must be set. Otherwise the card will respond with $61XX_{hex}$ or $6CXX_{hex}$. How to behave on given cases is specified in chapter 7.1 Card possible response in case of protocol T0.

2.6.1. Reading EstEID application version

To read EstEID card application version it is needed to executed the following command GET DATA:

CLA	INS	P1	P2	Le
00_{hex}	CA_{hex}	01_{hex}	00_{hex}	03_{hex}

For given example let the EstEID application version be 3.5.1

Card application will respond with following R-APDU containing data for EstEID application version:

Data	SW1	SW2
030501_{hex}	90_{hex}	00_{hex}

2.6.2. Reading CPLC data

To read CPLC data it is needed to execute the following command GET DATA:

CLA	INS	P1	P2	Le
00_{hex}	CA_{hex}	02_{hex}	00_{hex}	$2A_{hex}$

Card application will respond with following R-APDU containing data for EstEID application version:

Data	SW1	SW2
42_{dec} bytes of CPLC data	90_{hex}	00_{hex}

The contents of the CPLC data sequence is specified in the following table:

Table 2-7 Card Production Life Cycle data

CPLC field	Length
IC Fabricator	2_{dec}
IC Type	2_{dec}
Operating System Identifier	2_{dec}

Table 2-7 Card Production Life Cycle data	
CPLC field	Length
Operating System release date	2_{dec}
Operating System release level	2_{dec}
IC Fabrication Date	2_{dec}
IC Serial Number	4_{dec}
IC Batch Identifier	2_{dec}
IC Module Fabricator	2_{dec}
IC Module Packaging Date	2_{dec}
ICC Manufacturer	2_{dec}
IC Embedding Date	2_{dec}
IC Pre-personaliser	2_{dec}
IC Pre-personalisation Date	2_{dec}
IC Pre-personalisation Equipment Identifier	4_{dec}
IC Personaliser	2_{dec}
IC Personalisation Date	2_{dec}
IC Personalisation Equipment Identifier	4_{dec}

2.6.3. Reading data for available memory on chip

To read free memory of chip it is needed to execute the following command GET DATA:

CLA	INS	P1	P2	Le
00_{hex}	CA_{hex}	03_{hex}	00_{hex}	06_{hex}

Card application will respond with following R-APDU containing data for available memory on chip:

Data (Available memory...)			SW1	SW2
… that is freed on application deselecting or reset.	.. that is freed on application reset.	… that is for persistent use.		
$XXXX_{hex}$	$YYYY_{hex}$	$ZZZZ_{hex}$	90_{hex}	00_{hex}

If there is more free memory available than $FFFF_{hex}$ bytes then given memory value will be returned as $FFFF_{hex}$.

Command given in current chapter responds with sequence of results of JavaCard v2.2.2 API command JCSystem.getAvailableMemory. The sizte of the free memory for three different types of memory in the card are returned. These commands are executed with attributes in following order:

- MEMORY_TYPE_TRANSIENT_DESELECT
- MEMORY_TYPE_TRANSIENT_RESET
- MEMORY_TYPE_PERSISTENT

Look JavaCard v2.2.2 API for better overview.

3. Card application general operations

Card application enables PKI operations. Following sub-chapters describe how to perform operations regarding to cardholder authentication, digital signing and session key decryption.

3.1. Calculating the response for TLS challenge

Calculating the response for TLS challenge is executing the RSA encryption operation with private key. Encrypted TLS challenge is formatted according to PKCS#1 version 1.5 block type 1. For authorising cardholder for given operation it is needed to authenticate the user with PIN1.

For given example let the PIN1 code have the same value as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

This are just examples showing one possible way of doing it. Please refer to ISO7816-4 to exploit other methods and solutions.

1) In order to calculate the response by card application it is needed to execute following operations: Before currently selected key reference can be set it is needed to select DF FID $EEEE_{hex}$. First navigate to given directory by:
   a) selecting the application MF with command SELECT FILE:

CLA	INS	P1	P2	Le
00_{hex}	$A4_{hex}$	00_{hex}	$0C_{hex}$	00_{hex}

   b) and selecting directory file $EEEE_{hex}$ by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
00_{hex}	$A4_{hex}$	01_{hex}	04_{hex}	02_{hex}	$EEEE_{hex}$

2) Select security environment for TLS operations by executing following command MANAGE SECURITY ENVIRONMENT:

CLA	INS	P1	P2	Le
00_{hex}	22_{hex}	$F3_{hex}$	01_{hex}	00_{hex}

If the card has already been set to use decryption security environment after last card reset, then it is not required to execute given command.

3) Set the active authentication key reference for INTERNAL AUTHENTICATE command execution by executing command MANAGE SECURITY ENVIRONMENT:

CLA	INS	P1	P2	Lc	Data (TL of TLV || KST || Key reference)
00_{hex}	22_{hex}	41_{hex}	$B8_{hex}$	05_{hex}	8303_{hex} || 80_{hex} || 1100_{hex}

If the card hasn't been set to use secondary authentication key pair after last card reset, then it is not required to execute given command.

Key reference values as specified in Table 3-1 EF FID 0013hex key records and references of secret keys active keys should be used. The proper way to receive currently active authentication keys reference is described in chapter 2.4.3 Reading key references for active keys.

By default it is not necessary to execute the security environment and key reference commands in case of calculating response to TLS challenge.

4) Authenticate cardholder with PIN1 with executing command VERIFY to authorise executing command INTERNAL AUTHENTICATE:

CLA	INS	P1	P2	Lc	Data (PIN1 as ASCII)
00_{hex}	20_{hex}	00_{hex}	01_{hex}	04_{hex}	31323334_{hex}

5) Calculate the response for TLS challenge by executing command INTERNAL AUTHENTICATE:

CLA	INS	P1	P2	Lc	Data	Le
00_{hex}	88_{hex}	00_{hex}	00_{hex}	TLS challenge length	TLS challenge	00_{hex}

Card application responds with bytes which are the cryptogram of PKCS#1 ver. 1.5 block type 1 enveloped TLS challenge. The length of the response varies and is related to the length of provided TLS challenge. The encryption is done with RSA authentication private key.

According to TLS standard, the length of the challenge is 24_{hex} bytes. However, card application also enables the calculation of responses to challenges of other length. For the maximum length of TLS challenge it must be taken into account that it is being formatted according to PKCS#1 ver. 1.5 before cryptographic operation which requires 11_{dec} bytes from the length of input data. Therefore the maximum length of TLS challenge, which can be used, is $F5_{hex}$ bytes.

3.2. Calculating the electronic signature

The calculation of electronic signature is the encryption of a hash object which will be formatted according to PKCS#1 ver. 1.5 block type 1.

Card application enables the calculation of the electronic signature using the RSA key in two ways:

- Providing card application with already calculated hash for signing procedure.
- Providing card application with data to be hashed before the signature procedure.

The result of signing procedure is cryptogram of PKCS#1 ver. 1.5 block type 1 enveloped hash. The encryption is done with RSA signature private key.

3.2.1. Calculating the electronic signature with providing pre-calculated hash

Current chapter describes the signing method where hash is calculated by the host application and provided to card application prior to signing operation. For authorising cardholder for given operation it is needed to authenticate the user with PIN2.

For given example let the PIN2 code have the same value as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

Let the active signature private key reference be 0100_{hex} for given example.

To calculate electronic signature for given method, following procedures should be executed:

1) Before currently selected key reference can be set it is needed to select DF FID $EEEE_{hex}$. First navigate to given directory by:
   a) selecting the application MF with command SELECT FILE:

CLA	INS	P1	P2	Le
00_{hex}	$A4_{hex}$	00_{hex}	$0C_{hex}$	00_{hex}

   b) and selecting directory file $EEEE_{hex}$ by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
00_{hex}	$A4_{hex}$	01_{hex}	04_{hex}	02_{hex}	$EEEE_{hex}$

2) Select security environment for signature calculation operations by executing following command MANAGE SECURITY ENVIRONMENT:

CLA	INS	P1	P2	Le
00_{hex}	22_{hex}	$F3_{hex}$	01_{hex}	00_{hex}

3) Set the key reference to active signature key for COMPUTE DIGITAL SIGNATURE command execution by executing command MANAGE SECURITY ENVIRONMENT:

CLA	INS	P1	P2	Lc	Data (TL of TLV || KST || Key reference)
00_{hex}	22_{hex}	41_{hex}	$B8_{hex}$	05_{hex}	8303_{hex} || 80_{hex} || 0100_{hex}

If the card hasn't been set to use secondary signature key pair after last card reset, then it is not required to execute given command.

Key reference values as specified in Table 3-2 EF FID 0013hex key records and references of secret keys active keys should be used. The proper way to receive currently active signature key reference is described in chapter 2.4.3 Reading key references for active keys.

By default it is not necessary to execute the security environment and key reference commands in case of calculating the digital signature with pre-calculated hash.

4) Authorise cardholder for executing command COMPUTE DIGITAL SIGNATURE by authenticating user with PIN2 by executing command VERIFY:

CLA	INS	P1	P2	Lc	Data (PIN2 as ASCII)
00_{hex}	20_{hex}	00_{hex}	02_{hex}	05_{hex}	3132333435_{hex}

5) Calculate the electronic signature by executing command COMPUTE DIGITAL SIGNATURE:

CLA	INS	P1	P2	Lc	Data
00_{hex}	$2A_{hex}$	$9E_{hex}$	$9A_{hex}$	Data length	Hash algorithm identifier || Data of hash

The following list specifies supported hash algorithms and their identifiers:

- SHA-1: $3021300906052B0E03021A05000414_{hex}$
- SHA-224: $302D300D06096086480165030402040500041C_{hex}$
- SHA-256: $3031300D06096086480165030402010500042 0_{hex}$
- SHA-384: $3041300D06096086480165030402020500043 0_{hex}$
- SHA-512: $3051300D06096086480165030402030500044 0_{hex}$

The support of SHA-384 and SHA-512 hash function are optional and depend on the support of the underlying chip, the integrated circuit and the java operating system.

Data transmitted to card is not validated by the card application. In case of invalid data format the card application can respond with response codes described in chapter 7.3 Error response APDU messages.

For successful operation the card application responds with signature RSA private key encrypted PKCS#1 ver. 1.5 block type 1 wrapped data. For card application possible responses see chapter 7.2.13.3 COMPUTE DIGITAL SIGNATURE.

3.2.2. Calculating the electronic signature with internal hash calculating

Current chapter describes the signing method where SHA-1 hash will be calculated by the card application prior to signing operation. This kind of procedure may be needed for POS devices which do not have SHA-1 hashing functionality. For authorising cardholder for given operation it is needed to authenticate the user with PIN2.

For given example let the PIN2 code have the same value as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

Let the active signature private key reference be 0100_{hex} for given example.

To calculate electronic signature for given method, following procedures should be executed:

1) Before currently selected key reference can be set it is needed to select DF FID $EEEE_{hex}$. First navigate to given directory by:
   a) selecting the application MF with command SELECT FILE:

CLA	INS	P1	P2	Le
00_{hex}	$A4_{hex}$	00_{hex}	$0C_{hex}$	00_{hex}

   b) and selecting directory file $EEEE_{hex}$ by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
00_{hex}	$A4_{hex}$	01_{hex}	04_{hex}	02_{hex}	$EEEE_{hex}$

2) Select security environment for signature calculation operations by executing following command MANAGE SECURITY ENVIRONMENT:

CLA	INS	P1	P2	Le
00_{hex}	22_{hex}	$F3_{hex}$	01_{hex}	00_{hex}

3) Set the key reference for COMPUTE DIGITAL SIGNATURE command execution by executing command MANAGE SECURITY ENVIRONMENT:

CLA	INS	P1	P2	Lc	Data (TL of TLV || KST || Key reference)
00_{hex}	22_{hex}	41_{hex}	$B8_{hex}$	05_{hex}	8303_{hex} || 80_{hex} || 0100_{hex}

If the card hasn't been set to use secondary signature key pair after last card reset, then it is not required to execute given command.

Key reference values as specified in Table 3-3 EF FID 0013hex key records and references of secret keys active keys should be used. The proper way to receive currently active signature key reference is described in chapter 2.4.3 Reading key references for active keys.

4) Authorise cardholder for executing command COMPUTE DIGITAL SIGNATURE by authenticating user with PIN2 by executing command VERIFY:

CLA	INS	P1	P2	Lc	Data (PIN2 as ASCII)
00_{hex}	20_{hex}	00_{hex}	02_{hex}	05_{hex}	3132333435_{hex}

5) Transmit data to card application for SHA-1 hashing by executing command HASH:

CLA	INS	P1	P2	Lc	Data
00_{hex}	$2A_{hex}$	90_{hex}	$A0_{hex}$	Data length	SHA-1 identifier || Data for hashing

For successful operation, of current command, SHA-1 hash will be returned in R-APDU and as well the same hash is held inside the card application for using by the following command COMPUTE DIGITAL SIGNATURE.

If the field data length exceeds the length of normal APDU, command HASH must be transmitted as chained or extended. APDU chaining is described in chapter 7.4 Message chaining and extended APDU usage is described in chapter 7.5 Extended APDU.

6) Calculate the electronic signature by executing command COMPUTE DIGITAL SIGNATURE:

CLA	INS	P1	P2	Le
00_{hex}	$2A_{hex}$	$9E_{hex}$	$9A_{hex}$	00_{hex}

For successful operation the card application responds with signature RSA private key encrypted PKCS#1 ver. 1.5 block type 1 wrapped data. For card application possible responses see chapter 7.2.13.3 COMPUTE DIGITAL SIGNATURE.

3.3. Decrypting public key encrypted data

Current chapter describes the decrypting method which must be used to execute RSA decryption operation with private key on data which is encrypted with the corresponding authentication public key. Encrypted data must be formatted according to PKCS#1 version 1.5 block type 2. For authorising cardholder for given operation it is needed to authenticate the user with PIN1.

For given example let the PIN1 code have the same value as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

Let the active and secondary authentication private key references be respectively 1100_{hex} and 1200_{hex} for given example.

To decrypt cryptogram of PKCS#1 ver. 1.5 block type 2 enveloped data, following procedures should be executed:

1) Before currently selected key reference can be set it is needed to select DF FID $EEEE_{hex}$. First navigate to given directory by:
   a) selecting the application MF with command SELECT FILE:

CLA	INS	P1	P2	Le
00_{hex}	$A4_{hex}$	00_{hex}	$0C_{hex}$	00_{hex}

b) and selecting directory file $EEEE_{hex}$ by using command SELECT FILE again.

CLA	INS	P1	P2	Lc	Data (FID)
00_{hex}	$A4_{hex}$	01_{hex}	04_{hex}	02_{hex}	$EEEE_{hex}$

2) Select security environment for decryption operations by executing following command MANAGE SECURITY ENVIRONMENT:

CLA	INS	P1	P2	Le
00_{hex}	22_{hex}	$F3_{hex}$	06_{hex}	00_{hex}

If the card has already been set to use decryption security environment after last card reset, then it is not required to execute given command.

3) Set the key reference for DECIPHER command execution by executing command MANAGE SECURITY ENVIRONMENT:

or

a) For decrypting with private authentication key that has reference value 1100_{hex}, use command:

CLA	INS	P1	P2	Lc	Data (TL of TLV || KST || Key reference)
00_{hex}	22_{hex}	41_{hex}	$A4_{hex}$	05_{hex}	8303_{hex} || 80_{hex} || 1100_{hex}

If the card hasn't been set to use secondary authentication key pair after last card reset, then it is not required to execute given command.

b) For decrypting with private authentication key that has reference value 1200_{hex}, use command:

CLA	INS	P1	P2	Lc	Data (TL of TLV || KST || Key reference)
00_{hex}	22_{hex}	41_{hex}	$A4_{hex}$	05_{hex}	8303_{hex} || 80_{hex} || 1200_{hex}

Key reference values as specified in Table 3-4 EF FID 0013hex key records and references of secret keys active keys should be used. The proper way to receive currently active authentication keys reference is described in chapter 2.4.3 Reading key references for active keys.

It is not possible to use Signature key for deciphering operations. Only authentication keys can be used for this procedure.

4) Authorise cardholder for executing command DECIPHER by authenticating user with executing command VERIFY for verifying PIN1:

CLA	INS	P1	P2	Lc	Data (PIN1 as ASCII)
00_{hex}	20_{hex}	00_{hex}	01_{hex}	04_{hex}	31323334_{hex}

For the following operation it must be sure that decrypted data is formatted according to PKCS#1 ver. 1.5 block type 2 and the cryptogram is the result of encryption with RSA authentication public key. Otherwise the result will be an error situation.

The encrypted data transmitted to the card application must be pre-padded with 00_{hex} byte which indicates that it is formatted according to PKCS#1 ver. 1.5 block type 2.

5) Decrypt and obtain session key by executing command DECIPHER:

CLA	INS	P1	P2	Lc	Data
00_{hex}	$2A_{hex}$	80_{hex}	86_{hex}	Data length	Data for RSA public key encrypted session key

If the length of the cryptogram for command DECIPHER exceeds the data length of normal APDU it must be transmitted as chained or extended. APDU chaining is described in chapter 7.4 Message chaining and extended APDU usage is described in chapter 7.5 Extended APDU.

For successful operation the card application responds with plain data unwrapped from the PKCS#1 ver. 1.5 block type 2 envelope. For card application possible responses see chapter 7.2.13.2 DECIPHER.

4. Card application managing operations

Current chapter describes procedures for the case if there comes up a need to replace PKI objects on the card. These procedures are not meant to be performed by any other institution than card authority.

4.1. Secure channel communication

In terms of card application managing operations it is required to hold transmission channel secured. All messages are secured with 3DES session keys by encrypting data and calculating signature for command.

C-APDUs which support secure channel are described in following table.

Table 4-1 C-APDUs supporting secure channel

INS	Command
05_{hex}	Replace PINs/PUK

Table 4-1 C-APDUs supporting secure channel	
INS	Command
06_{hex}	Generate new key pair
07_{hex}	Replace certificate
$B0_{hex}$	Read Binary
$B2_{hex}$	Read Record
CD_{hex}	Get Data

4.1.1. Mutual Authentication

Mutual Authentication is operation where host application gets authorisation to access card application management operations. To get authorisation host application must authenticate itself to card application.

To authenticate the host application, following operations should be executed:

1) Get challenge (random number RND.ICC) from the card by executing command GET CHALLENGE:

CLA	INS	P1	P2	Le
00_{hex}	84_{hex}	00_{hex}	00_{hex}	08_{hex}

Card responds with 8_{dec} bytes challenge:

Data	SW1	SW2
RND.ICC (8_{dec} bytes)	90_{hex}	00_{hex}

2) Authenticate host application by executing command MUTUAL AUTHENTICATE. Before given command can be executed following operations must be performed:
   a) Generate host application 8 bytes of random data called RND.IFD.
   b) Generate $2*16_{dec}$ random data, called K.IFD, which is host application side data for session keys calculation.
   c) Concatenate together RND.IFD, RND.ICC and K.IFD in given order. You get 30_{hex} of data for encryption.
   d) Derive cardholder CMK key which is related to procedure taking place after Mutual Authentication.
   e) Encrypt 30_{hex} bytes data with derived cardholder CMK 3DES key using CBC mode.
   f) Use calculated cryptogram in command MUTUAL AUTHENTICATE data field and transmit to card application:

CLA	INS	P1	P2	Lc	Data	Le
00_{hex}	82_{hex}	00_{hex}	$0X_{hex}$	30_{hex}	Cryptogram 30_{hex} bytes	30_{hex}

Successful operation response:

Data	SW1	SW2
Cryptogram 30_{hex} bytes	90_{hex}	00_{hex}

g) Decrypt 30_{hex} bytes received data with derived cardholder CMK 3DES key using CBC mode.
h) Decrypted data contains components in following order: RND.ICC || RND.IFD || K.ICC.
i) Verify received RND.IFD by comparing it to generated RND.IFD. They must match!
j) Calculate session keys (SK) by applying XOR operation between K.ICC and K.IFD.
k) Encryption session key (SK1) is the 16_{dec} leftmost bytes of SK. Signature (MAC) session key (SK2) is the 16_{dec} rightmost bytes of SK.
l) Calculate IV called Send Sequence Counter (SSC) for following secured command executions. SCC is put together by concatenating 4 leftmost bytes of RND.IFD and 4 leftmost bytes of RND.ICC.

4.1.2. Channel securing

Commands which are sent to card application as secured must have command data encrypted and command signature (MAC) appended to the command. These operations can be performed after successful authentication procedure described in chapter 4.1.1 Mutual Authentication.

Data encryption and command MAC calculation must be performed with 3DES key using CBC mode. Still there is a small difference between these operations. For MAC calculating operation it is needed to use only 8 leftmost bytes of the DES key using CBC mode for N-1 blocks. Only for the last Nth block encrypting with the full 3DES key using CBC mode is performed. For encryption operation normal 3DES key is used. See figure below for encryption with DES CBC and 3DES CBC.

Data for MAC calculating and encryption operations must be padded according to ISO 9797-1 padding method 2. Given method tells to append 80_{hex} byte and as many 00_{hex} bytes to data until its length is modulus of DES algorithm block length, which is 8.

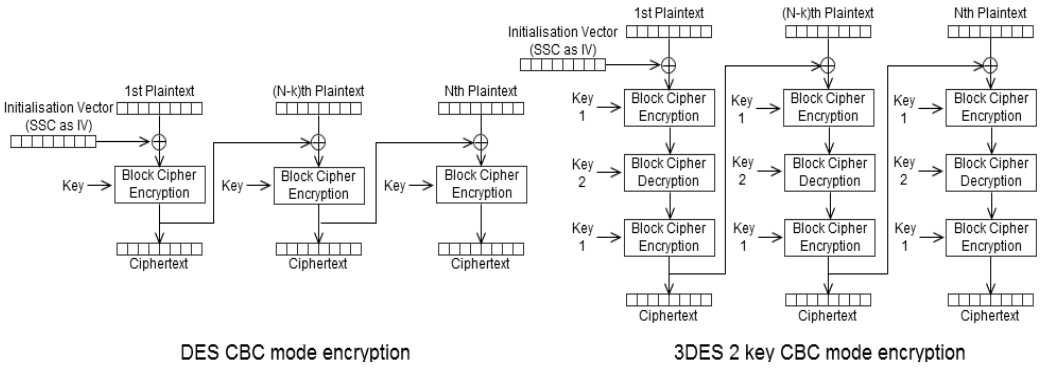

Figure 4-1 DES and 3DES CBC mode encryption

To get the overview of 3DES CBC mode decryption and actual MAC calculating operation, see figure below.

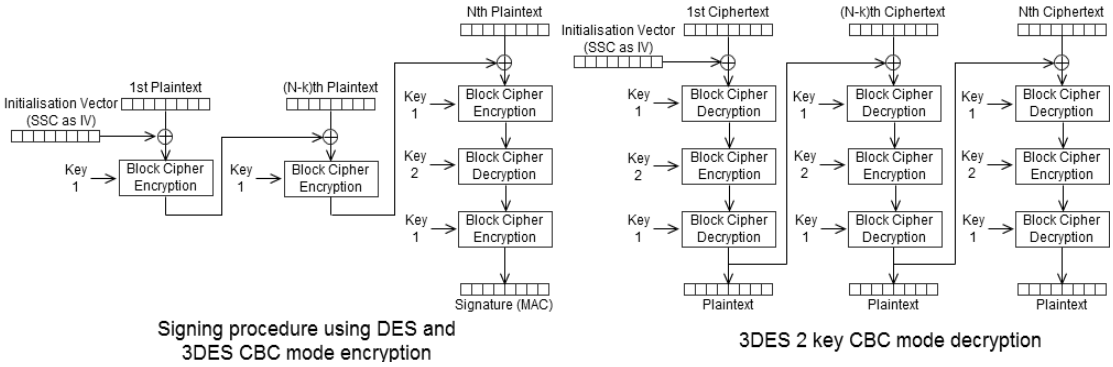

Figure 4-2 MAC signature calculation and 3DES CBC mode decryption

Specification for DES and TDES algorithms can be found in ISO 18033-3:2010.

To make secured C-APDU, following procedures must be performed:

1) Increase the value of SSC with one. If the value of SSC is $FFFFFFFFFFFFFFFF_{hex}$ then after increasing operation it gets value 0000000000000000_{hex}. See following examples:

   $FF73F9D201044A59_{hex}$ + 1 = $FF73F9D201044A59_{hex}$

   $FFFFFFFFFFFFFFFF_{hex}$ + 1 = 0000000000000000_{hex}

2) Change C-APDU CLA value to $0C_{hex}$, which indicates, that the command is secured.

3) If there is data present in C-APDU:

   a) Append padding to data according to ISO 9797-1 method 2.

   b) Encrypt the data of C-APDU with SK1 in 3DES CBC mode using SSC value from previous step as IV.

   c) Wrap cryptogram from previous step to TLV having tag 87_{hex}, which identifies that the value is encrypted.

   d) C-APDU data field must be switched with TLV from previous step.

4) Prepare data for MAC calculating. Get 4 header bytes of C-APDU which are CLA, INS, P1 and P2. Append 80000000_{hex} bytes to it, so the result is 8 bytes in length.

   CLA || INS || P1 || P2 || 80000000_{hex}

5) If there is TLV wrapped cryptogram present in C-APDU, append it to data for MAC calculation.

6) Append padding to MAC calculating data according to ISO 9797-1 method 2.

7) Sign the data with encryption method described in figure MAC signature calculation and 3DES CBC mode decryption. In the figure Key 1 is the 8 leftmost and Key 2 is the 8 rightmost bytes of SK2. As IV the value of SSC must be used. Result of given encryption operation is 8 bytes of MAC signature.

8) Wrap MAC signature from previous step to TLV having tag $8E_{hex}$, which indicates that the value is the MAC signature.

9) Append TLV from previous step to C-APDU data field.

Now C-APDU is secured and can be transmitted to card application. The response from card application as well is secured and must be verified and data decrypted.

To verify and decrypt data of secured R-APDU, following procedures must be performed:

1) Increase the value of SSC with one.
2) Find MAC TLV with tag $8E_{hex}$ from the 10_{dec} leftmost bytes of R-APDU data field. Unwrap the value from this TLV to get MAC signature.
3) Prepare data for MAC calculation. Take R-APDU data field without MAC signature TLV.
4) Append padding to MAC calculating data according to ISO 9797-1 method 2.
5) Sign the data with encryption method described in figure MAC signature calculation and 3DES CBC mode decryption. In the figure Key 1 is the 8 leftmost and Key 2 is the 8 rightmost bytes of SK2. As ICV the value of SSC must be used. Result of given encryption operation is 8 bytes of MAC signature.
6) Verify R-APDU MAC by comparing it to calculated MAC. They must match!
7) If the R-APDU data without MAC signature TLV is TLV with tag…
   a) …99_{hex}, then the TLV value marks the 2 byte status word. It is MAC signed and therefore it can be sure about its integrity.
   b) …87_{hex}, then the TLV value is the cryptogram of actual R-APDU data and it needs to be decrypted.

      Decrypt the data of R-APDU with SK1 in 3DES CBC mode using SSC value from the first step as IV. Result of given decryption operation is the plaintext data of R-APDU.

4.2. PIN1, PIN2 and PUK replacement

In case the cardholder has forgotten or lost PIN/PUK codes, they can be replaced by the EstEID card authority.

To replace PIN/PUK codes, the following procedures must be performed:

1) Perform Mutual Authentication with cardholder CMK derived from CMK_PIN.
2) Replace cardholder PIN1, PIN2 and PUK codes by executing command REPLACE PINS (SECURE) as encrypted and MAC signed as described in chapter 4.1.2 Channel securing:

CLA	INS	P1	P2	Lc
$0C_{hex}$	05_{hex}	00_{hex}	00_{hex}	11_{hex}

For successful operation card application responds with following encrypted and MAC signed R-APDU:

Data	SW1	SW2
empty	90_{hex}	00_{hex}

See APPENDIX chapter Replace cardholder PINs/PUK codes for example with secured messages and mutual authentication.

4.3. Certificate replacement

The validity period of the card may exceed the validity period of the certificates. In that case new certificates based on existing key pairs but with and extended validity period can be requested and loaded.

Current chapter describes the procedure to perform the replacement of cardholder certificate file.

For given example let the PIN1 code have the same values as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

To replace certificates, the following procedures must be performed:

1) Perform Mutual Authentication with cardholder CMK derived from CMK_CERT.
2) Verify cardholder with PIN1 by executing command VERIFY:

CLA	INS	P1	P2	Lc	Data (PIN1 as ASCII)
00_{hex}	20_{hex}	00_{hex}	01_{hex}	04_{hex}	31323334_{hex}

3) Replace current certificate with new one in the card application by executing command REPLACE CERTIFICATE (SECURE) as encrypted and MAC signed as described in chapter 4.1.2 Channel securing:
   a) To replace authentication certificate, execute:

CLA	INS	P1	P2	Lc	Data
$0C_{hex}$	07_{hex}	01_{hex}	00_{hex}	0600_{hex}	

   b) To replace signature certificate, execute:

CLA	INS	P1	P2	Lc	Data
$0C_{hex}$	07_{hex}	02_{hex}	00_{hex}	0600_{hex}	

For successful operation card application responds with following encrypted and MAC signed R-APDU:

Data (TLV formatted 271_{dec} bytes)	SW1	SW2
$7F49_{hex}$ || $82010A_{hex}$ || 81_{hex} || 820100_{hex} || public key || 82_{hex} || 04_{hex} || exponent	90_{hex}	00_{hex}

See APPENDIX chapter Replace Certificates for example with secured messages and mutual authentication.

4.4. New RSA key pair generation

The request and loading of new certificates is not limited to the use of the active key pairs. New key pairs can be generated prior to the request and loading of new certificates.

For given example let the PIN1 code have the same values as in chapter 2.2.1 Verify PIN1, PIN2 or PUK code.

To generate new key pair, the following procedures must be performed:

1) Perform Mutual Authentication with cardholder CMK derived from CMK_KEY.
2) Verify cardholder with PIN1 by executing command VERIFY:

CLA	INS	P1	P2	Lc	Data (PIN1 as ASCII)
00_{hex}	20_{hex}	00_{hex}	01_{hex}	04_{hex}	31323334_{hex}

3) Generate new key pair by executing command GENERATE KEY (SECURE) as encrypted and MAC signed as described in chapter 4.1.2 Channel securing:
   a) To generate new actively used authentication RSA key pair, execute:

CLA	INS	P1	P2	Le
$0C_{hex}$	06_{hex}	01_{hex}	01_{hex}	00_{hex}

   b) To generate new actively used signature RSA key pair, execute:

CLA	INS	P1	P2	Le
$0C_{hex}$	06_{hex}	01_{hex}	02_{hex}	00_{hex}

   c) To generate new secondary authentication RSA key pair, execute:

CLA	INS	P1	P2	Le
$0C_{hex}$	06_{hex}	02_{hex}	01_{hex}	00_{hex}

   d) To generate new secondary signature RSA key pair, execute:

CLA	INS	P1	P2	Le
$0C_{hex}$	06_{hex}	02_{hex}	02_{hex}	00_{hex}

After previous command active key references for corresponding authentication or signature key gets changed to one which was generated. The new reference value for currently active keys should be read from file with FID 0033_{hex} as described in chapter 2.4.3 Reading key references for active keys.

For successful operation card application responds with following encrypted and MAC signed R-APDU:

Data (TLV formatted 271_{dec} bytes)	SW1	SW2
$7F49_{hex}$ || $82010A_{hex}$ || 81_{hex} || 820100_{hex} || public key || 82_{hex} || 04_{hex} || exponent	90_{hex}	00_{hex}

See APPENDIX chapter Generate new key pair for example with secured messages and mutual authentication.

5. Card application security structure

Card application has three environments:

- Public environment – Reading data objects on the card.
- PKI environment – Needs PIN verification for operating.
- Card application authority environment – Using CMK secure messaging for operating.

All card operations and their access rights are showed in following table:

	Reading personal data file	Reading user certificates	Using Authentication key	Using signature key	PIN1 unblocking (also with PIN1 replacement)	PIN2 unblocking (also with PIN2 replacement)	Changing PIN1, PIN2 and PUK	Deciphering and authenticating operations	Digital signing operation	Unblocking and replacing PIN1, Pin2 and PUK	Replacing certificates	Generating new key pair
Public environment	ALW	ALW	NEV	NEV	PUK	PUK	PIN/ PUK	NEV	NEV	NEV	NEV	NEV
PKI environment	NEV	NEV	PIN1	PIN2	NEV	NEV	NEV	PIN1	PIN2	NEV	NEV	NEV
Card application authority environment	ALW	ALW	NEV	NEV	NEV	NEV	NEV	NEV	NEV	ALW	PIN1	PIN1

Acronym	Description
ALW	Always allowed
NEV	Not allowed
PIN1	Operation can be used after PIN1 is verified
PIN2	Operation can be used after PIN2 is verified
PIN/PUK	Each PIN or PUK can be changed with verifying corresponding current PIN or PUK

6. Card application constants

Some of the objects that are set on the card in personalisation phase cannot be manipulated afterwards. These constant values concern the maximum and minimum values of object and fixed object values.

Table 6-1 Card application constant values					
Length	Length in bytes			Initial value	Fixed value
	Min	Max	Initial		
PIN1	4	12_{dec}	4	From personalisation	
PIN2	5	12_{dec}	5		
PUK	8	12_{dec}	8		
RSA	256_{dec}	256_{dec}	256_{dec}	Generated in personalisation	
RSA public exponent	4	4	4	40000081_{hex}	40000081_{hex}
Certificate file	600_{hex}	600_{hex}	600_{hex}	From personalisation	
PIN/PUK retry counter	1	1	1	03_{hex}	
RSA usage countdown	3	3	3	$FFFFFF_{hex}$	
CMK_PIN	16_{dec}	16_{dec}	16_{dec}	From personalisation	From personalisation
CMK_ CERT	16_{dec}	16_{dec}	16_{dec}		
CMK_KEY	16_{dec}	16_{dec}	16_{dec}		

Lengths specified in chapter 2.1 Personal data file in table Personal Data file contents, should be taken as a part of current chapter.

The exponent of RSA keys is either 40000081_{hex}, if supported by the platform, or 00010001_{hex}, if the underlying platform is not capable of supporting arbitrary exponents.

7. APDU protocol

Communication between a chip card and a host application is performed over application-level APDU protocol. Current chapter and subchapters gives the basics of APDU protocol usage. APDU protocol itself is specified in ISO 7816-4 standard.

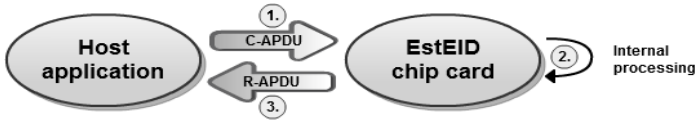

Figure 7-1 APDU master-slave communication

APDU messages compromise two structures: one used by the host application to send commands to the card whereas one is used by the chip to send command response back to the host application. Data transmission between two ends is performed as master-slave communication where a host application is the master and a chip is the slave.

Command sent by a host application is called Command APDU (C-APDU) or simply APDU. Command sent by the chip as a response to C-APDU is called Response APDU (R-APDU).

APDU messages can be transmitted with two different transmission-level Transmission Protocol Data Units (TPDU) which are T0 and T1. The T0 and T1 protocols are used to support APDU protocols transmission between chip reader and chip itself. APDU protocol is used between the chip application and the chip reader.

T1 is block oriented protocol which enables blocks or grouped collections of data to be transferred. These data groups are transferred as a whole between chip and reader. The theoretical maximum length of T1 grouped collections for C-APDU is 65535_{dec} and for R-APDU is 65536_{dec} bytes. The practical maximum length depends on the chip platform that is used for EstEID application.

T0 is byte oriented protocol which means that the minimum data that can be transferred has a length of one byte. The maximum length of data structure that can be transferred with this protocol for C-APDU is 255_{dec} and for R-APDU is 256_{dec} bytes.

APDU structure defined in ISO 7816-4 standard is very similar to TDPU structure used in T0. When APDU is transmitted with T0, the elements of APDU exactly overlay the elements of TPDU.

Table 7-1 C-APDU structure

Header				Body		
CLA	INS	P1	P2	Lc	Data	Le
				Optional for T1		
				Optional for T0		

Table 7-2 C-APDU contents

Code	Name	Length	Description
CLA	Class	1	Class of instruction
INS	Instruction	1	Instruction code. Basically a function number in the card.
P1	Parameter 1	1	Instruction parameter 1
P2	Parameter 2	1	Instruction parameter 2
Ex	Extended indicator	missing or 1	With value 00_{hex} indicates that the command data has extended format. Look chapter 7.5 Extended APDU.
Lc	Length	variable 1 or 2	Number of bytes present in the data field of the command
Data	Data	variable, equal to Lc	String of bytes sent in the data field of the command
Le	Length	variable 1 or 2	Maximum number of bytes expected in the data field of the response to the command

Keep in mind that by using T1 protocol either Le or data field has to be present always. When there is no specific value for Le or data field while using T1, then Le field must be set to value 00_{hex}.

Table 7-3 R-APDU structure		
Body	Trailer	
Data	SW1	SW2

Table 7-4 R-APDU contents			
Code	Name	Length	Description
Data	Data	variable, equal to Le if was present in C-APDU	Sequence of bytes received in the data field of the response (Optional field)
SW1	Status byte 1	1	Command processing status
SW2	Status byte 2	1	Command processing qualifier

7.1. Card possible response in case of protocol T0

When using protocol T0 and sending a C-APDU that should return data, the card responds with R-APDU that informs the host how many bytes are waiting to be read. The sequence of operations in given case is described in following example:

1) The card responds to the C-APDU with trailer $61XX_{hex}$ as a positive response. XX_{hex} in the response indicates how many bytes of data are waiting to be read from the card:

SW1	SW2
61_{hex}	XX_{hex}

2) In order to read the given bytes, the following GET RESPONSE command must be sent to the card:

CLA	INS	P1	P2	Le
00_{hex}	$C0_{hex}$	00_{hex}	00_{hex}	XX_{hex}

If more bytes are waiting to be read from the card, the card keeps responding $61XX_{hex}$ after every GET RESPONSE command execution as long there will be none waiting to be read.

3) The card responds:

Data	SW1	SW2
XX_{hex} bytes of data	90_{hex}	00_{hex}

For T0 there is also possible a case when card responds with status code which informs to reissue the same APDU command with Le byte set as marked in status. In given case card responds as follows where XX_{hex} marks Le byte value that should be used when reissuing the command:

SW1	SW2
$6C_{hex}$	XX_{hex}

After reissuing the APDU command the card responds as normally.

7.2. Command APDU

Card application APDU commands are derived from ISO 7816-4 but do not implement all given specification functionalities. Implemented is minimal of required functionalities for EstEID PKI operations. Current chapter gives detailed usage information of the implemented functions. All implemented APDU commands are listed in the following table.

Table 7-5 Card application implemented APDU commands

Command name	INS	Description
SELECT FILE	$A4_{hex}$	To change pointer for currently selected file.
READ RECORD	$B2_{hex}$	Reads data from Linear EF.
READ BINARY	$B0_{hex}$	Reads data from Transparent EF.
GET RESPONSE	$C0_{hex}$	Receive data from card in case of status $61XX_{hex}$.
GET DATA	CA_{hex}	Read data related to application or chip. ▪ Application version ▪ CPLC ▪ Available memory on chip
GET CHALLENGE	84_{hex}	Generates and returns random number for authentication purposes.
VERIFY	20_{hex}	To verify the presented user PIN1/PIN2/PUK code against the stored reference values.
CHANGE REFERENCE DATA	24_{hex}	To change PIN1/PIN2/PUK code values on the card.
RESET RETRY COUNTER	$2C_{hex}$	To reset PIN1 or PIN2 retry counters.
MANAGE SECURITY ENVIRONMENT	22_{hex}	Sets currently active key environment for cryptographic operations.
INTERNAL AUTHENTICATE	88_{hex}	To authenticate card by the host side
MUTUAL AUTHENTICATE	82_{hex}	To authenticate host for card management operations.
PERFORM SECURITY OPERATION ▪ HASH ▪ DECIPHER ▪ COMPUTE DIGITAL SIGNATURE	$2A_{hex}$	Functions to perform various cryptographic operations with card application RSA key pair private keys.
EstEID card application authority operations.		
REPLACE PINS (SECURE)	05_{hex}	To replace PIN/PUK codes for cardholder.
GENERATE KEY (SECURE)	06_{hex}	To generate new RSA key pair for cardholder.
REPLACE CERTIFICATE (SECURE)	07_{hex}	To replace cardholder certificate.

7.2.1. SELECT FILE

CLA	INS	P1	P2	Lc	Data	Le	Command description
00_{hex}	$A4_{hex}$	$0X_{hex}$	00_{hex}	Empty or 2. Described in P1 table below.	according to P1 field	empty or 00_{hex}	Response includes FCI (FCP+FMD)
		$0X_{hex}$	04_{hex}				Response includes FCP
		$0X_{hex}$	08_{hex}				Response includes FMD
		$0X_{hex}$	$0C_{hex}$				Response includes only OK status 0x9000 if successful

The SELECT FILE command is used to change the logical pointer of currently selected file to perform operations on. The file identification can be provided by file identifier (FID) on 2 bytes.

Selected file logical pointer changing method is defined in P1 field. These methods are provided in following table:

Value of X in P1	Lc	Data	Description
0	empty or	empty or	Select application MF
	2	$3F00_{hex}$	
1	2	FID	Select DF with file identifier declared in Data
2	2	FID	Select EF with file identifier declared in Data
3	empty	empty	Select parent DF of currently selected DF

Card application can answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
empty or data containing FCI, FCP or FMD	90_{hex}	00_{hex}	Successfully changed the file pointer (and returned Data)
	64_{hex}	09_{hex}	Could not generate FCP for selectable DF.
	67_{hex}	00_{hex}	Invalid length of data for provided P1 value
	$6A_{hex}$	80_{hex}	Invalid FID for MF
	$6A_{hex}$	82_{hex}	File not found for provided FID
	$6A_{hex}$	86_{hex}	Possible reasons: ▪ invalid P1 value ▪ invalid P2 value

7.2.2. READ RECORD

CLA	INS	P1	P2	Lc	Data	Le
00_{hex}	$B2_{hex}$	Record no. of the record to be read	04_{hex}	empty	empty	00_{hex} or exactly the length of the record

The READ RECORD command is used to read data records from Linear EF. Linear EF is structured file containing records.

Card application answers to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Data with length of record or provided by Le value	90_{hex}	00_{hex}	Successfully read the record
	62_{hex}	82_{hex}	Record's value is shorter than provided in Le
	67_{hex}	00_{hex}	Record's value is longer than provided in Le
	69_{hex}	81_{hex}	Trying to read record from non-Linear EF
	69_{hex}	86_{hex}	Missing selection pointer for EF
	$6A_{hex}$	83_{hex}	The Requested record is not found.
	$6A_{hex}$	86_{hex}	P2 value is not 04_{hex}

7.2.3. READ BINARY

CLA	INS	P1 || P2	Lc	Data	Le
00_{hex}	$B0_{hex}$	$XXXX_{hex}$ Offset to start a reading from the file	empty	empty	number of bytes to read

The READ BINARY command is used to read binary data from Transparent EF.

There is no need to read whole data from the file with multiple READ BINARY commands with different file reading offset for each command. READ BINCARY command supports extended length response. See chapter 7.5 Extended APDU.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Binary data with length of data or provided by Le	90_{hex}	00_{hex}	Successfully read the binary data from EF

Data	SW1	SW2	Description
Binary data with shorter length than requested by Le value	62_{hex}	82_{hex}	Returned file contents but warning, that data that was read was shorter than requested.
	69_{hex}	81_{hex}	Trying to read binary data from non-Transparent EF
	69_{hex}	86_{hex}	Missing selection pointer for EF
	$6B_{hex}$	00_{hex}	Offset is bigger than file actual size

7.2.4. GET RESPONSE

CLA	INS	P1	P2	Lc	Data	Le
00hex	C0hex	00hex	00hex	empty	empty	XXhex

The GET RESPONSE command is used for protocol T0 to get data from the card, which is sent by the card implicitly. Before given command can be executed, the card must send status 61XXhex, where XX marks the length of a data available for returning from the card. For GET RESPONSE command the same value of XX must be used as received in status.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Data with length as provided in Le	90_{hex}	00_{hex}	Successful operation
	61_{hex}	XX_{hex}	Additional data waiting to be returned from the card.
	--	--	No other errors than defined in chapter 7.3 Error response APDU messages.

7.2.5. GET DATA

CLA	INS	P1	P2	Lc	Data	Le
00_{hex}	CA_{hex}	XX_{hex}	00_{hex}	empty	empty	00_{hex}

The GET DATA command is used to get various information of the EstEID application and card itself. The information that the card should return is defined in P1 field. Possible P1 values and result descriptions are specified in the following table:

P1	Description for data returned by card
01_{hex}	EstEID application version.
02_{hex}	CPLC data for chip.
03_{hex}	Current free memory on the card.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
$XXXX_{hex}$	90_{hex}	00_{hex}	P1 = 01_{hex} – EstEID application version as BCD.
42 bytes of CPLC data.			P1 = 02_{hex} – CPLC data for chip.
$XXXX_{hex}$			P1 = 03_{hex} Free transient memory that is freed on application deselecting or reset.
|| $YYYY_{hex}$			Free transient memory that is freed on application reset.
|| $ZZZZ_{hex}$			Free persistent type memory.
	$6A_{hex}$	86_{hex}	Invalid P1 value

If there is more free memory available than $FFFF_{hex}$ then given memory value will be returned as $FFFF_{hex}$.

7.2.6. GET CHALLENGE

CLA	INS	P1	P2	Lc	Data	Le
00_{hex}	84_{hex}	00_{hex}	00_{hex}	empty	empty	00_{hex} or 08_{hex} or XX_{hex}

The GET CHALLENGE command is used to receive a challenge (e.g. random number) for use in a security related procedure.

Le field defines the length for data that should be generated in the card. If Le field is empty or has value 00_{hex} then the length of random is considered to be 08_{hex}. Only Le field with value 08_{hex} result is stored for further internal operations. Random numbers generated with other length are only for off card usage.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Data with length provided by Le.	90_{hex}	00_{hex}	Random data.
	$6A_{hex}$	86_{hex}	Invalid P1 or P2 value.

7.2.7. VERIFY

CLA	INS	P1	P2	Lc	Data	Le	Description
00_{hex}	20_{hex}	00_{hex}	01_{hex}	$04_{hex}...0C_{hex}$	PIN1	empty	PIN and PUK codes verification operations
			02_{hex}	$05_{hex}...0C_{hex}$	PIN2		
			00_{hex}	$08_{hex}...0C_{hex}$	PUK		

The VERIFY command is used to authenticate cardholder through PIN1, PIN2 or PUK code.

Upper and lower limits for the lengths of PIN1, PIN2 and PUK are marked in Lc field. Default verification data length for given verification method is the minimum marked in Lc field. Other lengths of verification data can be used after successful operation of command CHANGE REFERENCE DATA.

PIN1, PIN2 and PUK codes must be provided in communication as ASCII character numbers.

Unsuccessful operation of given command results in decrementing the corresponding PIN/PUK code retry counter.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
empty	90_{hex}	00_{hex}	Successful verification.
	63_{hex}	CX_{hex}	Verification failed. X in SW2 marks remaining tries for verification.
	67_{hex}	00_{hex}	Verification data can't be empty.
	69_{hex}	83_{hex}	Verification method blocked.
	$6A_{hex}$	80_{hex}	PIN/PUK code value is out of the expected limits range.
	$6A_{hex}$	86_{hex}	Invalid P1 or P2 value.

7.2.8. CHANGE REFERENCE DATA

CLA	INS	P1	P2	Lc	Data	Le	Ref. data
00_{hex}	24_{hex}	00_{hex}	01_{hex}	old length + new length	old PIN1 || new PIN1	empty	PIN1
			02_{hex}	old length + new length	old PIN2 || new PIN2		PIN2
			00_{hex}	old length + new length	old PUK || new PUK		PUK

The CHANGE REFERENCE DATA command is used to replace PIN1, PIN2 or PUK code. To change PIN1/PIN2/PUK code it is needed to know the currently active code. It is allowed to assign only new PIN1/PIN2/PUK which is different from the current one.

PIN1, PIN2 and PUK codes must be provided in communication as ASCII character numbers.

Unsuccessful operation of given command results in decrementing the corresponding PIN/PUK code retry counter.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
empty	90_{hex}	00_{hex}	Reference data successfully changed.
	63_{hex}	CX_{hex}	Verification failed. X in SW2 marks remaining tries for verification.
	67_{hex}	00_{hex}	Verification data can't be empty.
	69_{hex}	83_{hex}	Verification method blocked.
	69_{hex}	85_{hex}	If length is invalid
	$6A_{hex}$	80_{hex}	▪ Old and new PIN/PUK are equal. ▪ New PIN/PUK is is out of the expected limits range.
	$6A_{hex}$	86_{hex}	Invalid P1 or P2 value.

7.2.9. RESET RETRY COUNTER

CLA	INS	P1	P2	Lc	Data	Le	Description
00_{hex}	$2C_{hex}$	00_{hex}	$0X_{hex}$	PUK + PIN1/PIN2 length	PUK || new PIN1/PIN2	empty	Verify PUK and assign new PIN1/PIN2.
		03_{hex}	$0X_{hex}$	empty	empty	00_{hex}	Reset PIN1/PIN2. PUK pre-verified.
		$0X_{hex}$	01_{hex}	Defined by P1		empty	Operation with PIN1.
		$0X_{hex}$	02_{hex}				Operation with PIN2.

The RESET RETRY COUNTER command is used to replace reset or unblock PIN1 or PIN2 code.

To use P1 with value 03_{hex} it is needed to have command VERIFY PUK used. For operation P1 with value 00_{hex} for verification it is needed to provide PUK code as well with the new PIN1/PIN2 code.

The command cannot be used for PIN1/PIN2 which is in blocked state.

PIN1, PIN2 and PUK codes should be provided in communication as ASCII character numbers.

Unsuccessful operation of given command can result in decrementing PUK code retry counter if P1 as value 00_{hex} used.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
empty	90_{hex}	00_{hex}	Successful operation
	63_{hex}	CX_{hex}	Verification failed. X in SW2 marks remaining tries for PUK verification.
	69_{hex}	82_{hex}	PUK not pre-verified

Data	SW1	SW2	Description
	69_{hex}	83_{hex}	Verification method blocked.
	69_{hex}	85_{hex}	PIN trying to reset, is not blocked.
	$6A_{hex}$	86_{hex}	Invalid P1 or P2 value.

7.2.10. MANAGE SECURITY ENVIRONMENT

CLA	INS	P1	P2	Lc	Data	Le	Description
00_{hex}	22_{hex}	$F3_{hex}$	01_{hex}	empty	empty	00_{hex}	Set security environment for signing and authentication operations.
			06_{hex}				Set security environment for deciphering operation.
		41_{hex}	$A4_{hex}$ $B4_{hex}$ $B6_{hex}$ $B8_{hex}$	02_{hex}	8300_{hex}	empty	Reset key references to active ones.
				05_{hex}	830380_{hex} || $XXXX_{hex}$		Set key reference to specific key by providing key reference in the position of XXXX.

The MANAGE SECURITY ENVIRONMENT command is used to change the currently active pointers to keys for security operations.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
empty	90_{hex}	00_{hex}	Successful operation
	67_{hex}	00_{hex}	Invalid length of data for provided command parameters.
	69_{hex}	83_{hex}	Verification method blocked.
	$6A_{hex}$	86_{hex}	Invalid P1 or P2 value.
	$6A_{hex}$	80_{hex}	Incorrect data (invalid length).

7.2.11. INTERNAL AUTHENTICATE

CLA	INS	P1	P2	Lc	Data	Le
00_{hex}	88_{hex}	00_{hex}	00_{hex}	Token length	Authentication token	Empty

The INTERNAL AUTHENTICATE command is used to authenticate the cardholder by the host side. Data field in C-APDU must contain token that will be encrypted with private key stored in the card. Challenge can be verified by using public key of the same key pair.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
RSA authentication private key encrypted TLS challenge which is formatted according to PKCS#1 ver. 1.5 block type 1.	90_{hex}	00_{hex}	Successful. Data field contains encrypted challenge.
	69_{hex}	00_{hex}	Security environment is not set to Authenticate
	69_{hex}	82_{hex}	PIN1 is not validated.
	$6A_{hex}$	86_{hex}	Invalid P1 or P2 value.
	$6A_{hex}$	80_{hex}	Token has invalid length. Has to fit into RSA key length.

7.2.12. MUTUAL AUTHENTICATE

CLA	INS	P1	P2	Lc	Data	Le	For use of
00_{hex}	82_{hex}	00_{hex}	01_{hex}	30_{hex}	CMK encrypted RND.IFD || RND.ICC || K.IFD	empty or 00_{hex} or 30_{hex}	CMK_PIN
			02_{hex}				CMK_CERT
			03_{hex}				CMK_KEY

The MUTUAL AUTHENTICATION command is used for host authentication. After successful operation of given command card management commands can be used over secure encrypted channel.

The whole process of mutual authentication is described in chapter 4.1.1 Mutual Authentication.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Corresponding CMK encrypted RND.ICC || RND.IFD || K.ICC	90_{hex}	00_{hex}	Successful operation. Data field contains encrypted card and host challenges and card session key.
	63_{hex}	CF_{hex}	Mutual authentication failed.
	64_{hex}	00_{hex}	Incorrect P2 value. No such CMK
	67_{hex}	00_{hex}	Data has length different to 30_{hex}
	$6A_{hex}$	86_{hex}	Invalid P1 value.

7.2.13. PERFORM SECURITY OPERATION

CLA	INS	P1	P2	Lc	Data	Le
$0X_{hex}$	$2A_{hex}$	XX	XX	XX	XX	~~empty~~

The PERFORM SECURITY OPERATION command is for three cryptographic algorithms:

- HASH – calculates bit hash from the data transferred by the command.
- DECIPHER – decrypts cryptogram which is transferred by the command.

- COMPUTE DIGITAL SIGNATURE – computes digital signature for the data transferred by the command.

7.2.13.1. HASH

CLA	INS	P1	P2	Lc	Data	Le	Description
00_{hex}	$2A_{hex}$	90_{hex}	$A0_{hex}$	Data length	Data for hashing	empty	Last data block.
10_{hex}							Chain data block.

The HASH command is used to calculate unique data value on provided data. Algorithm used for hashing is SHA1.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Generated hash for provided data.	90_{hex}	00_{hex}	Successful operation.
	--	--	No other errors than defined in chapter 7.3 Error response APDU messages. .

7.2.13.2. DECIPHER

CLA	INS	P1	P2	Lc	Data (Data for deciphering)	Le	Description
00_{hex}	$2A_{hex}$	80_{hex}	86_{hex}	Data length	00_{hex} || cryptogram	empty	Last data block.
10_{hex}							Chain data block.

The DECIPHER command is used to decipher data provided by the command. Data has to be formatted according to PKCS#1 ver. 1.5 block type 2 with the respective public key. Given operation can be performed only with private authentication keys.

Data transmitted to the card application for deciphering must be pre-padded with 00_{hex} byte which indicates that the data under the cryptogram is formatted according to PKCS#1 ver. 1.5 block type 2:

00_{hex} || cryptogram

Needs PIN1 to be pre-verified.

DECIPHER command supports chaining and extended C-APDU for data transmitting. For extended APDU see chapter 7.5 Extended APDU and for chaining see chapter 7.4 Message chaining.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Deciphered data.	90_{hex}	00_{hex}	Successful deciphering.
	69_{hex}	00_{hex}	Security environment is not set to Decipher.

Data	SW1	SW2	Description
	69_{hex}	82_{hex}	PIN1 is not validated.

7.2.13.3. COMPUTE DIGITAL SIGNATURE

CLA	INS	P1	P2	Lc	Data	Le	Description
00_{hex}	$2A_{hex}$	$9E_{hex}$	$9A_{hex}$	Data length	Data for signature	empty	Data signing
				00_{hex}	empty		Hash signing

The COMPUTE DIGITAL SIGNATURE command is used to compute unique signature for data or for hash generated previous to this command. For signature it is used in card private key of RSA key pair.

Needs PIN2 to be pre-verified.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
Computed signature.	90_{hex}	00_{hex}	Successful operation.
	69_{hex}	00_{hex}	Security environment is not set to Digital Signature.
	69_{hex}	82_{hex}	PIN2 is not validated.
	$6A_{hex}$	88_{hex}	Missing hash that has to be generated prior to this command with HASH command.

7.2.14. REPLACE PINS (SECURE)

CLA	INS	P1	P2	Lc	Data (PIN1, PIN2 and PUK are as ASCII)	Le
$0C_{hex}$	05_{hex}	00_{hex}	00_{hex}	11_{hex}	PIN1 || PIN2 || PUK	empty

The REPLACE PINS command is used to replace current PINs and PUK codes with new ones by EstEID card authority.

Given APDU command requires secure communication channel for processing, as described in chapter 4.1 Secure channel communication.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
empty	90_{hex}	00_{hex}	Successful operation.
	$6A_{hex}$	86_{hex}	Invalid P1 or P2 value.

Data	SW1	SW2	Description
	69_{hex}	86_{hex}	Wrong Mutual Authentication CMK key used for current command.
	67_{hex}	00_{hex}	Missing hash that has to be generated prior to this command with HASH command.

7.2.15. GENERATE KEY (SECURE)

CLA	INS	P1	P2	Lc	Data	Le	Description
$0C_{hex}$	06_{hex}	01_{hex}	01_{hex}	empty	empty	empty	Authentication key nr 1
			02_{hex}			or 00_{hex}	Signature key nr 1
		02_{hex}	01_{hex}			or $010F_{hex}$	Authentication key nr 2
			02_{hex}			as extended	Signature key nr 2

The GENERATE KEY command is used to generate new RSA key pairs by EstEID card authority.

Given APDU command requires secure communication channel for processing, as described in chapter 4.1 Secure channel communication.

Accessing given command requires as well the acceptance from the cardholder by verifying PIN1 code with command VERIFY.

Card application answer to given command with R-APDU described in following table.

Data (271_{dec} bytes)	SW1	SW2	Description
$7F49_{hex}$ || $82010A_{hex}$ || 81_{hex} || 820100_{hex} || public key || 82_{hex} || 04_{hex} || exponent	90_{hex}	00_{hex}	Successful operation. Data field containing TLV data for public key of generated RSA key pair.
	$6A_{hex}$	86_{hex}	Invalid P1 or P2 value.
	69_{hex}	86_{hex}	Wrong Mutual Authentication CMK key used for current command.
	67_{hex}	00_{hex}	Missing hash that has to be generated prior to this command with HASH command.

For detailed information of the template look ISO 7816-8.

7.2.16. REPLACE CERTIFICATE (SECURE)

CLA	INS	P1 8bit	P1 7-1bit	P2	Lc	Data	Le	Description
$0C_{hex}$	06_{hex}	0_{bit}	XX_{hex} ($00\text{-}7F_{hex}$)	XX_{hex}	Data length	Certificate data	empty	Authentication certificate
		1_{bit}						Signature certificate

The REPLACE CERTIFICATE command is used to replace cardholder authentication or signature certificate by EstEID card authority.

The maximum length of data for the certificate can be is 600_{hex} bytes. The certificate data must fit into this length and have padding according to ISO 9797-1 padding method 2.

The certificate file must be written to card as multiple chunks. Each following chunk must be send to the card by using offset of the last sent data.

Given APDU command requires secure communication channel for processing, as described in chapter 4.1 Secure channel communication.

Accessing given command requires as well the acceptance from the cardholder by verifying PIN1 code with command VERIFY.

Card application answer to given command with R-APDU described in following table.

Data	SW1	SW2	Description
empty	90_{hex}	00_{hex}	Successful operation.
	69_{hex}	86_{hex}	Wrong Mutual Authentication CMK key used for current command or PIN1 is not successfully validated.
	$6A_{hex}$	86_{hex}	Invalid P1 or P2 value.

7.3. Error response APDU messages

Error codes provided in the following table can be returned by any of C-APDUs. These errors are not the result of the usual command processing.

Table 7-6 APDU error status codes

SW1	SW2	Definition
68_{hex}	84_{hex}	Command chaining not supported.
$6D_{hex}$	00_{hex}	Command instruction not supported.
$6E_{hex}$	00_{hex}	Command class not supported.
$6F_{hex}$	00_{hex}	No precise diagnosis.
$6F_{hex}$	66_{hex}	Internal inconsistency.

7.4. Message chaining

This chapter explains the basis of APDU message chaining in case of larger data to be transmitted.

JavaCard framework 2.2.2 and above support extended length APDU messages. Still TPDU T0 protocol does not support extended length APDU processing. Thus the data to be transmitted between the host and the card which doesn't fit into the usual length of APDU messages must be transmitted as chained in the case of T0 usage.

EstEID card application does not respond with data longer than maximum of standard APDU response messages. Thus there is no need for message chaining in given case.

The maximum length of data that can be sent by normal APDU is 255_{dec} bytes. If there is more data do transfer to the card than the maximum length then it is needed to split the data into as many blocks it could be delivered to the card.

The execution of the operation takes place when the last block is received by the card. Chained blocks and last block of data are distinguished by the command class. The command class bit which determines that the command is a part of the chained command sequence is 10_{hex}. The last block of the sequence must have the chaining bit set off in command class.

For getting a better overview, following example where it is needed to send 700_{dec} bytes of data to the card for card internal processing.

1) First C-APDU:

CLA	INS	P1	P2	Lc	Data	Le
10_{hex}	XX	XX	XX	255_{dec}	First block of 255_{dec} bytes of 700_{dec} bytes	XX

Data	SW1	SW2	Description
empty	90_{hex}	00_{hex}	Block received. Waiting for another one.

2) Second C-APDU:

CLA	INS	P1	P2	Lc	Data	Le
10_{hex}	XX	XX	XX	255_{dec}	Second block of 255_{dec} bytes of 700_{dec} bytes	XX

Data	SW1	SW2	Description
empty	90_{hex}	00_{hex}	Block received. Waiting for another one.

3) Third and last C-APDU:

CLA	INS	P1	P2	Lc	Data	Le
00_{hex}	XX	XX	XX	190_{dec}	Last block of 190_{dec} bytes of 700_{dec} bytes	XX

Data	SW1	SW2	Description
Operation result data	90_{hex}	00_{hex}	Last block received. Operation successful and result data returned.

7.5. Extended APDU

As mentioned in previous chapter JavaCard framework 2.2.2 and above support extended length APDU messages. If data for the command exceeds the maximum of normal C-APDU, which is 255_{dec} bytes, the data can be sent as extended. Maximum length for extended length data that can be transmitted is 65536_{dec} bytes – the actual size of short data type.

Extended APDU can only be used with protocol T1.

In extended C-APDU in Lc and Le fields have length of 2 bytes. Extended C-APDU has Le field always present. Lc field is optional. C-APDU body must always be pre-padded with 00_{hex} which is the indicator of extended APDU.

- If Lc and data fields are present:

CLA	INS	P1	P2	Ex. APDU indicator	Lc	Data	Le
XX	XX	XX	XX	00_{hex}	$XXXX_{hex}$	Extended data	$XXXX_{hex}$

- If Lc and data fields are absent:

CLA	INS	P1	P2	Ex. APDU indicator	Lc	Data	Le
XX	XX	XX	XX	00_{hex}	empty	empty	$XXXX_{hex}$

Extended length R-APDU does not have differences compared to normal one. Just the data that is returned by R-APDU is exceeding the length of 256_{dec} bytes.

APPENDIX

This appendix contains real life card application operations logs, which should give a better overview of the commands. Following operations are performed with test environment card application using transmission protocol T1 with Le always present.

Master PIN/PUK codes and CMK keys used in following operations are the same as mentioned in chapter 2.2.1 Verify PIN1, PIN2 or PUK code and 2.5 Card application management keys: CMK_PIN, CMK_CERT & CMK_.

Reset the chip with EstEID card application installed on

Chip responds with ATR.

```
<<   3B FE 18 00 00 80 31 FE 45 45 73 74 45 49 44 20 76 65 72 20 31 2E 30 A8
     TS  : 3B  Direct logic
     T0  : FE  K    =    14 byte [historical characters]
     TA1 : 18  Fi/f =   372/ 5   [clock rate conversion factor / max. frequency (MHz)]
               Di   =    12      [bit rate conversion factor]
     TB1 : 00  pa   =     4 %    [programming voltage current]
               I    =    25 mA   [maximum current]
               P    =     0 V    [programming voltage]
     TC1 : 00  N    =     0 etu  [extra guard time]
     TD1 : 80  T    =   T=0      [protocol type]
     TD2 : 31  T    =   T=1      [protocol type]
     TA3 : FE  IFSC =   254      [information field size]
     TB3 : 45  CWT  =    43 etu  [character waiting time]
               BWT  = 15371 etu  [block waiting time]
     (place for historical bytes)
```

PIN1, PIN2 and PUK operations

```
>>   00 20 00 01 04 31 32 33 34 00 – VERIFY (PIN1)
<<   90 00 - OK
>>   00 20 00 02 05 31 32 33 34 35 00 – VERIFY (PIN2)
<<   90 00 - OK
>>   00 20 00 00 08 31 32 33 34 35 36 37 38 00 – VERIFY (PUK)
<<   90 00 - OK
// Change PIN1 “1234” => “4321”
>>   00 24 00 01 08 31 32 33 34 34 33 32 31 00 – CHANGE REFERENCE DATA (PIN1)
<<   90 00 - OK
// Change PIN2 “12345” => “54321”
>>   00 24 00 02 0A 31 32 33 34 35 35 34 33 32 31 00 – CHANGE REFERENCE DATA (PIN2)
<<   90 00 - OK
// Change PIN2 “12345678” => “87654321”
>>   00 24 00 00 10 31 32 33 34 35 36 37 38 38 37 36 35 34 33 32 31 00 – CHANGE REFERENCE
     DATA (PIN3)
<<   90 00 - OK
// Block PIN1
>>   00 20 00 01 04 31 32 33 34 00 – VERIFY (PIN1)
```

```
<<   63 C2 – FAILED (2 tries remaining)
>>   00 20 00 01 04 31 32 33 34 00 – VERIFY (PIN1)
<<   63 C1 - OK (1 tries remaining)
>>   00 20 00 01 04 31 32 33 34 00 – VERIFY (PIN1)
<<   63 C0 - OK (0 tries remaining, blocked)
// Block PIN2
>>   00 20 00 02 05 31 32 33 34 35 00 – VERIFY (PIN2)
<<   63 C2 – FAILED (2 tries remaining)
>>   00 20 00 02 05 31 32 33 34 35 00 – VERIFY (PIN2)
<<   63 C1 - OK (1 tries remaining)
>>   00 20 00 02 05 31 32 33 34 35 00 – VERIFY (PUK)
<<   63 C0 - OK (0 tries remaining, blocked)
// Unblock PIN1 with pre-verified PUK
>>   00 20 00 00 08 38 37 36 35 34 33 32 31 00 – VERIFY (PUK)
<<   90 00 - OK
>>   00 2C 03 01 00 – RESET RESTY COUNTER
<<   90 00 - OK
// Unblock PIN2 with same command PUK verification and new PIN2 assigning
>>   00 2C 00 02 0C 3 37 36 35 34 33 32 31 31 32 33 34 35 – RESET RESTY COUNTER
<<   90 00 - OK
// Select EF FID 0016 for reading
>>   00 A4 02 0C 02 00 16 00 - SELECT (EF 0016)
<<   90 00 - OK
// Read record 1 containing info for PIN1
>>   00 B2 01 04 00 – READ RECORD
<<   80 01 03 90 01 03 83 02 00 00 90 00 - OK (Tries: max = 3, remaining = 3)
// Read record 2 containing info for PIN2
>>   00 B2 02 04 00 – READ RECORD
<<   80 01 03 90 01 03 83 02 00 00 90 00 - OK (Tries: max = 3, remaining = 3)
// Read record 3 containing info for PUK
>>   00 B2 03 04 00 – READ RECORD
<<   80 01 03 90 01 03 90 00 - OK (Tries: max = 3, remaining = 3)
```

Navigate to DF FID $EEEE_{hex}$

```
>>   00 A4 00 0C 00 – SELECT (MF)
<<   90 00 - OK
>>   00 A4 01 0C 02 EE EE 00 – SELECT (DF EEEE)
<<   90 00 - OK
```

Select EF FID 5044_{hex} and read all of its contents

```
>>   00 A4 02 0C 02 50 44 00 – SELECT (EF Personal data)
<<   90 00 – OK
>>   00 B2 01 04 00 – READ RECORD (Surname)
<<   4D C4 4E 4E 49 4B 90 00 – OK “MÄNNIK”
>>   00 B2 02 04 00 – READ RECORD (First name 1)
<<   4D 41 52 49 2D 4C 49 49 53 90 00 – OK “MARI-LIIS”
```

```
>>   00 B2 03 04 00 – READ RECORD (First name 2)
<<   90 00 – OK “”
>>   00 B2 04 04 00 – READ RECORD (Sex)
<<   4E 90 00 – OK “N”
>>   00 B2 05 04 00 – READ RECORD ()
<<   45 53 54 90 00 – OK “EST”
>>   00 B2 06 04 00 – READ RECORD (Birth date)
<<   30 31 2E 30 31 2E 31 39 37 30 90 00 – OK “01.01.1971”
>>   00 B2 07 04 00 – READ RECORD (Personal identification number)
<<   34 37 31 30 31 30 31 30 30 33 33 90 00 – OK “47101010033” [Seed for CMKs]
>>   00 B2 08 04 00 – READ RECORD (Document number)
<<   41 53 30 30 31 31 31 32 35 90 00 – OK “AS0011125”
>>   00 B2 09 04 00 – READ RECORD (Expiration date)
<<   30 31 2E 30 32 2E 32 30 31 37 90 00 – OK “01.02.2017”
>>   00 B2 0A 04 00 – READ RECORD (Birth place)
<<   45 45 53 54 49 20 2F 20 45 53 54 90 00 – OK “EESTI / EST”
>>   00 B2 0B 04 00 – READ RECORD (Issuance date)
<<   30 31 2E 30 31 2E 32 30 31 32 90 00 – OK “01.01.2012”
>>   00 B2 0C 04 00 – READ RECORD (Residence permit type)
<<   90 00 – OK “”
>>   00 B2 0D 04 00 – READ RECORD (Notes 1)
<<   90 00 – OK “”
>>   00 B2 0E 04 00 – READ RECORD (Notes 2)
<<   90 00 – OK “”
>>   00 B2 0F 04 00 – READ RECORD (Notes 3)
<<   90 00 – OK “”
>>   00 B2 10 04 00 – READ RECORD (Notes 4)
<<   90 00 – OK “”
```

Read certificate files

Read authentication certificate using multiple C-APDUs

```
>>   00 A4 02 0C 02 AA CE – SELECT (EF Authentication certificate)
<<   90 00 - OK
>>   00 B0 00 00 00 – READ BINARY
<<   30 82 05 3C 30 82 04 24 A0 03 02 01 02 02 10 02 EF 82 9D E1 97 96 94 50 1F 73 79 B6 6E
     05 DA 30 0D 06 09 2A 86 48 86 F7 0D 01 01 05 05 00 30 6C 31 0B 30 09 06 03 55 04 06 13
     02 45 45 31 22 30 20 06 03 55 04 0A 0C 19 41 53 20 53 65 72 74 69 66 69 74 73 65 65 72
     69 6D 69 73 6B 65 73 6B 75 73 31 1F 30 1D 06 03 55 04 03 0C 16 54 45 53 54 20 6F 66 20
     45 53 54 45 49 44 2D 53 4B 20 32 30 31 31 31 18 30 16 06 09 2A 86 48 86 F7 0D 01 09 01
     16 09 70 6B 69 40 73 6B 2E 65 65 30 1E 17 0D 31 32 30 38 30 36 30 37 33 34 31 37 5A 17
     0D 31 37 30 32 30 31 32 31 35 39 35 39 5A 30 81 9B 31 0B 30 09 06 03 55 04 06 13 02 45
     45 31 0F 30 0D 06 03 55 04 0A 0C 06 45 53 54 45 49 44 31 17 30 15 06 03 55 04 0B 0C 0E
     61 75 74 68 65 6E 74 69 63 61 74 69 6F 6E 31 26 30 24 06 03 55 04 03 0C 90 00 - OK
```

File length: $53C_{hex}$ + 4 = 540_{hex}

Parts to read: (540_{hex} + 100_{hex}) / 100_{hex} = 6

Last part length: 540_{hex} % 100_{hex} = 40_{hex}

```
>>   00 B0 01 00 00 – READ BINARY (2nd part)
```

```
<<   1D 4D C3 84 4E 4E 49 4B 2C 4D 41 52 49 2D 4C 49 49 53 2C 34 37 31 30 31 30 31 30 30 33
     33 31 10 30 0E 06 03 55 04 04 0C 07 4D C3 84 4E 4E 49 4B 31 12 30 10 06 03 55 04 2A 0C
     09 4D 41 52 49 2D 4C 49 49 53 31 14 30 12 06 03 55 04 05 13 0B 34 37 31 30 31 30 31 30
     30 33 33 30 82 01 23 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01 05 00 03 82 01 10 00 30 82
     01 0B 02 82 01 01 00 91 B9 0A F5 4B E5 14 C1 CD C0 8F 50 7C 39 BC D2 8F F9 89 5F 1C 88
     1B 1B 85 60 2C CA ED F4 65 1E EA 8B 25 4F 69 AB 4C 7B 22 1F D8 EF 49 29 17 A4 EA 4C AD
     C4 72 CC AF 5C 0D 16 C4 02 2C D0 AD 1A 24 BE 68 46 BE C2 9A 74 3D 6C 66 81 69 7C D4 5C
     D6 62 64 8A CD 60 DC 50 AE 46 2F ED 35 D0 D5 EE 84 FC 12 CE EA 93 ED 8B 65 8D BC 11 33
     39 F1 CB C2 DF 9D 76 55 48 F7 A1 54 52 C6 48 62 E5 93 51 E6 1C 9F 9B 9F 90 00 - OK

>>   00 B0 02 00 00 – READ BINARY (3rd part)

<<   D6 77 57 1A 94 6B C6 23 FD 90 AD 5D 0C 9E 03 DE E6 C2 4F E9 94 B7 BD C0 D4 30 4F 2A 12
     04 AF BE BD 1B D4 70 CF BB C6 38 42 71 91 EC E6 D6 BF 72 A4 55 98 58 05 4E 80 BE 2C 63
     1A 56 95 56 40 F1 F1 20 72 A4 EE B3 99 C9 E7 8B 6F 7A 4F FE EE 5C 8B 5F 3A F4 1B D0 48
     DB 6B 8D 84 82 88 94 F9 38 D3 A9 2D F2 44 21 9C 98 74 41 47 2C 0B 4E 1D 01 F9 EB 55 11
     2F 9A 12 7A 2B 06 4D 02 04 7E 0B 83 71 A3 82 01 A7 30 82 01 A3 30 09 06 03 55 1D 13 04
     02 30 00 30 0E 06 03 55 1D 0F 01 01 FF 04 04 03 02 04 B0 30 81 99 06 03 55 1D 20 04 81
     91 30 81 8E 30 81 8B 06 0A 2B 06 01 04 01 CE 1F 03 01 01 30 7D 30 58 06 08 2B 06 01 05
     05 07 02 02 30 4C 1E 4A 00 41 00 69 00 6E 00 75 00 6C 00 74 00 20 00 74 00 65 00 73 00
     74 00 69 00 6D 00 69 00 73 00 65 00 6B 00 73 00 2E 00 20 00 4F 00 6E 00 90 00 - OK

>>   00 B0 03 00 00 – READ BINARY (4th part)

<<   6C 00 79 00 20 00 66 00 6F 00 72 00 20 00 74 00 65 00 73 00 74 00 69 00 6E 00 67 00 2E
     30 21 06 08 2B 06 01 05 05 07 02 01 16 15 68 74 74 70 3A 2F 2F 77 77 77 2E 73 6B 2E 65
     65 2F 63 70 73 2F 30 27 06 03 55 1D 11 04 20 30 1E 81 1C 6D 61 72 69 2D 6C 69 69 73 2E
     6D 61 6E 69 6B 2E 31 34 40 65 65 73 74 69 2E 65 65 30 1D 06 03 55 1D 0E 04 16 04 14
     54 6A 42 9E DF B4 88 33 78 85 4A 70 9A 84 AD EE D6 4C 5F 1A 30 20 06 03 55 1D 25 01 01
     FF 04 16 30 14 06 08 2B 06 01 05 05 07 03 02 06 08 2B 06 01 05 05 07 03 04 30 18 06 08
     2B 06 01 05 05 07 01 03 04 0C 30 0A 30 08 06 06 04 00 8E 46 01 01 30 1F 06 03 55 1D 23
     04 18 30 16 80 14 41 B6 FE C5 B1 B1 B4 53 13 8C FA FA 62 D0 34 6D 6D 22 34 0A 30 45 06
     03 55 1D 1F 04 3E 30 3C 30 3A A0 38 A0 36 86 34 68 74 74 70 3A 2F 2F 77 90 00 - OK

>>   00 B0 04 00 00 – READ BINARY (5th part)

<<   77 77 2E 73 6B 2E 65 65 2F 72 65 70 6F 73 69 74 6F 72 79 2F 63 72 6C 73 2F 74 65 73 74
     5F 65 73 74 65 69 64 32 30 31 31 2E 63 72 6C 30 0D 06 09 2A 86 48 86 F7 0D 01 01 05 05
     00 03 82 01 01 00 5D E2 16 91 4D A4 6C 24 41 99 F4 5B EE 3B B2 F1 60 9A EB 77 4D EE 3C
     2C B8 B3 6F 00 5D 2E 25 F0 5D E7 D2 C6 73 42 02 4E 31 E9 DA B2 68 77 66 39 FB 99 27 E9
     F8 58 23 E3 FA 46 14 54 0E 02 F7 93 3D 0E 9A 58 E1 84 E8 6D D8 26 A7 70 F7 C7 DA B6 EB
     6B 65 21 FE 9C 45 BD E3 27 9E 4E A4 F5 02 27 D3 17 BC 77 D5 7B 4D 8A 27 1D EB 18 52 A3
     02 68 2E 5D 9A 00 6D 6A 85 50 F9 52 A6 8E B8 78 D4 98 05 B6 33 69 E8 A7 CD FB 00 DE 4A
     12 4B 62 7A E8 29 4C BC 32 63 3E 20 A1 DC 68 D9 AA B9 DA 25 19 FE 76 83 EA 76 66 AC CB
     66 14 F5 B5 5B 32 83 06 F8 F8 C5 8C 2E B7 DA 63 29 81 49 46 53 20 56 5F 90 00 - OK

>>   00 B0 05 00 40 – READ BINARY (6th part)

<<   1E 8F F5 41 4C 8C 8B 77 29 2F DE 67 5E EC AD 7E 42 EE A5 81 4C 17 BC 25 3D F8 43 23 77
     47 33 B1 86 BB DB 79 67 E4 FC 69 94 B5 C1 9A 4D 72 0F D8 28 FD 61 C7 21 D2 83 8C F8 C6
     44 05 39 0A C6 25 90 00 – OK

## Derive active authentication key from certificate ##

RSA public modulus:
     F82F7ECD9FB168FE8550FE10E383C25D9AD460EC4BC64B9994D053A3EBD56BD651B75090A80883CAE5E1F6
     A62E9E395B7F0E6DF444227241C84CAE8AC0AEEE728E6C7CC9634FD0930340F94BCC8B04E0D9ADD14423B1
     F0F8217566B64C6645FBF7E4F31671FB6DB345262976524F4B564A074F906617E77BF00897DCE78FC00F0E
     84B2F7C4988D0CB15D1A9E8ABAF66C383FF0A68A7956C277CE0210436F142FC60BF1CEDE88B3C607B41B54
     4E4D67171333BEEF618666B04D9A02A24FE8E0D75A9A9C95674D9E66416F5B400FD167AE71A0D48057E8BA
     401EE68E9A63595178E4978594427C19068B90192FA23EBE6C36A53AA7078CEC1925CA87CD4FDAF197

RSA public exponent: 40000081
```

Read signature certificate using extended C-APDU

```
>>   00 A4 02 0C 02 DD CE – SELECT (EF Signature certificate)

<<   90 00 - OK
```

```
>>    00 B0 00 00 00 06 00 – READ BINARY (EXTENDED)
<<    30 82 04 F4 30 82 03 DC A0 03 02 01 02 02 10 51 B7 08 FF 70 E0 79 E5 50 1F 74 BD 85 42
      24 79 30 0D 06 09 2A 86 48 86 F7 0D 01 01 05 05 00 30 6C 31 0B 30 09 06 03 55 04 06 13
      02 45 45 31 22 30 20 06 03 55 04 0A 0C 19 41 53 20 53 65 72 74 69 66 69 74 73 65 65 72
      69 6D 69 73 6B 65 73 6B 75 73 31 1F 30 1D 06 03 55 04 03 0C 16 54 45 53 54 20 6F 66 20
      45 53 54 45 49 44 2D 53 4B 20 32 30 31 31 31 18 30 16 06 09 2A 86 48 86 F7 0D 01 09 01
      16 09 70 6B 69 40 73 6B 2E 65 65 30 1E 17 0D 31 32 30 38 30 36 30 37 33 39 34 31 5A 17
      0D 31 37 30 32 30 31 32 31 35 39 35 39 5A 30 81 9E 31 0B 30 09 06 03 55 04 06 13 02 45
      45 31 0F 30 0D 06 03 55 04 0A 0C 06 45 53 54 45 49 44 31 1A 30 18 06 03 55 04 0B 0C 11
      64 69 67 69 74 61 6C 20 73 69 67 6E 61 74 75 72 65 31 26 30 24 06 03 55 04 03 0C 1D 4D
      C3 84 4E 4E 49 4B 2C 4D 41 52 49 2D 4C 49 49 53 2C 34 37 31 30 31 30 31 30 30 33 33 31
      10 30 0E 06 03 55 04 04 0C 07 4D C3 84 4E 4E 49 4B 31 12 30 10 06 03 55 04 2A 0C 09 4D
      41 52 49 2D 4C 49 49 53 31 14 30 12 06 03 55 04 05 13 0B 34 37 31 30 31 30 31 30 30 33
      33 30 82 01 23 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01 05 00 03 82 01 10 00 30 82 01 0B
      02 82 01 01 00 94 9B 68 23 BE 18 18 64 F1 8E 45 51 52 83 D2 3A 31 BA 69 91 54 08 F3 A7
      71 80 5F A2 3C 36 1D 94 FD 30 D0 14 3D 45 D3 C2 9C 20 31 3B 9F CF 80 F0 81 36 36 33 57
      6F E7 5F E9 A0 34 AE 49 A9 3F BF 47 73 D8 17 3B 5C 99 D3 05 4B 5F 8D B1 00 3D FB D4 50
      ED 09 1D E7 59 24 CE FB A4 86 DC 0C A1 2A 8B 2F 7E DC C5 E0 99 7A 1E C1 5C 48 AC 63 A8
      91 F2 F0 46 C1 C7 E6 E0 EA EF 3C E6 15 1B AD 0A 9E 9F 1A DF 37 AC 22 AF 3E 5B 06 D0 14
      9A 95 22 A6 45 10 05 D5 A2 B0 50 2F 73 17 F8 EE 08 28 25 93 2B 3A 38 5E 6C 30 96 92 A4
      3A FE 49 D8 C4 37 0A D0 45 50 80 4D 88 F2 2F 13 B9 C8 E6 60 43 66 E0 B7 AE 91 8F AC 1D
      A0 99 0D 6F 6C 4A C0 FA 78 CD 71 76 9B 95 21 05 57 B6 E0 A3 BE 1F 76 EE B4 CC 51 C0 E7
      4F C6 F6 B3 9D AF 0A DA D7 65 2C A2 F8 BD 42 3B BA 38 EC E2 DE 94 F5 8B 8C C9 D4 E6 81
      02 04 2C F7 B4 B7 A3 82 01 5C 30 82 01 58 30 09 06 03 55 1D 13 04 02 30 00 30 0E 06 03
      55 1D 0F 01 01 FF 04 04 03 02 06 40 30 81 99 06 03 55 1D 20 04 81 91 30 81 8E 30 81 8B
      06 0A 2B 06 01 04 01 CE 1F 03 01 01 30 7D 30 58 06 08 2B 06 01 05 05 07 02 02 30 4C 1E
      4A 00 41 00 69 00 6E 00 75 00 6C 00 74 00 20 00 74 00 65 00 73 00 74 00 69 00 6D 00 69
      00 73 00 65 00 6B 00 73 00 2E 00 20 00 4F 00 6E 00 6C 00 79 00 20 00 66 00 6F 00 72 00
      20 00 74 00 65 00 73 00 74 00 69 00 6E 00 67 00 2E 30 21 06 08 2B 06 01 05 05 07 02 01
      16 15 68 74 74 70 3A 2F 2F 77 77 77 2E 73 6B 2E 65 65 2F 63 70 73 2F 30 1D 06 03 55 1D
      0E 04 16 04 14 11 E2 9A 16 42 D7 73 64 DB 4E C7 D2 40 09 90 E0 08 18 B1 A5 30 18 06 08
      2B 06 01 05 05 07 01 03 04 0C 30 0A 30 08 06 06 04 00 8E 46 01 01 30 1F 06 03 55 1D 23
      04 18 30 16 80 14 41 B6 FE C5 B1 B1 B4 53 13 8C FA FA 62 D0 34 6D 6D 22 34 0A 30 45 06
      03 55 1D 1F 04 3E 30 3C 30 3A A0 38 A0 36 86 34 68 74 74 70 3A 2F 2F 77 77 77 2E 73 6B
      2E 65 65 2F 72 65 70 6F 73 69 74 6F 72 79 2F 63 72 6C 73 2F 74 65 73 74 5F 65 73 74 65
      69 64 32 30 31 31 2E 63 72 6C 30 0D 06 09 2A 86 48 86 F7 0D 01 01 05 05 00 03 82 01 01
      00 C4 EB 5D 08 98 A7 75 B6 60 84 EB 25 78 65 F1 51 7E 70 92 47 B1 81 17 F4 9B 6A 60 66
      6E 66 D5 B3 28 2C 02 50 18 36 14 90 AF 0A 35 36 A6 B5 2D 9A B9 EC E3 6D E8 11 42 20 30
      1B 33 2F 54 13 19 45 2D FE 15 E4 75 DA B3 92 A0 C0 74 5F EF C5 16 8F A8 B9 06 39 F6 C6
      31 DB 8E 38 58 F4 BB ED 66 E7 1A 7B 30 9B 83 7E 7D 64 9E 0D 46 C9 0B 16 55 B3 15 A3 0F
      A3 01 D6 F9 8E B2 5C C7 B1 E8 5B F7 01 0E 1B 80 16 B0 45 86 88 EA 66 C2 8F 8B 04 F5 CC
      D7 27 45 8C 9C BC 36 A9 37 C2 76 27 3F DE 0D 03 E2 60 92 79 92 00 27 77 E9 43 3E 5A 8A
      2C 15 82 96 FC BA 2F 77 56 8B D8 CE 65 DE EE 09 55 14 5E EC 69 AE 83 B1 25 64 07 E9 26
      D0 D3 64 51 BB EC 48 77 1F EC 5F 23 FC 16 DF 99 8A 70 F6 DE 0E 82 2B 5A 89 F3 BF 2E 35
      96 46 4A E6 B8 6E C6 2F 6E 2C 6B 86 A6 18 36 AA 19 86 3E 37 E2 B0 51 62 49 80 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 90
      00
```

Read cardholder secret keys info from file 0013$_{hex}$

```
>>    00 A4 02 0C 02 00 13 00 – SELECT (EF 0013)
<<    90 00 - OK
```

```
>>   00 B2 01 04 00 – READ RECORD (1)
<<   83 04 01 00 00 00 C0 02 81 FF 91 03 FF FF FF 90 00 – OK (Key reference = 0100, )
>>   00 B2 02 04 00 – READ RECORD (2)
<<   83 04 02 00 00 00 C0 02 81 00 91 03 FF FF FF 90 00 – OK (Key reference = 0200, )
>>   00 B2 03 04 00 – READ RECORD (3)
<<   83 04 11 00 00 00 C0 02 81 FF 91 03 FF FF FF 90 00 – OK (Key reference = 1100, )
>>   00 B2 04 04 00 – READ RECORD (4)
<<   83 04 12 00 00 00 C0 02 81 00 91 03 FF FF FF 90 00 – OK (Key reference = 1200, )
```

Read miscellaneous information

```
>>   00 CA 01 00 00 – GET DATA (Card application version)
<<   03 05 90 00 – OK (v3.5)
>>   00 CA 02 00 00 – GET DATA (CPLC)
<<   40 90 61 64 40 91 90 37 2A 00 01 82 20 01 2A 1A 9C BC 40 90 02 99 40 92 02 99 30 37 35
     36 54 36 35 33 30 37 35 36 54 36 35 33 90 00 - OK
>>   00 CA 03 00 00 – GET DATA (Memory)
<<   05 9B 05 9B 7F FF 90 00 – OK
```

Card application general operations

Calculate response for TLS challenge

```
>>   00 22 F3 01 00 – MANAGE SECURITY ENVIRONMENT (Select active keys)
<<   90 00 - OK
>>   00 20 00 01 04 34 33 32 31 00 – VERIFY (PIN1)
<<   90 00 - OK
Random 24 bytes: 3F 4B E6 4B C9 06 6F 14 8A 39 21 D8 7C 94 41 40 99 72 4B 58 75 A1 15 78
>>   00 88 00 00 18 3F 4B E6 4B C9 06 6F 14 8A 39 21 D8 7C 94 41 40 99 72 4B 58 75 A1 15 78
     00 – INTERNAL AUTHENTICATE
<<   D5 85 2F 7F 9E A4 A9 DC D7 16 7D 23 DE A2 5B 06 A8 AA 09 CF 48 2B 50 A6 30 23 1F D4 B9
     86 10 93 3A 6E 4C 6F 1A 54 4B D1 16 8D 30 A4 52 24 10 1E 12 DB 86 31 F8 A5 E0 9A 05 77
     93 86 34 5A 7F 26 AB 65 17 62 5C 04 69 DF 5C 07 24 95 22 F9 57 F4 24 A5 DA 83 E9 60 39
     0B DB CF 8A 87 55 DB F4 4C 14 B6 65 47 F1 EE 4E 97 CB 1A 32 BF 8A 69 07 3D 3A 2A 44 75
     75 1F 58 75 27 6C 7B 32 78 C2 68 38 B8 9E E2 9F 81 68 29 6F EA 38 BC 1D A8 42 A1 56 6D
     ED 01 61 F6 F2 60 36 2A A0 31 0D 5F BD 4B ED 2E 53 32 FF FD F6 EB 81 56 E5 9A BF 97 CA
     93 09 42 08 4C 57 CA 2C EF 9A 9D F3 84 4B F3 5C 95 6D C1 C6 B2 21 AB C6 31 FF 24 D1 9B
     76 E5 79 FB 1B AF 67 EA 8D 77 A0 40 CF 7E BD 62 3E AE 85 A7 2C 3A 19 C4 03 1C 5E 81 DD
     38 74 57 B2 33 D6 27 CC 17 97 6E 85 6A 02 4A 58 5B 6B DE 21 2C FC AC D5 90 00 - OK
Decrypted_Random = RSApub.decrypt(R-APDU data):  3F 4B E6 4B C9 06 6F 14 8A 39 21 D8 7C 94
                                                 41 40 99 72 4B 58 75 A1 15 78
Random == Decrypted_Random (true)
```

Calculate electronic signature from pre-calculated SHA1 hash

```
>>   00 22 F3 01 00 – MANAGE SECURITY ENVIRONMENT (Select active keys)
<<   90 00 - OK
>>   00 20 00 02 05 31 32 33 34 35 00 – VERIFY (PIN2)
<<   90 00 - OK
SHA1("MARI-LIIS MÄNNIK"): F8 E5 40 13 C8 61 C2 A7 46 3E 50 B9 BD 4C E3 2D 64 9D EF 9C
```

```
>>   00 2A 9E 9A 23 30 21 30 09 06 05 2B 0E 03 02 1A 05 00 04 14 F8 E5 40 13 C8 61 C2 A7 46
     3E 50 B9 BD 4C E3 2D 64 9D EF 9C 00 – PERFORM SECURITY OPERATION (COMPUTE DIGITAL
     SIGNATURE)
<<   34 86 60 A4 2B 6C 58 F4 D6 0D 18 67 60 55 2C B3 38 EC C7 0D 1A E8 1E B5 F9 6A 86 9D 90
     81 43 3C 94 61 5C 8E 74 EA 60 D6 C9 52 F4 FF 42 A1 BE 33 A1 7D 13 1C B7 F5 CA C8 59 BE
     52 1D 6D E9 BA 1E E3 81 39 A8 01 F7 EE BA B9 34 03 46 68 16 D3 84 B0 DA D3 E0 A2 61 B7
     37 7A E3 A2 8E 71 2B 59 EE 62 76 5A C3 9E D4 EA 43 BA E8 E0 86 52 F9 5D A6 24 C6 C0 8E
     55 BC 24 24 1F 58 71 8C EE 88 FE 3A 5E 32 45 51 BA EF D8 63 76 D7 FF 39 36 7E F2 9F 09
     46 27 92 48 76 5C F1 E6 1E 88 67 46 CC F2 69 08 06 01 90 6D 9D C4 E5 9C F2 31 B8 D0 0F
     24 DB 61 95 6C 10 09 65 C2 47 6B 2A D8 54 95 94 FA 85 48 1F 02 FE 71 CB 02 8E 32 EC 4C
     FD FA 0D AE 69 4F B6 E8 DB DC 15 A8 DD 07 1F 94 AA 0C DB 50 29 9C 3F 8A 30 26 C1 0D E5
     D6 8F DC A3 DE 8E 3C 71 61 11 9A E9 5A 70 C0 68 64 71 DA DB C4 75 F3 29 90 00 - OK
```

Calculate electronic signature with in-card SHA1 calculation

```
>>   00 22 F3 01 00 – MANAGE SECURITY ENVIRONMENT (Select active keys)
<<   90 00 - OK
>>   00 20 00 02 05 31 32 33 34 35 00 – VERIFY (PIN2)
<<   90 00 - OK
>>   00 2A 90 A0 10 4D 41 52 49 2D 4C 49 49 53 20 4D C4 4E 4E 49 4B 00 - PERFORM SECURITY
     OPERATION (HASH)
<<   F8 E5 40 13 C8 61 C2 A7 46 3E 50 B9 BD 4C E3 2D 64 9D EF 9C 90 00 - OK
>>   00 2A 9E 9A 00 - PERFORM SECURITY OPERATION (COMPUTE DIGITAL SIGNATURE)
<<   34 86 60 A4 2B 6C 58 F4 D6 0D 18 67 60 55 2C B3 38 EC C7 0D 1A E8 1E B5 F9 6A 86 9D 90
     81 43 3C 94 61 5C 8E 74 EA 60 D6 C9 52 F4 FF 42 A1 BE 33 A1 7D 13 1C B7 F5 CA C8 59 BE
     52 1D 6D E9 BA 1E E3 81 39 A8 01 F7 EE BA B9 34 03 46 68 16 D3 84 B0 DA D3 E0 A2 61 B7
     37 7A E3 A2 8E 71 2B 59 EE 62 76 5A C3 9E D4 EA 43 BA E8 E0 86 52 F9 5D A6 24 C6 C0 8E
     55 BC 24 24 1F 58 71 8C EE 88 FE 3A 5E 32 45 51 BA EF D8 63 76 D7 FF 39 36 7E F2 9F 09
     46 27 92 48 76 5C F1 E6 1E 88 67 46 CC F2 69 08 06 01 90 6D 9D C4 E5 9C F2 31 B8 D0 0F
     24 DB 61 95 6C 10 09 65 C2 47 6B 2A D8 54 95 94 FA 85 48 1F 02 FE 71 CB 02 8E 32 EC 4C
     FD FA 0D AE 69 4F B6 E8 DB DC 15 A8 DD 07 1F 94 AA 0C DB 50 29 9C 3F 8A 30 26 C1 0D E5
     D6 8F DC A3 DE 8E 3C 71 61 11 9A E9 5A 70 C0 68 64 71 DA DB C4 75 F3 29 90 00 – OK
```

Perform deciphering operation

```
>>   00 22 41 A4 05 83 03 80 11 00 00 – MANAGE SECURITY ENVIRONMENT (Select active keys)
<<   90 00 - OK
>>   00 20 00 01 04 34 33 32 31 00 – VERIFY (PIN1)
<<   90 00 – OK
Input_data = (“MARI-LIIS MÄNNIK”)
RSApub.encrypt(Input_data) for command DECIPHER
>>   00 2A 80 86 00 01 01 00 51 F5 BA 50 3A 44 D2 77 6D FF 63 F9 BD EE 29 76 E2 F6 1C E6 02
     0B 69 75 AF BC 12 F3 A3 B2 72 85 2D 80 54 1E 25 EA 4D FD 39 A7 F7 C1 4D 2D 3A 50 F3 6A
     F4 DE 60 C2 98 C0 22 01 14 E3 FE 65 B6 48 68 DB 4A FB 01 4B 33 42 61 10 1D 94 FD 33 77
     6A 50 A4 DB 45 EF 2B 4A 5B BA D8 D8 9E B8 E2 01 52 D4 9B DB 37 83 61 EE 4C 30 8F 6A 02
     D6 E5 B4 F7 36 77 B6 39 5F 4A EC 41 A0 2B 34 6C 6E 0A 5B AC CB 58 85 D3 EF B8 C6 D3 DA
     C5 86 A8 9B 6F 02 5E F3 48 0B A7 CC CF 2C 45 DA 59 84 F1 46 CC F4 C3 3C 7D C4 BC 0C 02
     BF 71 18 87 3E 23 5A DD 4F 4D 8C DB F7 A6 94 06 B6 51 49 E4 9C D5 47 DF 6F F2 A5 F7 72
     03 E9 F9 FE 65 06 1B DE 7B C7 2E 47 E0 97 27 BC 3E D4 A9 18 36 1F D0 57 07 B6 4B 15 9F
     EB CC B3 95 B2 8D A8 59 05 A5 CF B2 53 FF 19 E0 65 AB CC 83 E6 ED AB 05 CE 30 56 37 12
     75 14 90 00 00
<<   4D 41 52 49 2D 4C 49 49 53 20 4D C4 4E 4E 49 4B 90 00 - OK
Input_data == R-APDU data (true)
```

Card application managing operations

Replace cardholder PINs/PUK codes

```
// Use master CMK_PIN
// Calulate cardholder CMK. “47101010033” as seed, from EF 5044 record 7.
    SHA1("47101010033") = 74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36 23 3A B3 35 37 0C
    Take 16 leftmost bytes of calulated SHA1:   74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36
                                                23 3A
    CMK.encrypt(74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36 23 3A) =  A7 5F 61 AF 5A E4 74 F1
                                                                    BD BD 0B AB 3B AE 9E DD
    Set off LSB bits of every byte: A65E60AE5AE474F0BCBC0AAA3AAE9EDC (cardholder CMK)
>>  00 84 00 00 08 – GET CHALLENGE
<<  9F 44 39 78 09 B3 C7 E9 90 00 - OK
RND_ICC: 9F 44 39 78 09 B3 C7 E9
Generate RND_IFD: E8 81 41 E4 DC A1 99 82
Generate K_IFD:   8E 8F B4 A3 9F C8 2D 96 7A AB C0 BD BD 8D 85 55 85 09 74 A6 F9 AC D2 5B
                  07 BC 1D E7 6D EF 7C BE
CMK.encrypt(RND_IFD || RND_ICC || K_IFD) = 17 FC F7 A7 7B B6 8E 85 E1 00 F9 B4 4A 87 71 7C
                                           37 66 1B 65 BA D1 2F 0D 67 6C 0C 2C B5 D1 EB 4C
                                           86 2B FF 81 71 3C 85 3D B0 0D 8B C5 74 1D 29 A4
>>  00 82 00 01 30 17 FC F7 A7 7B B6 8E 85 E1 00 F9 B4 4A 87 71 7C 37 66 1B 65 BA D1 2F 0D
    67 6C 0C 2C B5 D1 EB 4C 86 2B FF 81 71 3C 85 3D B0 0D 8B C5 74 1D 29 A4 30 – MUTUAL
    AUTHENTICATE
<<  9B 16 44 7F 98 DC BC 83 1B 25 D5 7D 66 60 68 B4 9E 30 61 46 C7 33 40 D0 7A B3 08 C6 60
    71 91 1A D1 EC 4A 7D 5B 9F 4A A5 1B 24 EA 06 69 40 B9 0B 90 00 - OK
CMK.decrypt(chip_response) =  9F 44 39 78 09 B3 C7 E9 E8 81 41 E4 DC A1 99 82 C8 A7 E8 21
                              0F 6D 73 07 73 5A 80 77 CD A7 F9 A5 27 1A B4 0E 6C EC 28 35
                              1A AE AB 57 86 7D 99 5E
_RND_ICC: 9F 44 39 78 09 B3 C7 E9
_RND_IFD: E8 81 41 E4 DC A1 99 82
RND_IFD == _RND_IFD (true)
K_ICC:  C8 A7 E8 21 0F 6D 73 07 73 5A 80 77 CD A7 F9 A5 27 1A B4 0E 6C EC 28 35 1A AE AB 57
        86 7D 99 5E
SK = K_IFD XOR _K_ICC:  46 28 5C 82 90 A5 5E 91 09 F1 40 CA 70 2A 7C F0 A2 13 C0 A8 95 40
                        FA 6E 1D 12 B6 B0 EB 92 E5 E0
SSC = RND_IFD[4..7] || RND_ICC[4..7]: DC A1 99 82 09 B3 C7 E9
SK1 = SK[0..15]: 46 28 5C 82 90 A5 5E 91 09 F1 40 CA 70 2A 7C F0
SK2 = SK[16..31]: A2 13 C0 A8 95 40 FA 6E 1D 12 B6 B0 EB 92 E5 E0
## Mutual authentication successful ##

// Secure the command
CLA = CLA | 0C: 0C
SSC(DC A1 99 82 09 B3 C7 E9) + 1 = DC A1 99 82 09 B3 C7 EA
Data = Data || ISO9797 method 2 padding:   31 32 33 34 31 32 33 34 35 31 32 33 34 35 36 37
                                           38 80 00 00 00 00 00 00
Cryptogram = SK1.encrypt(data, IV(SSC)) =  07 35 7E 32 CF 2C 41 D4 3B 62 06 64 84 02 DF C8
                                           5A BC 3A DA F0 20 84 8C
// Prepare MACData
    Append header: 0C 05 00 00
```

```
     Append 80000000: 0C 05 00 00 80 00 00 00
     Wrap Cryptogram into TLV with tag 87.
     Data = Tag(87) || Length || Value(Cryptogram):   87 19 01 07 35 7E 32 CF 2C 41 D4 3B
                                                      62 06 64 84 02 DF C8 5A BC 3A DA F0
                                                      20 84 8C
     MACData = Append ISO9797 method 2 padding:  0C 05 00 00 80 00 00 00 87 19 01 07 35 7E
                                                 32 CF 2C 41 D4 3B 62 06 64 84 02 DF C8 5A
                                                 BC 3A DA F0 20 84 8C 80 00 00 00 00
// Calculate MAC
SK2Key1 = SK2[0..7]: A2 13 C0 A8 95 40 FA 6E
SK2Key1 = SK2[8..15]: 1D 12 B6 B0 EB 92 E5 E0
MAC = SK2Key1.CBC_encrypt(MACData, IV(SSC)): 50 12 FB 97 30 08 B8 DE
MAC = SK2Key2.decrypt(MAC): 52 AF B4 E2 12 95 F1 5B
MAC = SK2Key1.encrypt(MAC): 12 6E F1 7F 76 A3 E1 1A
// Wrap MAC into TLV with tag 8E.
     MAC = Tag(8E) || Length || Value(MAC):  87 19 01 07 35 7E 32 CF 2C 41 D4 3B 62 06 64
                                             84 02 DF C8 5A BC 3A DA F0 20 84 8C 8E 08 12
                                             6E F1 7F 76 A3 E1 1A
// Append MAC to Data
Data = Data || MAC:  87 19 01 07 35 7E 32 CF 2C 41 D4 3B 62 06 64 84 02 DF C8 5A BC 3A DA
                     F0 20 84 8C 8E 08 12 6E F1 7F 76 A3 E1 1A
>>   0C 05 00 00 25 87 19 01 07 35 7E 32 CF 2C 41 D4 3B 62 06 64 84 02 DF C8 5A BC 3A DA F0
     20 84 8C 8E 08 12 6E F1 7F 76 A3 E1 1A 00 – SECURE REPLACE PINS
<<   99 02 90 00 8E 08 55 9D 67 F4 99 C0 27 D3 90 00 - OK
// Verify MAC and decrypt
SSC(DC A1 99 82 09 B3 C7 EA) + 1 = DC A1 99 82 09 B3 C7 EB
MAC:  55 9D 67 F4 99 C0 27 D3
// Prepare MACData
     Append R-APDU data without MAC TLV(8E): 99 02 90 00
     MACData = Append ISO9797 method 2 padding: 99 02 90 00 80 00 00 00
_MAC = SK2Key1.CBC_encrypt(MACData, IV(SSC)): A2 00 5D F8 A0 31 1E 44
_MAC = SK2Key2.decrypt(_MAC): 6B 80 A2 F3 75 23 33 BF
_MAC = SK2Key1.encrypt(_MAC): 55 9D 67 F4 99 C0 27 D3
MAC == _MAC (true)
```

Generate new key pair

```
// Use master CMK_KEY
// Calulate cardholder CMK. “47101010033” as seed, from EF 5044 record 7.
     SHA1("47101010033") = 74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36 23 3A B3 35 37 0C
     Take 16 leftmost bytes of calulated SHA1:  74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36
                                                23 3A
     CMK.encrypt(74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36 23 3A) =  BA F8 F0 01 7A 4E 9A 39
                                                                     47 38 47 24 6D FE 89 B5
     Set off LSB bits of every byte: BAF8F0007A4E9A38463846246CFE88B4 (cardholder CMK)
>>   00 84 00 00 08 – GET CHALLENGE
<<   CB 14 A4 85 D5 D7 36 4F 90 00 - OK
RND_ICC: CB 14 A4 85 D5 D7 36 4F
Generate RND_IFD: A0 A0 33 46 49 2E DD 66
```

```
Generate K_IFD:   65 E3 90 DD AA 37 87 AB C4 C9 56 52 4E 89 BD 29 3C C9 17 94 20 5C B1 7C
                  A2 BC 31 73 50 58 20 0E
CMK.encrypt(RND_IFD || RND_ICC || K_IFD) =  EA 02 3F A9 68 2B 80 28 70 F6 FC 68 C6 EE A2
                                            11 23 04 47 86 E2 7D 52 D9 21 81 B2 FB A0 78
                                            B3 8C F5 1C F9 5B D8 01 1C 26 20 39 9D 32 96
                                            9D 42 1A
>>   00 82 00 02 30 EA 02 3F A9 68 2B 80 28 70 F6 FC 68 C6 EE A2 11 23 04 47 86 E2 7D 52 D9
     21 81 B2 FB A0 78 B3 8C F5 1C F9 5B D8 01 1C 26 20 39 9D 32 96 9D 42 1A 30 – MUTUAL
     AUTHENTICATE
<<   36 42 B0 5B AC 9A A0 B1 E4 DC E7 51 22 5E 92 2D 1F C7 C0 C8 37 CE E2 7B 25 6E 91 7B A8
     71 95 39 71 17 26 40 7A 4F 57 0D AD 6D 99 95 F6 FF D3 90 90 00 - OK
CMK.decrypt(chip_response) =  CB 14 A4 85 D5 D7 36 4F A0 A0 33 46 49 2E DD 66 20 DA F3 96
                              1D 48 71 9F 46 62 AF B9 DB BD B8 3D C2 BF AF CC 92 D2 A8 D9
                              A0 41 C7 A6 80 A7 BE 33
_RND_ICC: CB 14 A4 85 D5 D7 36 4F
_RND_IFD: A0 A0 33 46 49 2E DD 66
RND_IFD == _RND_IFD (true)
K_ICC:    20 DA F3 96 1D 48 71 9F 46 62 AF B9 DB BD B8 3D C2 BF AF CC 92 D2 A8 D9 A0 41 C7
          A6 80 A7 BE 33
SK = K_IFD XOR _K_ICC:  45 39 63 4B B7 7F F6 34 82 AB F9 EB 95 34 05 14 FE 76 B8 58 B2 8E
                        19 A5 02 FD F6 D5 D0 FF 9E 3D
SSC = RND_IFD[4..7] || RND_ICC[4..7]: 49 2E DD 66 D5 D7 36 4F
SK1 = SK[0..15]: 45 39 63 4B B7 7F F6 34 82 AB F9 EB 95 34 05 14
SK2 = SK[16..31]: FE 76 B8 58 B2 8E 19 A5 02 FD F6 D5 D0 FF 9E 3D
## Mutual Authentication successful ##

>>   00 20 00 01 04 31 32 33 34 00 – VERIFY (PIN1)
<<   90 00 - OK
// Secure the command
CLA = CLA | 0C: 0C
SSC(49 2E DD 66 D5 D7 36 4F) + 1 = 49 2E DD 66 D5 D7 36 50
// Prepare MACData
     Append header: 0C 06 01 01
     Append 80000000: 0C 06 01 01 80 00 00 00
     MACData = Append ISO9797 method 2 padding:  0C 06 01 01 80 00 00 00 80 00 00 00 00 00
                                                 00 00
// Calculate MAC
SK2Key1 = SK2[0..7]: FE 76 B8 58 B2 8E 19 A5
SK2Key1 = SK2[8..15]: 02 FD F6 D5 D0 FF 9E 3D
MAC = SK2Key1.CBC_encrypt(MACData, IV(SSC)): 7B D7 BE DA 02 92 AA A1
MAC = SK2Key2.decrypt(MAC): 63 C1 07 CE BF 19 2C C6
MAC = SK2Key1.encrypt(MAC): CF 25 E9 28 D3 35 48 8A
// Wrap MAC into TLV with tag 8E.
     MAC = Tag(8E) || Length || Value(MAC): 8E 08 CF 25 E9 28 D3 35 48 8A
Data = Data || MAC: 8E 08 CF 25 E9 28 D3 35 48 8A
>>   0C 06 01 01 0A 8E 08 CF 25 E9 28 D3 35 48 8A 00 – SECURE GENERATE KEY
<<   87 82 01 10 9E 69 28 BA 46 0D 8C DD AF F5 F4 6F 45 6F 57 1E A1 79 56 5C F3 C4 4C 44 55
     2A D0 0F 37 75 3D D6 44 1E 2B 67 8F AA 86 17 DC A2 C8 3B 8D 87 ED 84 63 0A 07 A0 28 6F
     9E 93 56 33 CB 84 BD 88 67 22 05 CA 81 BB C1 32 92 F6 18 4E D5 B4 EB 56 99 D7 3A 2F DF
     09 75 81 F0 EB DF 1D F9 C6 E5 68 4B FA 26 06 70 57 C0 7B 6C C7 F2 03 F9 99 5A 38 0B 4A
```

```
    21 CA BB EE FA 2E 10 83 65 FD 52 A0 47 A1 81 A6 B9 37 31 C4 6D F4 FC 89 A2 F8 A4 81 24
    00 05 06 9C CF 94 5C B8 3D 2E 4D AA 1B 80 4F 7B EC 78 A4 97 B7 64 16 33 9D 9F FA D1 F8
    31 39 72 AB 25 EC 11 A7 89 2A 1A 9D 88 E1 1B C6 2B 1D 92 8B 49 C1 35 E0 8F A1 67 25 0F
    95 6D A3 75 39 EE A7 B3 A4 56 C6 94 0B 4E 3B 25 23 01 32 5D A5 27 CF 3D 2F 7F 72 6E 04
    66 DD 8D 33 D5 86 BC 0A D0 42 43 76 F5 13 A3 75 AC 38 73 E5 A9 7A 1F AF CC 08 D6 CC 1C
    98 0F 03 17 5F 07 2C 0F 65 40 BA 1F 40 BB 97 8E 08 83 CF C3 0D F4 3C 83 86 90 00 - OK
// Verify MAC and decrypt
SSC(49 2E DD 66 D5 D7 36 50) + 1 = 49 2E DD 66 D5 D7 36 51
MAC: 83CFC30DF43C8386
// Prepare MACData
     Append R-APDU data without MAC TLV(8E):
               87 82 01 10 9E 69 28 BA 46 0D 8C DD AF F5 F4 6F 45 6F 57 1E A1 79 56 5C F3
               C4 4C 44 55 2A D0 0F 37 75 3D D6 44 1E 2B 67 8F AA 86 17 DC A2 C8 3B 8D 87
               ED 84 63 0A 07 A0 28 6F 9E 93 56 33 CB 84 BD 88 67 22 05 CA 81 BB C1 32 92
               F6 18 4E D5 B4 EB 56 99 D7 3A 2F DF 09 75 81 F0 EB DF 1D F9 C6 E5 68 4B FA
               26 06 70 57 C0 7B 6C C7 F2 03 F9 99 5A 38 0B 4A 21 CA BB EE FA 2E 10 83 65
               FD 52 A0 47 A1 81 A6 B9 37 31 C4 6D F4 FC 89 A2 F8 A4 81 24 00 05 06 9C CF
               94 5C B8 3D 2E 4D AA 1B 80 4F 7B EC 78 A4 97 B7 64 16 33 9D 9F FA D1 F8 31
               39 72 AB 25 EC 11 A7 89 2A 1A 9D 88 E1 1B C6 2B 1D 92 8B 49 C1 35 E0 8F A1
               67 25 0F 95 6D A3 75 39 EE A7 B3 A4 56 C6 94 0B 4E 3B 25 23 01 32 5D A5 27
               CF 3D 2F 7F 72 6E 04 66 DD 8D 33 D5 86 BC 0A D0 42 43 76 F5 13 A3 75 AC 38
               73 E5 A9 7A 1F AF CC 08 D6 CC 1C 98 0F 03 17 5F 07 2C 0F 65 40 BA 1F 40 BB
               97
     MACData = Append ISO9797 method 2 padding:
               87 82 01 10 9E 69 28 BA 46 0D 8C DD AF F5 F4 6F 45 6F 57 1E A1 79 56 5C F3
               C4 4C 44 55 2A D0 0F 37 75 3D D6 44 1E 2B 67 8F AA 86 17 DC A2 C8 3B 8D 87
               ED 84 63 0A 07 A0 28 6F 9E 93 56 33 CB 84 BD 88 67 22 05 CA 81 BB C1 32 92
               F6 18 4E D5 B4 EB 56 99 D7 3A 2F DF 09 75 81 F0 EB DF 1D F9 C6 E5 68 4B FA
               26 06 70 57 C0 7B 6C C7 F2 03 F9 99 5A 38 0B 4A 21 CA BB EE FA 2E 10 83 65
               FD 52 A0 47 A1 81 A6 B9 37 31 C4 6D F4 FC 89 A2 F8 A4 81 24 00 05 06 9C CF
               94 5C B8 3D 2E 4D AA 1B 80 4F 7B EC 78 A4 97 B7 64 16 33 9D 9F FA D1 F8 31
               39 72 AB 25 EC 11 A7 89 2A 1A 9D 88 E1 1B C6 2B 1D 92 8B 49 C1 35 E0 8F A1
               67 25 0F 95 6D A3 75 39 EE A7 B3 A4 56 C6 94 0B 4E 3B 25 23 01 32 5D A5 27
               CF 3D 2F 7F 72 6E 04 66 DD 8D 33 D5 86 BC 0A D0 42 43 76 F5 13 A3 75 AC 38
               73 E5 A9 7A 1F AF CC 08 D6 CC 1C 98 0F 03 17 5F 07 2C 0F 65 40 BA 1F 40 BB
               97 80 00 00 00
_MAC = SK2Key1.CBC_encrypt(MACData, IV(SSC)): F7 0F 61 06 9D FA 04 AF
_MAC = SK2Key2.decrypt(_MAC): 68 95 FF 72 9E BD 89 D1
_MAC = SK2Key1.encrypt(_MAC): 83 CF C3 0D F4 3C 83 86
MAC == _MAC (true)
Unwrap cryptogram from TLV: 9E 69 28 BA 46 0D 8C DD AF F5 F4 6F 45 6F 57 1E A1 79 56 5C F3
    C4 4C 44 55 2A D0 0F 37 75 3D D6 44 1E 2B 67 8F AA 86 17 DC A2 C8 3B 8D 87 ED 84 63 0A
    07 A0 28 6F 9E 93 56 33 CB 84 BD 88 67 22 05 CA 81 BB C1 32 92 F6 18 4E D5 B4 EB 56 99
    D7 3A 2F DF 09 75 81 F0 EB DF 1D F9 C6 E5 68 4B FA 26 06 70 57 C0 7B 6C C7 F2 03 F9 99
    5A 38 0B 4A 21 CA BB EE FA 2E 10 83 65 FD 52 A0 47 A1 81 A6 B9 37 31 C4 6D F4 FC 89 A2
    F8 A4 81 24 00 05 06 9C CF 94 5C B8 3D 2E 4D AA 1B 80 4F 7B EC 78 A4 97 B7 64 16 33 9D
    9F FA D1 F8 31 39 72 AB 25 EC 11 A7 89 2A 1A 9D 88 E1 1B C6 2B 1D 92 8B 49 C1 35 E0 8F
    A1 67 25 0F 95 6D A3 75 39 EE A7 B3 A4 56 C6 94 0B 4E 3B 25 23 01 32 5D A5 27 CF 3D 2F
    7F 72 6E 04 66 DD 8D 33 D5 86 BC 0A D0 42 43 76 F5 13 A3 75 AC 38 73 E5 A9 7A 1F AF CC
    08 D6 CC 1C 98 0F 03 17 5F 07 2C 0F 65 40 BA 1F 40 BB 97
SK1.CBC_decrypt(Data, IV(SSC)) : 7F 49 82 01 0A 81 82 01 00 B6 F3 35 66 79 71 46 9C 7B 55
    9B 37 BB 99 A0 E1 47 F1 B7 2F 79 86 13 FB 1A 39 A3 1D ED 22 0C F1 8A 6C 8E 93 6C DF 0B
    57 7A 57 87 D3 41 B0 22 E2 68 79 45 15 24 E2 E7 F4 C5 FF B9 29 09 BE 5F DA 8D 0B 63 E3
    F8 B1 FD F7 6F 08 5B 6F 54 14 3A 5C D1 58 11 6F BC B8 C5 22 84 8E 11 ED DE 5E 20 45 39
    93 1E FB 5E 0B F6 95 BC F6 AB 66 E8 60 A4 AB 16 D4 D4 99 66 4A BD D6 84 AE 42 62 83 A9
    4B D5 5F 94 D1 F5 B3 2A 62 21 1B 18 6B 1A 0E CC CF 76 37 B5 10 E0 99 FD 62 86 F9 F3 AA
    1F 3F F6 44 71 FE 27 63 95 97 3E 18 D1 4F D3 4C 4A 94 8A 71 0F 9D 18 C3 EB 60 E5 49 C6
```

```
    43 F5 65 9A 85 6F 95 AF 04 0B F4 B2 36 A1 EE E7 73 CB 96 F9 F1 E9 99 C2 C2 24 C7 E0 49
    17 16 12 9F 70 F3 89 2C 98 6D 67 06 7B 37 24 C4 07 C6 F5 11 BC BC A7 28 63 7C FC 0D FA
    70 3B E5 6F 81 91 A7 E2 E1 6E 96 C6 61 A9 82 04 40 00 00 81 80
// Secure command.
CLA = CLA | 0C: 0C
SSC(49 2E DD 66 D5 D7 36 51) + 1 = 49 2E DD 66 D5 D7 36 52
// Prepare MACData
    Append header: 0C 06 01 02
    Append 80000000: 0C 06 01 02 80 00 00 00
    MACData = Append ISO9797 method 2 padding:
              0C 06 01 02 80 00 00 00 80 00 00 00 00 00 00 00
// Calculate MAC
SK2Key1 = SK2[0..7]: 46 28 5C 82 90 A5 5E 91 09 F1 40 CA 70 2A 7C F0
SK2Key1 = SK2[8..15]: A2 13 C0 A8 95 40 FA 6E 1D 12 B6 B0 EB 92 E5 E0
MAC = SK2Key1.CBC_encrypt(MACData, IV(SSC)): 32 D6 33 FB E1 05 F2 B3
MAC = SK2Key2.decrypt(MAC): 59 1B 00 D3 E5 31 2D F8
MAC = SK2Key1.encrypt(MAC): A9 3D 4B F1 7C 1B AE F0
// Wrap MAC into TLV with tag 8E.
    MAC = Tag(8E) || Length || Value(MAC): 8E 08 A9 3D 4B F1 7C 1B AE F0
Data = Data || MAC: 8E 08 A9 3D 4B F1 7C 1B AE F0
>>  0C 06 01 02 0A 8E 08 A9 3D 4B F1 7C 1B AE F0 00 – SECURE GENERATE KEY
<<  87 82 01 10 F5 03 50 13 78 DC 4C 58 05 32 F4 35 D1 3A 17 7F 4D 3D 76 F2 17 C3 E4 7D 69
    07 D0 FD F5 CF DA C2 5F 30 9E E3 CA 20 BF AE AF 99 53 2A BA 43 3B 82 BF C7 70 A7 CB 31
    09 E3 8C 35 9F 91 BA 1A 05 4E 36 36 A2 E4 E3 20 31 90 89 45 2E CB C5 9F 25 00 7F 06 28
    0C 5A D1 07 DB DC 10 A3 9D 4C 35 ED 8E 1E 4A 62 70 DB EB 20 F6 82 41 FF C9 A0 DC 56 12
    90 B3 B5 F6 FD 78 DE 4E 90 D5 A4 D1 7E E2 1E 12 E1 73 5E 2C AA 22 EE F4 E4 9D 95 F9 85
    58 1A 18 C9 7B 60 4D F5 85 FB 2C 0C 14 A4 B3 A7 C8 ED D4 75 12 F1 8A 01 E8 67 1C 4A 30
    1A 44 84 6E 20 61 A1 CA B2 8B 0F C9 9B C0 DE 2B F0 C4 0C 73 E1 38 44 A2 DE 0D DC 70 B1
    00 7A D1 D8 B0 A4 8A 58 6F 5F 1B 56 1E 26 4B 85 9C 78 CF 58 01 18 08 23 23 AC 23 BE 3D
    DC 97 3A 55 52 F3 D7 A7 8F 2C AD 2F 20 48 1E 7A 71 3E D2 C6 65 0F 16 23 75 F6 A9 EF CB
    9A 56 C7 B0 B9 81 EE B8 7F 96 75 92 64 33 05 8E 08 2C 8F 7B F2 A9 17 70 48 90 00 - OK
// Verify MAC and decrypt
SSC(49 2E DD 66 D5 D7 36 52) + 1 = 49 2E DD 66 D5 D7 36 53
// Prepare MACData
    Append R-APDU data without MAC TLV(8E):
              87 82 01 10 F5 03 50 13 78 DC 4C 58 05 32 F4 35 D1 3A 17 7F 4D 3D 76 F2 17
              C3 E4 7D 69 07 D0 FD F5 CF DA C2 5F 30 9E E3 CA 20 BF AE AF 99 53 2A BA 43
              3B 82 BF C7 70 A7 CB 31 09 E3 8C 35 9F 91 BA 1A 05 4E 36 36 A2 E4 E3 20 31
              90 89 45 2E CB C5 9F 25 00 7F 06 28 0C 5A D1 07 DB DC 10 A3 9D 4C 35 ED 8E
              1E 4A 62 70 DB EB 20 F6 82 41 FF C9 A0 DC 56 12 90 B3 B5 F6 FD 78 DE 4E 90
              D5 A4 D1 7E E2 1E 12 E1 73 5E 2C AA 22 EE F4 E4 9D 95 F9 85 58 1A 18 C9 7B
              60 4D F5 85 FB 2C 0C 14 A4 B3 A7 C8 ED D4 75 12 F1 8A 01 E8 67 1C 4A 30 1A
              44 84 6E 20 61 A1 CA B2 8B 0F C9 9B C0 DE 2B F0 C4 0C 73 E1 38 44 A2 DE 0D
              DC 70 B1 00 7A D1 D8 B0 A4 8A 58 6F 5F 1B 56 1E 26 4B 85 9C 78 CF 58 01 18
              08 23 23 AC 23 BE 3D DC 97 3A 55 52 F3 D7 A7 8F 2C AD 2F 20 48 1E 7A 71 3E
              D2 C6 65 0F 16 23 75 F6 A9 EF CB 9A 56 C7 B0 B9 81 EE B8 7F 96 75 92 64 33
              05
    MACData = Append ISO9797 method 2 padding:
              87 82 01 10 F5 03 50 13 78 DC 4C 58 05 32 F4 35 D1 3A 17 7F 4D 3D 76 F2 17
              C3 E4 7D 69 07 D0 FD F5 CF DA C2 5F 30 9E E3 CA 20 BF AE AF 99 53 2A BA 43
              3B 82 BF C7 70 A7 CB 31 09 E3 8C 35 9F 91 BA 1A 05 4E 36 36 A2 E4 E3 20 31
              90 89 45 2E CB C5 9F 25 00 7F 06 28 0C 5A D1 07 DB DC 10 A3 9D 4C 35 ED 8E
              1E 4A 62 70 DB EB 20 F6 82 41 FF C9 A0 DC 56 12 90 B3 B5 F6 FD 78 DE 4E 90
              D5 A4 D1 7E E2 1E 12 E1 73 5E 2C AA 22 EE F4 E4 9D 95 F9 85 58 1A 18 C9 7B
```

```
                60 4D F5 85 FB 2C 0C 14 A4 B3 A7 C8 ED D4 75 12 F1 8A 01 E8 67 1C 4A 30 1A
                44 84 6E 20 61 A1 CA B2 8B 0F C9 9B C0 DE 2B F0 C4 0C 73 E1 38 44 A2 DE 0D
                DC 70 B1 00 7A D1 D8 B0 A4 8A 58 6F 5F 1B 56 1E 26 4B 85 9C 78 CF 58 01 18
                08 23 23 AC 23 BE 3D DC 97 3A 55 52 F3 D7 A7 8F 2C AD 2F 20 48 1E 7A 71 3E
                D2 C6 65 0F 16 23 75 F6 A9 EF CB 9A 56 C7 B0 B9 81 EE B8 7F 96 75 92 64 33
                05 80 00 00 00
MAC: 2C 8F 7B F2 A9 17 70 48
_MAC = SK2Key1.CBC_encrypt(MACData, IV(SSC)): 7D5063D5AC000BE4
_MAC = SK2Key2.decrypt(_MAC): 6A1062B065531D70
_MAC = SK2Key1.encrypt(_MAC): 2C8F7BF2A9177048
MAC == _MAC (true)
Unwrap cryptogram from TLV:
    F5 03 50 13 78 DC 4C 58 05 32 F4 35 D1 3A 17 7F 4D 3D 76 F2 17 C3 E4 7D 69 07 D0 FD F5
    CF DA C2 5F 30 9E E3 CA 20 BF AE AF 99 53 2A BA 43 3B 82 BF C7 70 A7 CB 31 09 E3 8C 35
    9F 91 BA 1A 05 4E 36 36 A2 E4 E3 20 31 90 89 45 2E CB C5 9F 25 00 7F 06 28 0C 5A D1 07
    DB DC 10 A3 9D 4C 35 ED 8E 1E 4A 62 70 DB EB 20 F6 82 41 FF C9 A0 DC 56 12 90 B3 B5 F6
    FD 78 DE 4E 90 D5 A4 D1 7E E2 1E 12 E1 73 5E 2C AA 22 EE F4 E4 9D 95 F9 85 58 1A 18 C9
    7B 60 4D F5 85 FB 2C 0C 14 A4 B3 A7 C8 ED D4 75 12 F1 8A 01 E8 67 1C 4A 30 1A 44 84 6E
    20 61 A1 CA B2 8B 0F C9 9B C0 DE 2B F0 C4 0C 73 E1 38 44 A2 DE 0D DC 70 B1 00 7A D1 D8
    B0 A4 8A 58 6F 5F 1B 56 1E 26 4B 85 9C 78 CF 58 01 18 08 23 23 AC 23 BE 3D DC 97 3A 55
    52 F3 D7 A7 8F 2C AD 2F 20 48 1E 7A 71 3E D2 C6 65 0F 16 23 75 F6 A9 EF CB 9A 56 C7 B0
    B9 81 EE B8 7F 96 75 92 64 33 05
SK1.CBC_decrypt(Data, IV(SSC)):
    7F 49 82 01 0A 81 82 01 00 93 C3 F0 E3 A9 B7 A0 AB 65 6D 15 D0 ED E0 C7 B7 20 06 12 B1
    4C 71 6B FE 42 10 77 79 F7 79 C7 B9 D3 58 A7 6B 3A CE 56 99 92 6D 82 E7 52 82 0E 8A 05
    28 60 B1 9A B8 87 96 93 AD 57 64 5F F6 1D FD 11 4D 5D B2 F5 55 F7 2A E4 AF 47 58 BE 2F
    DA 9F 06 CA D1 EB 68 73 C7 A1 31 A1 52 0C 7B C5 50 75 18 80 BE 52 1A A4 2C D3 03 08 64
    FA 17 19 96 DA 0A 0C 24 4C ED AB CC 60 96 5C CB F6 D7 5C 74 63 D8 97 3B E4 23 84 C3 61
    92 E3 5C 35 4D 6C 97 14 81 F1 25 2C 92 FB 42 08 13 73 7C 8B D3 11 4B 47 4C 04 97 B0 DB
    BE E3 68 84 A8 C8 9D 0D B7 36 99 4B 4C DF 2F 28 6D 11 42 B6 4B FD D1 E4 07 B1 D7 24 D8
    BA 5E 58 B9 CE 38 D9 C4 15 1A A0 4C 4D E4 54 72 78 66 1C DF 13 6B 3B BB C7 55 C0 70 96
    F6 ED 5E 4E 2C 4B 8A CC C2 1B 62 FC E5 FC B1 A8 83 6D 6D B9 9D 51 AD D3 24 E7 88 D4 23
    5E 7F B3 E3 82 04 40 00 00 81 80
```

Replace Certificates

```
// Use master CMK_CERT
// Calulate cardholder CMK. “47101010033” as seed, from EF 5044 record 7.
    SHA1("47101010033") = 74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36 23 3A B3 35 37 0C
    Take 16 leftmost bytes of calulated SHA1:  74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36
                                               23 3A
    CMK.encrypt(74 B5 97 30 6B C5 9B 67 2F B1 64 B2 0F 36 23 3A) =  82 9C AD 1E DE DB 26 91
                                                                    BB 88 58 77 58 48 BA DC
    Set off LSB bits of every byte: 829CAC1EDEDA2690BA8858765848BADC (cardholder CMK)
>>  00 84 00 00 08 – GET CHALLENGE
<<  43 71 27 41 6C EB FD 57 90 00 - OK
RND_ICC: 43 71 27 41 6C EB FD 57
Generate RND_IFD: 4B 08 36 AB 4F 74 3D C4
Generate K_IFD: 11 719AFF52B5000AC129843EA94F630BCC3C5553CE775B1F05453ED3825936A0
CMK.encrypt(RND_IFD || RND_ICC || K_IFD) =
    9DC09142926D08A15E16053EC47024788358E863EACF22F139AA94746604F1AB41E7091E17A7F4EE69C51A
    DA70A9027D
>>  00 82 00 03 30 9D C0 91 42 92 6D 08 A1 5E 16 05 3E C4 70 24 78 83 58 E8 63 EA CF 22 F1
    39 AA 94 74 66 04 F1 AB 41 E7 09 1E 17 A7 F4 EE 69 C5 1A DA 70 A9 02 7D 30 – MUTUAL
    AUTHENTICATE
```

```
<<   3D 66 9C 23 BA 43 F1 B1 E9 9D D3 86 F1 6C 83 24 35 2B B3 77 67 13 24 45 BB CD D9 98 8A
     27 11 59 4A 38 2E 29 C6 70 9C B9 98 4F 6E 3E F9 EA C1 8B 90 00 - OK
CMK.decrypt(chip_response) =   43 71 27 41 6C EB FD 57 4B 08 36 AB 4F 74 3D C4 0C 21 2D B1
                               AF F8 88 D0 9D 31 A5 3F 3A E8 03 36 12 73 C5 AD 58 8D 79 D5
                               97 13 B7 53 3B 28 C2 41
_RND_ICC: 437127416CEBFD57
_RND_IFD: 4B0836AB4F743DC4
RND_IFD == _RND_IFD (true)
K_ICC: 0C212DB1AFF888D09D31A53F3AE803361273C5AD588D79D59713B7533B28C241
SK = K_IFD XOR _K_ICC : 1D50B74EFD4D88DA5C18210193A7603DDE4F90FE96FA22CA92568980B971F4E1
SSC = RND_IFD[4..7] || RND_ICC[4..7]: 4F743DC46CEBFD57
SK1 = SK[0..15] : 1D 50 B7 4E FD 4D 88 DA 5C 18 21 01 93 A7 60 3D
SK2 = SK[16..31] : DE 4F 90 FE 96 FA 22 CA 92 56 89 80 B9 71 F4 E1
## Mutual Authentication successful ##

>>   00 20 00 01 04 31 32 33 34 00
<<   90 00
// Secure the command
CLA = CLA | 0C: 0C
SSC(4F743DC46CEBFD57) + 1 = 4F743DC46CEBFD58
Data = Data || ISO9797 method 2 padding:
     3082053C30820424A003020102021002EF829DE1979694501F7379B66E05DA300D06092A864886F70D0101
     050500306C310B300906035504061302454531223020660355040A0C19415320536572746966697473656 5
     72696D69736B65736B7573311F301D06035504030C1654455354206F662045535445494 42D534B20323031
     313118301606092A864886F70D0109011609706B6940736B2E6565301E170D313230383036303733343137
     5A170D313730323031323135395395A30819B310B300906035504061302454531 0F300D060355040A0C06
     455354454944311730150606035504 0B0C0E61757468656E7469636174696F6E312630240603550403 0C1D4D
     C3844E4E494B2C4D4152492D4C4949532C343731303130313030333331103 00E06035504040C074DC3844E
     4E494B31123010060355042A0C094D4152492D4C494953311430120603550405130B343731303130313030
     333330820123300D06092A864886F70D01010105000382011000308201 0B028201010091B90AF54BE514C1
     CDC08F507C39BCD28FF9895F1C881B1B85602CCAEDF4651EEA8B254F69AB4C7B221FD8EF492917A4EA4CAD
     C472CCAF5C0D16C4022CD0AD1A24BE6846BEC29A743D6C6681697CD45CD662648ACD60DC50AE462FED35D0
     D5EE84FC12CEEA93ED8B658DBC113339F1CBC2DF9D765548F7A15452C64862E59351E61C9F9B9FD677571A
     946BC623FD90AD5D0C9E03DEE6C24FE994B7BDC0D4304F2A1204AFBEBD1BD470CFBBC638427191ECE6D6BF
     72A4559858054E80BE2C631A56955640F1F12072A4EEB399C9E78B6F7A4FFEEE5C8B5F3AF41BD048DB6B8D
     84828894F938D3A92DF244219C987441472C0B4E1D01F9EB55112F9A127A2B064D02047E0B8371A38201A7
     308201A330090603551D1304023000300E0603551D0F0101FF0404030204B03081990603551D2004819130
     818E30818B060A2B06010401CE1F030101307D305806082B06010505070202304C1E4A00410069006E0075
     006C00740020007400650073007400690060D00690073006500 6B0073002E0020004F006E006C0079002000
     66006F00720020007400650073007400690060E0067002E302106082B06010505070201161568747 4703A2F
     2F7777772E736B2E65652F6370732F30270603551D110420301E811C6D6172692D6C6969732E6D616E6E69
     6B2E313440656573746 92E6565301D0603551D0E04160414546A429EDFB4883378854A709A84ADEED64C5F
     1A302006035 51D250101FF041630140608 2B060105050703020608 2B060105050703043018060 82B060105
     05070103040C300A300806060400 8E460101301F0603551D230418301680144 1B6FEC5B1B1B453138CFAFA
     62D0346D6D22340A30450603551D1F043E303C303AA038A0368634687474703A2F2F7777772E736B2E6565
     2F7265706F7369746F72792F63726C732F746573745F6573746569643230313 12E63726C300D06092A8648
     86F70D0101050500038201010 05DE216914DA46C244199F45BEE3BB2F1609AEB774DEE3C2CB8B36F005D2E
     25F05DE7D2C67342024E31E9DAB268776639FB9927E9F85823E3FA4614540E02F7933D0E9A58E184E86DD8
     26A770F7C7DAB6EB6B6521FE9C45BDE3279E4EA4F50227D317BC77D57B4D8A271DEB1852A302682E5D9A00
     6D6A8550F952A68EB878D49805B63369E8A7CDFB00DE4A124B627AE8294CBC32633E20A1DC68D9AAB9DA25
     19FE7683EA7666ACCB6614F5B55B328306F8F8C58C2EB7DA6329814946532056 5F1E8FF5414C8C8B77292F
     DE675EECAD7E42EEA5814C17BC253DF84323774733B186BBDB7967E4FC6994B5C19A4D720FD828FD61C721
     D2838CF8C64405390AC6258000000000000000000000000000000000000000000000000000000000000000
     0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
     0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    0000000000000000000000000000000000000000000000000000000000000000000000800000000000000000
Cryptogram = SK1.encrypt(data, IV(SSC)):
    81BCCBB003C8C401CA4976AC94AE7032ACAB10E9127D85AF01D02C51B94C71EAAF263DD354DDAFAE4CD362
    FB0DA09B69F0C6C399361AD53FC443FD2CA3C6BEE4DA6B4D1E14195503AA688691826B5AFD59D18794EF80
    27F38B7D3F86A7BCE6182036EBA2BF5423DBC73660702619CD29CB3F7AFA7ACE6963F3DB7958DB6BA1E607
    6A1B56FE205378C33D7F8E284B5315CFDF3F247EB19FB72B5C8189B7500CCB1D31B0AC7DFB236CE33FCAF5
    3038F15F426189225B5FDB876C40C4726935ED4E91B541C7835D8D16B8B5CB342DF4D255BF7C77251B4FE9
    3838267ABB394166C546DAF69ED97E0552225AFF8BF497E21F705635441E06E039331F291B375778699BFB
    96743D16598B26771E382EFE7134AED8AA3ECD6BA0A8CB73C41A4134E99819EB96143644E46023DF41A371
    C6C537173677B9E10CD468604AA697E9B6BF30B9CCCF64AE12A705457DF5FD092D3234E5B3F542D971AD30
    16C997901E58F8CF5A03BA19DECF3C4D48EA1562DEDFA622DCD1D686B26291E1F8EE614110400F114767BC
    04F341444F43F1A6BA934753A33E43BDCCF6B96CFC41B6FCEDADF4EC911912EFB2D37905784B15CA3234D3
    B208D5D9CC351F00196EAF6304CB288047CFFAF5DDA26B087379C60A590C3FFEAED6A0D2D5CD2BFDED5D68
    A53C89B3D8DFAE670D97ED6199151562B8AECF3D96FF80E5E6318C275B105DE8888F8848977346330FCDDC
    D2DEBCF633FC97D3CEF22AFD3A62297902629C466F48C55BFF5B1714F7FE9580FF2D5ED0C9B72307EAB4B6
    8C95E96FEAC99DE1B0C84A9B529D21B4FC97C2C51701A030F72A4DD8B94D374C94C5BBD355EB306DA88796
    360EC1DE7E63078D56C3EFC41A7AE628F272FAE9C2773039C500E5B941F87A9818426A9CE6C607687FA460
    B0193FBDE92C215D4C0BB5C8FD546F64FDB6003BAFB23B1ADE68089CDBD4DCD0E9B964DBE74377510286CF
    FEFE91BA067AEAB78D1833F406AAD1DFD039A3ADD86EEC8606A48A851105F3B8E14102FD93E0237363AFD6
    A06B7642DB63D12FA48256A52DBD9E05BD94FBE9388905B39F9165A1DC54BD60899E39370FB4D792E526B1
    E53DBA0EFEB7DBF2CCB00EC1B2CFDBF55F74F77C59A1B1F8EA225C0D8215EC847231E327F468B17EEFF791
    C9546EF85C77F11E55188D0A7E1784C8E12C1E0EFE168BC1BA3DF7F7E76103EC2D1300C77FB18264D8E242
    940EBDBFF801ADB74EE847436F6B590C0BC033E52DAF20A827967B7F1D9D071FDF5ECA67F7B6259850BDF8
    7F45FEB8272951766CDB4EF92264F0E21D7C7722C66CCE86D4F1A4C5B81F52CCD22D34248F0A2FCFAE8484
    DF00543E3204AACF1E433B13E33D58F453707B6BD149317BA706D5EDC5C354F7435B48029C492788A4BEDB
    BDD70BAF1B40854A304CC91C551C4048B8535C3DE2849FBF82A318DC54E6F8663AFB4E371C29D63614FFF8
    5871179A5DBD3982034E6F9BE60287D9638ED0BCCC6FB99BE3952CCAA506352C83FF2365E71A5568C595BD
    72631E1A343EB6301FAF372C2BC10A4110116CC0009E560785F2253BA46FFA1F57A9CD2CA1097BEB1FF2D0
    0DC8A3AF74075B7E67396E957D3B92F8ABD31E83F8A5D97161C8DE4114C94F8D00714EA193A51384F9E838
    2C4C219420C268A5ADB0E9ECCEBF362AF8D2BA671562BB45315071F9BA89FD24493535A8BD48F5A6C0B580
    8926BD9AA49E879E850E340A154C88F27390E9BC6681AC9B0C78F3AEB9C798ED15616D66AFCA05E7D2F7F2
    B30645BEA6C7131FEC34FDF6C2A19499950FD1FDE8E4ECAC3BFF8959D1E270A47E26495DB0EC65EE8F250B
    8FD5D69FEDDD7DB4D764B1AF15D95F60D41E9FBBEC13E055A6C8E31A3CE170B24CEF0FA0D56AA5BCA2C6A6
    81DC2522C12208D19B54328765E5298F6D305608896D6AE090FC4AE8DD0B3C9A0951B4252BD41A0E7C0FE4
    D6F7223A27C11F96EBCB5DBF361B2F136EF5C58203C5D4D303E6839DF6CC992B59DF0FFF51AC33599451EC
    BBB64A790EA3CB7A31FE6A91B249A37CAA0C34CBCC74968B27DA590AFA22E0C13473353B3D87843347592E
    E9126DE8F8629AC49AB4E0E3CE4B054729B3C8270042D1A2EA733D8C3E0B80584C5F0D999F0620D5CB53F2
    CA9BA50C37872E6EA1A2C099F150847B3A35ADC8B6968A7B2C442696413B632FA7BDC37A68CDB4
// Prepare MACData
    Append header: 0C070100
    Append 80000000: 0C07010080000000
    // Wrap Cryptogram into TLV with tag 87.
    Data = Tag(87) || Length || Value(Cryptogram):
        878206090181BCCBB003C8C401CA4976AC94AE7032ACAB10E9127D85AF01D02C51B94C71EAAF263DD3
        54DDAFAE4CD362FB0DA09B69F0C6C399361AD53FC443FD2CA3C6BEE4DA6B4D1E14195503AA68869182
        6B5AFD59D18794EF8027F38B7D3F86A7BCE6182036EBA2BF5423DBC73660702619CD29CB3F7AFA7ACE
        6963F3DB7958DB6BA1E6076A1B56FE205378C33D7F8E284B5315CFDF3F247EB19FB72B5C8189B7500C
        CB1D31B0AC7DFB236CE33FCAF53038F15F426189225B5FDB876C40C4726935ED4E91B541C7835D8D16
        B8B5CB342DF4D255BF7C77251B4FE93838267ABB394166C546DAF69ED97E0552225AFF8BF497E21F70
        5635441E06E039331F291B375778699BFB96743D16598B26771E382EFE7134AED8AA3ECD6BA0A8CB73
        C41A4134E99819EB96143644E46023DF41A371C6C537173677B9E10CD468604AA697E9B6BF30B9CCCF
        64AE12A705457DF5FD092D3234E5B3F542D971AD3016C997901E58F8CF5A03BA19DECF3C4D48EA1562
        DEDFA622DCD1D686B26291E1F8EE614110400F114767BC04F341444F43F1A6BA934753A33E43BDCCF6
        B96CFC41B6FCEDADF4EC911912EFB2D37905784B15CA3234D3B208D5D9CC351F00196EAF6304CB2880
        47CFFAF5DDA26B087379C60A590C3FFEAED6A0D2D5CD2BFDED5D68A53C89B3D8DFAE670D97ED619915
        1562B8AECF3D96FF80E5E6318C275B105DE8888F8848977346330FCDDCD2DEBCF633FC97D3CEF22AFD
        3A62297902629C466F48C55BFF5B1714F7FE9580FF2D5ED0C9B72307EAB4B68C95E96FEAC99DE1B0C8
        4A9B529D21B4FC97C2C51701A030F72A4DD8B94D374C94C5BBD355EB306DA88796360EC1DE7E63078D
```

56C3EFC41A7AE628F272FAE9C2773039C500E5B941F87A9818426A9CE6C607687FA460B0193FBDE92C
215D4C0BB5C8FD546F64FDB6003BAFB23B1ADE68089CDBD4DCD0E9B964DBE74377510286CFFEFE91BA
067AEAB78D1833F406AAD1DFD039A3ADD86EEC8606A48A851105F3B8E14102FD93E0237363AFD6A06B
7642DB63D12FA48256A52DBD9E05BD94FBE9388905B39F9165A1DC54BD60899E39370FB4D792E526B1
E53DBA0EFEB7DBF2CCB00EC1B2CFDBF55F74F77C59A1B1F8EA225C0D8215EC847231E327F468B17EEF
F791C9546EF85C77F11E55188D0A7E1784C8E12C1E0EFE168BC1BA3DF7F7E76103EC2D1300C77FB182
64D8E242940EBDBFF801ADB74EE847436F6B590C0BC033E52DAF20A827967B7F1D9D071FDF5ECA67F7
B6259850BDF87F45FEB8272951766CDB4EF92264F0E21D7C7722C66CCE86D4F1A4C5B81F52CCD22D34
248F0A2FCFAE8484DF00543E3204AACF1E433B13E33D58F453707B6BD149317BA706D5EDC5C354F743
5B48029C492788A4BEDBBDD70BAF1B40854A304CC91C551C4048B8535C3DE2849FBF82A318DC54E6F8
663AFB4E371C29D63614FFF85871179A5DBD3982034E6F9BE60287D9638ED0BCCC6FB99BE3952CCAA5
06352C83FF2365E71A5568C595BD72631E1A343EB6301FAF372C2BC10A4110116CC0009E560785F225
3BA46FFA1F57A9CD2CA1097BEB1FF2D00DC8A3AF74075B7E67396E957D3B92F8ABD31E83F8A5D97161
C8DE4114C94F8D00714EA193A51384F9E8382C4C219420C268A5ADB0E9ECCEBF362AF8D2BA671562BB
45315071F9BA89FD24493535A8BD48F5A6C0B5808926BD9AA49E879E850E340A154C88F27390E9BC66
81AC9B0C78F3AEB9C798ED15616D66AFCA05E7D2F7F2B30645BEA6C7131FEC34FDF6C2A19499950FD1
FDE8E4ECAC3BFF8959D1E270A47E26495DB0EC65EE8F250B8FD5D69FEDDD7DB4D764B1AF15D95F60D4
1E9FBBEC13E055A6C8E31A3CE170B24CEF0FA0D56AA5BCA2C6A681DC2522C12208D19B54328765E529
8F6D305608896D6AE090FC4AE8DD0B3C9A0951B4252BD41A0E7C0FE4D6F7223A27C11F96EBCB5DBF36
1B2F136EF5C58203C5D4D303E6839DF6CC992B59DF0FFF51AC33599451ECBBB64A790EA3CB7A31FE6A
91B249A37CAA0C34CBCC74968B27DA590AFA22E0C13473353B3D87843347592EE9126DE8F8629AC49A
B4E0E3CE4B054729B3C8270042D1A2EA733D8C3E0B80584C5F0D999F0620D5CB53F2CA9BA50C37872E
6EA1A2C099F150847B3A35ADC8B6968A7B2C442696413B632FA7BDC37A68CDB4

MACData = Append ISO9797 method 2 padding:

0C07010080000000878206090181BCCBB003C8C401CA4976AC94AE7032ACAB10E9127D85AF01D02C51
B94C71EAAF263DD354DDAFAE4CD362FB0DA09B69F0C6C399361AD53FC443FD2CA3C6BEE4DA6B4D1E14
195503AA688691826B5AFD59D18794EF8027F38B7D3F86A7BCE6182036EBA2BF5423DBC73660702619
CD29CB3F7AFA7ACE6963F3DB7958DB6BA1E6076A1B56FE205378C33D7F8E284B5315CFDF3F247EB19F
B72B5C8189B7500CCB1D31B0AC7DFB236CE33FCAF53038F15F426189225B5FDB876C40C4726935ED4E
91B541C7835D8D16B8B5CB342DF4D255BF7C77251B4FE93838267ABB394166C546DAF69ED97E055222
5AFF8BF497E21F705635441E06E039331F291B375778699BFB96743D16598B26771E382EFE7134AED8
AA3ECD6BA0A8CB73C41A4134E99819EB96143644E46023DF41A371C6C537173677B9E10CD468604AA6
97E9B6BF30B9CCCF64AE12A705457DF5FD092D3234E5B3F542D971AD3016C997901E58F8CF5A03BA19
DECF3C4D48EA1562DEDFA622DCD1D686B26291E1F8EE614110400F114767BC04F341444F43F1A6BA93
4753A33E43BDCCF6B96CFC41B6FCEDADF4EC911912EFB2D37905784B15CA3234D3B208D5D9CC351F00
196EAF6304CB288047CFFAF5DDA26B087379C60A590C3FFEAED6A0D2D5CD2BFDED5D68A53C89B3D8DF
AE670D97ED6199151562B8AECF3D96FF80E5E6318C275B105DE8888F8848977346330FCDDCD2DEBCF6
33FC97D3CEF22AFD3A62297902629C466F48C55BFF5B1714F7FE9580FF2D5ED0C9B72307EAB4B68C95
E96FEAC99DE1B0C84A9B529D21B4FC97C2C51701A030F72A4DD8B94D374C94C5BBD355EB306DA88796
360EC1DE7E63078D56C3EFC41A7AE628F272FAE9C2773039C500E5B941F87A9818426A9CE6C607687F
A460B0193FBDE92C215D4C0BB5C8FD546F64FDB6003BAFB23B1ADE68089CDBD4DCD0E9B964DBE74377
510286CFFEFE91BA067AEAB78D1833F406AAD1DFD039A3ADD86EEC8606A48A851105F3B8E14102FD93
E0237363AFD6A06B7642DB63D12FA48256A52DBD9E05BD94FBE9388905B39F9165A1DC54BD60899E39
370FB4D792E526B1E53DBA0EFEB7DBF2CCB00EC1B2CFDBF55F74F77C59A1B1F8EA225C0D8215EC8472
31E327F468B17EEFF791C9546EF85C77F11E55188D0A7E1784C8E12C1E0EFE168BC1BA3DF7F7E76103
EC2D1300C77FB18264D8E242940EBDBFF801ADB74EE847436F6B590C0BC033E52DAF20A827967B7F1D
9D071FDF5ECA67F7B6259850BDF87F45FEB8272951766CDB4EF92264F0E21D7C7722C66CCE86D4F1A4
C5B81F52CCD22D34248F0A2FCFAE8484DF00543E3204AACF1E433B13E33D58F453707B6BD149317BA7
06D5EDC5C354F7435B48029C492788A4BEDBBDD70BAF1B40854A304CC91C551C4048B8535C3DE2849F
BF82A318DC54E6F8663AFB4E371C29D63614FFF85871179A5DBD3982034E6F9BE60287D9638ED0BCCC
6FB99BE3952CCAA506352C83FF2365E71A5568C595BD72631E1A343EB6301FAF372C2BC10A4110116C
C0009E560785F2253BA46FFA1F57A9CD2CA1097BEB1FF2D00DC8A3AF74075B7E67396E957D3B92F8AB
D31E83F8A5D97161C8DE4114C94F8D00714EA193A51384F9E8382C4C219420C268A5ADB0E9ECCEBF36
2AF8D2BA671562BB45315071F9BA89FD24493535A8BD48F5A6C0B5808926BD9AA49E879E850E340A15
4C88F27390E9BC6681AC9B0C78F3AEB9C798ED15616D66AFCA05E7D2F7F2B30645BEA6C7131FEC34FD
F6C2A19499950FD1FDE8E4ECAC3BFF8959D1E270A47E26495DB0EC65EE8F250B8FD5D69FEDDD7DB4D7
64B1AF15D95F60D41E9FBBEC13E055A6C8E31A3CE170B24CEF0FA0D56AA5BCA2C6A681DC2522C12208
D19B54328765E5298F6D305608896D6AE090FC4AE8DD0B3C9A0951B4252BD41A0E7C0FE4D6F7223A27
C11F96EBCB5DBF361B2F136EF5C58203C5D4D303E6839DF6CC992B59DF0FFF51AC33599451ECBBB64A
790EA3CB7A31FE6A91B249A37CAA0C34CBCC74968B27DA590AFA22E0C13473353B3D87843347592EE9
126DE8F8629AC49AB4E0E3CE4B054729B3C8270042D1A2EA733D8C3E0B80584C5F0D999F0620D5CB53

```
        F2CA9BA50C37872E6EA1A2C099F150847B3A35ADC8B6968A7B2C442696413B632FA7BDC37A68CDB480
        0000
SK2Key1 = SK2[0..7]: DE 4F 90 FE 96 FA 22 CA
SK2Key1 = SK2[8..15]: 92 56 89 80 B9 71 F4 E1
MAC = SK2Key1.CBC_encrypt(MACData, IV(SSC)): 03EA3EBD02A46470
MAC = SK2Key2.decrypt(MAC): A566A9838835DC03
MAC = SK2Key1.encrypt(MAC): 591B43B55556BF0F
// Wrap MAC into TLV with tag 8E.
    MAC = Tag(8E) || Length || Value(MAC):
        878206090181BCCBB003C8C401CA4976AC94AE7032ACAB10E9127D85AF01D02C51B94C71EAAF263DD3
        54DDAFAE4CD362FB0DA09B69F0C6C399361AD53FC443FD2CA3C6BEE4DA6B4D1E14195503AA68869182
        6B5AFD59D18794EF8027F38B7D3F86A7BCE6182036EBA2BF5423DBC73660702619CD29CB3F7AFA7ACE
        6963F3DB7958DB6BA1E6076A1B56FE205378C33D7F8E284B5315CFDF3F247EB19FB72B5C8189B7500C
        CB1D31B0AC7DFB236CE33FCAF53038F15F426189225B5FDB876C40C4726935ED4E91B541C7835D8D16
        B8B5CB342DF4D255BF7C77251B4FE93838267ABB394166C546DAF69ED97E0552225AFF8BF497E21F70
        5635441E06E039331F291B375778699BFB96743D16598B26771E382EFE7134AED8AA3ECD6BA0A8CB73
        C41A4134E99819EB96143644E46023DF41A371C6C537173677B9E10CD468604AA697E9B6BF30B9CCCF
        64AE12A705457DF5FD092D3234E5B3F542D971AD3016C997901E58F8CF5A03BA19DECF3C4D48EA1562
        DEDFA622DCD1D686B26291E1F8EE614110400F114767BC04F341444F43F1A6BA934753A33E43BDCCF6
        B96CFC41B6FCEDADF4EC911912EFB2D37905784B15CA3234D3B208D5D9CC351F00196EAF6304CB2880
        47CFFAF5DDA26B087379C60A590C3FFEAED6A0D2D5CD2BFDED5D68A53C89B3D8DFAE670D97ED619915
        1562B8AECF3D96FF80E5E6318C275B105DE8888F8848977346330FCDDCD2DEBCF633FC97D3CEF22AFD
        3A62297902629C466F48C55BFF5B1714F7FE9580FF2D5ED0C9B72307EAB4B68C95E96FEAC99DE1B0C8
        4A9B529D21B4FC97C2C51701A030F72A4DD8B94D374C94C5BBD355EB306DA88796360EC1DE7E63078D
        56C3EFC41A7AE628F272FAE9C2773039C500E5B941F87A9818426A9CE6C607687FA460B0193FBDE92C
        215D4C0BB5C8FD546F64FDB6003BAFB23B1ADE68089CDBD4DCD0E9B964DBE74377510286CFFEFE91BA
        067AEAB78D1833F406AAD1DFD039A3ADD86EEC8606A48A851105F3B8E14102FD93E0237363AFD6A06B
        7642DB63D12FA48256A52DBD9E05BD94FBE9388905B39F9165A1DC54BD60899E39370FB4D792E526B1
        E53DBA0EFEB7DBF2CCB00EC1B2CFDBF55F74F77C59A1B1F8EA225C0D8215EC847231E327F468B17EEF
        F791C9546EF85C77F11E55188D0A7E1784C8E12C1E0EFE168BC1BA3DF7F7E76103EC2D1300C77FB182
        64D8E242940EBDBFF801ADB74EE847436F6B590C0BC033E52DAF20A827967B7F1D9D071FDF5ECA67F7
        B6259850BDF87F45FEB8272951766CDB4EF92264F0E21D7C7722C66CCE86D4F1A4C5B81F52CCD22D34
        248F0A2FCFAE8484DF00543E3204AACF1E433B13E33D58F453707B6BD149317BA706D5EDC5C354F743
        5B48029C492788A4BEDBBDD70BAF1B40854A304CC91C551C4048B8535C3DE2849FBF82A318DC54E6F8
        663AFB4E371C29D63614FFF85871179A5DBD3982034E6F9BE60287D9638ED0BCCC6FB99BE3952CCAA5
        06352C83FF2365E71A5568C595BD72631E1A343EB6301FAF372C2BC10A4110116CC0009E560785F225
        3BA46FFA1F57A9CD2CA1097BEB1FF2D00DC8A3AF74075B7E67396E957D3B92F8ABD31E83F8A5D97161
        C8DE4114C94F8D00714EA193A51384F9E8382C4C219420C268A5ADB0E9ECCEBF362AF8D2BA671562BB
        45315071F9BA89FD24493535A8BD48F5A6C0B5808926BD9AA49E879E850E340A154C88F27390E9BC66
        81AC9B0C78F3AEB9C798ED15616D66AFCA05E7D2F7F2B30645BEA6C7131FEC34FDF6C2A19499950FD1
        FDE8E4ECAC3BFF8959D1E270A47E26495DB0EC65EE8F250B8FD5D69FEDDD7DB4D764B1AF15D95F60D4
        1E9FBBEC13E055A6C8E31A3CE170B24CEF0FA0D56AA5BCA2C6A681DC2522C12208D19B54328765E529
        8F6D305608896D6AE090FC4AE8DD0B3C9A0951B4252BD41A0E7C0FE4D6F7223A27C11F96EBCB5DBF36
        1B2F136EF5C58203C5D4D303E6839DF6CC992B59DF0FFF51AC33599451ECBBB64A790EA3CB7A31FE6A
        91B249A37CAA0C34CBCC74968B27DA590AFA22E0C13473353B3D87843347592EE9126DE8F8629AC49A
        B4E0E3CE4B054729B3C8270042D1A2EA733D8C3E0B80584C5F0D999F0620D5CB53F2CA9BA50C37872E
        6EA1A2C099F150847B3A35ADC8B6968A7B2C442696413B632FA7BDC37A68CDB48E08591B43B55556BF
        0F
Data = Data || MAC:
    878206090181BCCBB003C8C401CA4976AC94AE7032ACAB10E9127D85AF01D02C51B94C71EAAF263DD354DD
    AFAE4CD362FB0DA09B69F0C6C399361AD53FC443FD2CA3C6BEE4DA6B4D1E14195503AA688691826B5AFD59
    D18794EF8027F38B7D3F86A7BCE6182036EBA2BF5423DBC73660702619CD29CB3F7AFA7ACE6963F3DB7958
    DB6BA1E6076A1B56FE205378C33D7F8E284B5315CFDF3F247EB19FB72B5C8189B7500CCB1D31B0AC7DFB23
    6CE33FCAF53038F15F426189225B5FDB876C40C4726935ED4E91B541C7835D8D16B8B5CB342DF4D255BF7C
    77251B4FE93838267ABB394166C546DAF69ED97E0552225AFF8BF497E21F705635441E06E039331F291B37
    5778699BFB96743D16598B26771E382EFE7134AED8AA3ECD6BA0A8CB73C41A4134E99819EB96143644E460
    23DF41A371C6C537173677B9E10CD468604AA697E9B6BF30B9CCCF64AE12A705457DF5FD092D3234E5B3F5
    42D971AD3016C997901E58F8CF5A03BA19DECF3C4D48EA1562DEDFA622DCD1D686B26291E1F8EE61411040
```

```
0F114767BC04F341444F43F1A6BA934753A33E43BDCCF6B96CFC41B6FCEDADF4EC911912EFB2D37905784B
15CA3234D3B208D5D9CC351F00196EAF6304CB288047CFFAF5DDA26B087379C60A590C3FFEAED6A0D2D5CD
2BFDED5D68A53C89B3D8DFAE670D97ED6199151562B8AECF3D96FF80E5E6318C275B105DE8888F88489773
46330FCDDCD2DEBCF633FC97D3CEF22AFD3A62297902629C466F48C55BFF5B1714F7FE9580FF2D5ED0C9B7
2307EAB4B68C95E96FEAC99DE1B0C84A9B529D21B4FC97C2C51701A030F72A4DD8B94D374C94C5BBD355EB
306DA88796360EC1DE7E63078D56C3EFC41A7AE628F272FAE9C2773039C500E5B941F87A9818426A9CE6C6
07687FA460B0193FBDE92C215D4C0BB5C8FD546F64FDB6003BAFB23B1ADE68089CDBD4DCD0E9B964DBE743
77510286CFFEFE91BA067AEAB78D1833F406AAD1DFD039A3ADD86EEC8606A48A851105F3B8E14102FD93E0
237363AFD6A06B7642DB63D12FA48256A52DBD9E05BD94FBE9388905B39F9165A1DC54BD60899E39370FB4
D792E526B1E53DBA0EFEB7DBF2CCB00EC1B2CFDBF55F74F77C59A1B1F8EA225C0D8215EC847231E327F468
B17EEFF791C9546EF85C77F11E55188D0A7E1784C8E12C1E0EFE168BC1BA3DF7F7E76103EC2D1300C77FB1
8264D8E242940EBDBFF801ADB74EE847436F6B590C0BC033E52DAF20A827967B7F1D9D071FDF5ECA67F7B6
259850BDF87F45FEB8272951766CDB4EF92264F0E21D7C7722C66CCE86D4F1A4C5B81F52CCD22D34248F0A
2FCFAE8484DF00543E3204AACF1E433B13E33D58F453707B6BD149317BA706D5EDC5C354F7435B48029C49
2788A4BEDBBDD70BAF1B40854A304CC91C551C4048B8535C3DE2849FBF82A318DC54E6F8663AFB4E371C29
D63614FFF85871179A5DBD3982034E6F9BE60287D9638ED0BCCC6FB99BE3952CCAA506352C83FF2365E71A
5568C595BD72631E1A343EB6301FAF372C2BC10A4110116CC0009E560785F2253BA46FFA1F57A9CD2CA109
7BEB1FF2D00DC8A3AF74075B7E67396E957D3B92F8ABD31E83F8A5D97161C8DE4114C94F8D00714EA193A5
1384F9E8382C4C219420C268A5ADB0E9ECCEBF362AF8D2BA671562BB45315071F9BA89FD24493535A8BD48
F5A6C0B5808926BD9AA49E879E850E340A154C88F27390E9BC6681AC9B0C78F3AEB9C798ED15616D66AFCA
05E7D2F7F2B30645BEA6C7131FEC34FDF6C2A19499950FD1FDE8E4ECAC3BFF8959D1E270A47E26495DB0EC
65EE8F250B8FD5D69FEDDD7DB4D764B1AF15D95F60D41E9FBBEC13E055A6C8E31A3CE170B24CEF0FA0D56A
A5BCA2C6A681DC2522C12208D19B54328765E5298F6D305608896D6AE090FC4AE8DD0B3C9A0951B4252BD4
1A0E7C0FE4D6F7223A27C11F96EBCB5DBF361B2F136EF5C58203C5D4D303E6839DF6CC992B59DF0FFF51AC
33599451ECBBB64A790EA3CB7A31FE6A91B249A37CAA0C34CBCC74968B27DA590AFA22E0C13473353B3D87
843347592EE9126DE8F8629AC49AB4E0E3CE4B054729B3C8270042D1A2EA733D8C3E0B80584C5F0D999F06
20D5CB53F2CA9BA50C37872E6EA1A2C099F150847B3A35ADC8B6968A7B2C442696413B632FA7BDC37A68CD
B48E08591B43B55556BF0F
```

```
>>  0C 07 01 00 00 06 17 87 82 06 09 01 81 BC CB B0 03 C8 C4 01 CA 49 76 AC 94 AE 70 32 AC
    AB 10 E9 12 7D 85 AF 01 D0 2C 51 B9 4C 71 EA AF 26 3D D3 54 DD AF AE 4C D3 62 FB 0D A0
    9B 69 F0 C6 C3 99 36 1A D5 3F C4 43 FD 2C A3 C6 BE E4 DA 6B 4D 1E 14 19 55 03 AA 68 86
    91 82 6B 5A FD 59 D1 87 94 EF 80 27 F3 8B 7D 3F 86 A7 BC E6 18 20 36 EB A2 BF 54 23 DB
    C7 36 60 70 26 19 CD 29 CB 3F 7A FA 7A CE 69 63 F3 DB 79 58 DB 6B A1 E6 07 6A 1B 56 FE
    20 53 78 C3 3D 7F 8E 28 4B 53 15 CF DF 3F 24 7E B1 9F B7 2B 5C 81 89 B7 50 0C CB 1D 31
    B0 AC 7D FB 23 6C E3 3F CA F5 30 38 F1 5F 42 61 89 22 5B 5F DB 87 6C 40 C4 72 69 35 ED
    4E 91 B5 41 C7 83 5D 8D 16 B8 B5 CB 34 2D F4 D2 55 BF 7C 77 25 1B 4F E9 38 38 26 7A BB
    39 41 66 C5 46 DA F6 9E D9 7E 05 52 22 5A FF 8B F4 97 E2 1F 70 56 35 44 1E 06 E0 39 33
    1F 29 1B 37 57 78 69 9B FB 96 74 3D 16 59 8B 26 77 1E 38 2E FE 71 34 AE D8 AA 3E CD 6B
    A0 A8 CB 73 C4 1A 41 34 E9 98 19 EB 96 14 36 44 E4 60 23 DF 41 A3 71 C6 C5 37 17 36 77
    B9 E1 0C D4 68 60 4A A6 97 E9 B6 BF 30 B9 CC CF 64 AE 12 A7 05 45 7D F5 FD 09 2D 32 34
    E5 B3 F5 42 D9 71 AD 30 16 C9 97 90 1E 58 F8 CF 5A 03 BA 19 DE CF 3C 4D 48 EA 15 62 DE
    DF A6 22 DC D1 D6 86 B2 62 91 E1 F8 EE 61 41 10 40 0F 11 47 67 BC 04 F3 41 44 4F 43 F1
    A6 BA 93 47 53 A3 3E 43 BD CC F6 B9 6C FC 41 B6 FC ED AD F4 EC 91 19 12 EF B2 D3 79 05
    78 4B 15 CA 32 34 D3 B2 08 D5 D9 CC 35 1F 00 19 6E AF 63 04 CB 28 80 47 CF FA F5 DD A2
    6B 08 73 79 C6 0A 59 0C 3F FE AE D6 A0 D2 D5 CD 2B FD ED 5D 68 A5 3C 89 B3 D8 DF AE 67
    0D 97 ED 61 99 15 15 62 B8 AE CF 3D 96 FF 80 E5 E6 31 8C 27 5B 10 5D E8 88 8F 88 48 97
    73 46 33 0F CD DC D2 DE BC F6 33 FC 97 D3 CE F2 2A FD 3A 62 29 79 02 62 9C 46 6F 48 C5
    5B FF 5B 17 14 F7 FE 95 80 FF 2D 5E D0 C9 B7 23 07 EA B4 B6 8C 95 E9 6F EA C9 9D E1 B0
    C8 4A 9B 52 9D 21 B4 FC 97 C2 C5 17 01 A0 30 F7 2A 4D D8 B9 4D 37 4C 94 C5 BB D3 55 EB
    30 6D A8 87 96 36 0E C1 DE 7E 63 07 8D 56 C3 EF C4 1A 7A E6 28 F2 72 FA E9 C2 77 30 39
    C5 00 E5 B9 41 F8 7A 98 18 42 6A 9C E6 C6 07 68 7F A4 60 B0 19 3F BD E9 2C 21 5D 4C 0B
    B5 C8 FD 54 6F 64 FD B6 00 3B AF B2 3B 1A DE 68 08 9C DB D4 DC D0 E9 B9 64 DB E7 43 77
    51 02 86 CF FE FE 91 BA 06 7A EA B7 8D 18 33 F4 06 AA D1 DF D0 39 A3 AD D8 6E EC 86 06
    A4 8A 85 11 05 F3 B8 E1 41 02 FD 93 E0 23 73 63 AF D6 A0 6B 76 42 DB 63 D1 2F A4 82 56
    A5 2D BD 9E 05 BD 94 FB E9 38 89 05 B3 9F 91 65 A1 DC 54 BD 60 89 9E 39 37 0F B4 D7 92
    E5 26 B1 E5 3D BA 0E FE B7 DB F2 CC B0 0E C1 B2 CF DB F5 5F 74 F7 7C 59 A1 B1 F8 EA 22
    5C 0D 82 15 EC 84 72 31 E3 27 F4 68 B1 7E EF F7 91 C9 54 6E F8 5C 77 F1 1E 55 18 8D 0A
    7E 17 84 C8 E1 2C 1E 0E FE 16 8B C1 BA 3D F7 F7 E7 61 03 EC 2D 13 00 C7 7F B1 82 64 D8
    E2 42 94 0E BD BF F8 01 AD B7 4E E8 47 43 6F 6B 59 0C 0B C0 33 E5 2D AF 20 A8 27 96 7B
    7F 1D 9D 07 1F DF 5E CA 67 F7 B6 25 98 50 BD F8 7F 45 FE B8 27 29 51 76 6C DB 4E F9 22
    64 F0 E2 1D 7C 77 22 C6 6C CE 86 D4 F1 A4 C5 B8 1F 52 CC D2 2D 34 24 8F 0A 2F CF AE 84
```

```
    84 DF 00 54 3E 32 04 AA CF 1E 43 3B 13 E3 3D 58 F4 53 70 7B 6B D1 49 31 7B A7 06 D5 ED
    C5 C3 54 F7 43 5B 48 02 9C 49 27 88 A4 BE DB BD D7 0B AF 1B 40 85 4A 30 4C C9 1C 55 1C
    40 48 B8 53 5C 3D E2 84 9F BF 82 A3 18 DC 54 E6 F8 66 3A FB 4E 37 1C 29 D6 36 14 FF F8
    58 71 17 9A 5D BD 39 82 03 4E 6F 9B E6 02 87 D9 63 8E D0 BC CC 6F B9 9B E3 95 2C CA A5
    06 35 2C 83 FF 23 65 E7 1A 55 68 C5 95 BD 72 63 1E 1A 34 3E B6 30 1F AF 37 2C 2B C1 0A
    41 10 11 6C C0 00 9E 56 07 85 F2 25 3B A4 6F FA 1F 57 A9 CD 2C A1 09 7B EB 1F F2 D0 0D
    C8 A3 AF 74 07 5B 7E 67 39 6E 95 7D 3B 92 F8 AB D3 1E 83 F8 A5 D9 71 61 C8 DE 41 14 C9
    4F 8D 00 71 4E A1 93 A5 13 84 F9 E8 38 2C 4C 21 94 20 C2 68 A5 AD B0 E9 EC CE BF 36 2A
    F8 D2 BA 67 15 62 BB 45 31 50 71 F9 BA 89 FD 24 49 35 35 A8 BD 48 F5 A6 C0 B5 80 89 26
    BD 9A A4 9E 87 9E 85 0E 34 0A 15 4C 88 F2 73 90 E9 BC 66 81 AC 9B 0C 78 F3 AE B9 C7 98
    ED 15 61 6D 66 AF CA 05 E7 D2 F7 F2 B3 06 45 BE A6 C7 13 1F EC 34 FD F6 C2 A1 94 99 95
    0F D1 FD E8 E4 EC AC 3B FF 89 59 D1 E2 70 A4 7E 26 49 5D B0 EC 65 EE 8F 25 0B 8F D5 D6
    9F ED DD 7D B4 D7 64 B1 AF 15 D9 5F 60 D4 1E 9F BB EC 13 E0 55 A6 C8 E3 1A 3C E1 70 B2
    4C EF 0F A0 D5 6A A5 BC A2 C6 A6 81 DC 25 22 C1 22 08 D1 9B 54 32 87 65 E5 29 8F 6D 30
    56 08 89 6D 6A E0 90 FC 4A E8 DD 0B 3C 9A 09 51 B4 25 2B D4 1A 0E 7C 0F E4 D6 F7 22 3A
    27 C1 1F 96 EB CB 5D BF 36 1B 2F 13 6E F5 C5 82 03 C5 D4 D3 03 E6 83 9D F6 CC 99 2B 59
    DF 0F FF 51 AC 33 59 94 51 EC BB B6 4A 79 0E A3 CB 7A 31 FE 6A 91 B2 49 A3 7C AA 0C 34
    CB CC 74 96 8B 27 DA 59 0A FA 22 E0 C1 34 73 35 3B 3D 87 84 33 47 59 2E E9 12 6D E8 F8
    62 9A C4 9A B4 E0 E3 CE 4B 05 47 29 B3 C8 27 00 42 D1 A2 EA 73 3D 8C 3E 0B 80 58 4C 5F
    0D 99 9F 06 20 D5 CB 53 F2 CA 9B A5 0C 37 87 2E 6E A1 A2 C0 99 F1 50 84 7B 3A 35 AD C8
    B6 96 8A 7B 2C 44 26 96 41 3B 63 2F A7 BD C3 7A 68 CD B4 8E 08 59 1B 43 B5 55 56 BF 0F
    00 00
<<  6E 00
Card error SW1/SW2=6e00 - Checking error: Class not supported
// Secure the command
CLA = CLA | 0C: 0C
SSC(4F 74 3D C4 6C EB FD 58) + 1 = 4F 74 3D C4 6C EB FD 59
Data = Data || ISO9797 method 2 padding:
    308204F4308203DCA003020102021051B708FF70E079E5501F74BD85422479300D06092A864886F70D0101
    050500306C310B3009060355040613024545312230206035504 0A0C19415320536572746966697473656 5
    72696D69736B65736B7573311F301D06035504030C1654455354206F662045535445494 42D534B20323031
    313118301606092A864886F70D0109011609706B6940736B2E6565301E170D3132303830363033393431
    5A170D3137303230313231353935395A30819E310B30090603550406130245453 10F300D060355040A0C06
    455354454944311A3018060355040B0C116469676974616C207369676E617475726531263026040603550403
    0C1D4DC3844E4E494B2C4D4152492D4C4949532C343731303130313030333331103 00E06035504040C074D
    C3844E4E494B311230 10060355042A0C094D4152492D4C4949533114301206035504051 30B343731303130
    313030333330820123300D06092A864886F70D01010105000382011000308201 0B0282010100949B6823BE
    181864F18E45515283D23A31BA69915408F3A771805FA23C361D94FD30D0143D45D3C29C20313B9FCF80F0
    81363633576FE75FE9A034AE49A93FBF4773D8173B5C99D3054B5F8DB1003DFBD450ED091DE75924CEFBA4
    86DC0CA12A8B2F7EDCC5E0997A1EC15C48AC63A891F2F046C1C7E6E0EAEF3CE6151BAD0A9E9F1ADF37AC22
    AF3E5B06D0149A9522A6451005D5A2B0502F7317F8EE082825932B3A385E6C309692A43AFE49D8C4370AD0
    4550804D88F22F13B9C8E6604366E0B7AE918FAC1DA0990D6F6C4AC0FA78CD71769B95210557B6E0A3BE1F
    76EEB4CC51C0E74FC6F6B39DAF0ADAD7652CA2F8BD423BBA38ECE2DE94F58B8CC9D4E68102042CF7B4B7A3
    82015C3082015830090603551D1304023000300E0603551D0F0101FF040403020640308199060355 1D2004
    819130818E30818B060A2B06010401CE1F030101307D305806082B06010505070202304C1E4A0041006900
    6E0075006C00740020007400650073007400690060D0069007300650060B0073002E0020004F006E006C0079
    00200066006F00720020007400650073007400690060E0067002E302106082B06010505070201161568747 4
    703A2F2F7777772E736B2E65652F6370732F301D0603551D0E0416041411E29A1642D77364DB4EC7D24009
    90E00818B1A5301806082B06010505070103040C300A3008060604008E460101301F0603551D2304183016
    801441B6FEC5B1B1B453138CFAFA62D0346D6D22340A30450603551D1F043E303C303AA038A03686346874
    74703A2F2F7777772E736B2E65652F7265706F7369746F72792F63726C732F746573745F65737465696432
    3031312E63726C300D06092A864886F70D01010505000382010100C4EB5D0898A775B66084EB257865F151
    7E709247B18117F49B6A60666E66D5B3282C025018361490AF0A3536A6B52D9AB9ECE36DE8114220301B33
    2F541319452DFE15E475DAB392A0C0745FEFC5168FA8B90639F6C631DB8E3858F4BBED66E71A7B309B837E
    7D649E0D46C90B1655B315A30FA301D6F98EB25CC7B1E85BF7010E1B8016B0458688EA66C28F8B04F5CCD7
    27458C9CBC36A937C276273FDE0D03E260927992002777E9433E5A8A2C158296FCBA2F77568BD8CE65DEEE
    0955145EEC69AE83B1256407E926D0D36451BBEC48771FEC5F23FC16DF998A70F6DE0E822B5A89F3BF2E35
    96464AE6B86EC62F6E2C6B86A61836AA19863E37E2B0516249800000000000000000000000000000000000
    000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
     00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
     00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
     00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
     00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
     000000000000000000000000000000000000000000000000000000000000008000000000000000
Cryptogram = SK1.encrypt(data, IV(SSC)) =
     9EFFA8536486573A8F48FECB8E9E20A1599578AB59ED04725A42FBEACD60E64DB1E645A6E495CFC84A7914
     5D94123B8A49DFB395E502AFBD65780F8E3DA459E7A32375DC75987AF5C39A8BE4B0BB9020E2B6396C16B5
     E9DC48ADDD0A107C9A2047D06D3399B30CAD17A129C7609AF7EAFC0DA41EEF81E113DE977C2E905D578B1D
     A8DE04D7D706E7481F0948CF15DF688EE4039A78BFF20065FCAAA88E969897AEF86CCE00D35AF4EFB8F0D1
     65C591FC3EE3F9408FD69869EA41E74DC77C6EC4C1513DDA92CCC01195131F0AE96C06D304744EF90AD7A6
     8BE73CB08DECDEBC13D4CD022C6FB15D211AAFCA414D4A00840261B03B121110C940046A0C82FF4AF8EEBE
     D3D367BFB8062FED3ED7010F54BE937092F5E25B6AADC51DC1567B3CDEC6E24BFA7E4D768E24E4DFEF3961
     B9A89F5B83A3388EFF15439BBF1221DF22DFB3EDB5923B6773E9B9D5F2C398B63BA242FF74FB47E37535F0
     21501DD662687B2278227E6417B86F40E628A9FE463F56659929160ED89495B8984F8AF7C436CE4B830D68
     8BF3CC4D99BA310AF9A9E6C03CF59338592F9A19509AAEED3F58CBC14394FD0AA8557ED5FF1CA17B8FB3C7
     03DFBEF4397D8E962CF38E27F31919A2644A065D07226E676D31ED1FE95625AB10D82D8A1BCDA8D0E91830
     4410A2BFF61D75EA6B8865E6BD60BAF21AF30E39452C4CBDA4E72E74DCFAD7F8808D13BF190BE1540497D5
     A557397D619706735EB28AF4242009B72CA4C1562903491310538DC74FCACA59CA102466DF01B313C55272
     43ADE312CD2A91759599ABC7B7E0AC3D4CC21FBA38AD42BA93A5C70E91818CC1754541790CFDC0092FD008
     CA45499C8D928AE08878BCD606445848408FC8DF7DFED6D5B1257B335869F319BDD68E99117C4DBA5DC2D6
     EA1778EB97C85162D7453C1B7B7B550C332E3B1EA4A12880DE5DB0B74B3FFB49A81C59146AA49908CD6393
     C1B9250E5E2BFBF1D74B461E7D2E4C21D2CDE939D5AB0FF624C6CB8A263776515F6E3746DF25CD8989CEEF
     0ABF63158F3F742B9D24E74948C209C03BC2587C0C38D3824B2A595D787DD5951448411EC1A0FA0F18959F
     5AD91D89D32F2DE77B1C0411A91DC94BDCC00B908669B6B4872756CF3F1116A03B5F180493F10BCD847CEF
     9BCDA2E7CAFB27DB539720A0D51FBC28CB5521FE2F243E4F43FFB7009E991558948043492ECCEA0C28E9C2
     A9E76E80CB493A0420599AC30524EEA1065B20C03C5AA4ABBD75438DB2E6DAFFB3B078754DBADDB2A87858
     79C14BFA30C032C1B9FBECE8C0A002485474A3CF15CE323DE9FFAA1AC1748737EDB96DFC59F6FACD6004CF
     D8D6925642B00368D0B9E02B6AA4C7FF19773C2B1B36A3C7DF507159E2A1A51A7DF2BD68631B592D81DCCD
     2718F111FDD7271214F3F7FF1F9180069567DCE91C28D317CAE57B1C7843061A5616239B386BA4797D0D1F
     EC411A57A49B455943B6AB0C5DA1511C784C84BE702A0AE9DDCBA6515BBE58102B0A4F367F350D156A138C
     9F1D03B7EFB60129220F79CE95C165B028C75C83CC5C48E48914EC33265C7A1352AF6D4FBAB901463998DB
     9FBB56EF5648C8AFF4A4CBD2F35BA4733EE4C2ECAB8EA60B9D092311DDBF423468564D3115CFC33AE9998E
     ED2FDBD809F8972E6CC25597576919D3B00C41815AB3E8AB5A5BF7D113C093F7D22E787B0B4F221C29FAEF
     9F0670E3D6035DCE27403AAD13CFB2DE95C372EA4EFA26D3BC9DBF5E234BF8E418260887A964E2459EBB1D
     27F27011C62FDF5258E623F270F1413FC89082A50A60075400541FAE022838342EB860BA7D7BEB98574A43
     202FFF3FA8E42058B2F85CF7E0C2EE3CCD22FF172A61C278512FC11834DFFC9793AD8A60FECDB0247F610F
     55482FA0D76B09674D5C4FA24016204A1186FB300D84169676C90288730733BCC8E325FB36955A75145BB6
     57F2B0E37FEDCDC34D8DDF57EECCBDD4EAE919395E132D078FC266879996226104D42A38DA0C1EF99C789F
     92CA1595B9E0B7CCB8D1CFF38FF2251F6BCF7DD92A9F90C1B36CC3BA972E4770981279BB7D6723419AC9C8
     597BA3208FE7307D93C6C879737425999FB5804022FC2E66A7073A52EFFD5E8522B58DAE5DE408AD717A57
     5CE22FA2591D13FF4740E2E2F792496D4B6012B636AEED4A3ED2432B422CF46CAED09C7CBB9E69
// Prepare MACData
     Append header: 0C070200
     Append 80000000: 0C07020080000000
     // Wrap Cryptogram into TLV with tag 87.
     Data = Tag(87) || Length || Value(Cryptogram):
          87820609019EFFA8536486573A8F48FECB8E9E20A1599578AB59ED04725A42FBEACD60E64DB1E645A6
          E495CFC84A79145D94123B8A49DFB395E502AFBD65780F8E3DA459E7A32375DC75987AF5C39A8BE4B0
          BB9020E2B6396C16B5E9DC48ADDD0A107C9A2047D06D3399B30CAD17A129C7609AF7EAFC0DA41EEF81
          E113DE977C2E905D578B1DA8DE04D7D706E7481F0948CF15DF688EE4039A78BFF20065FCAAA88E9698
          97AEF86CCE00D35AF4EFB8F0D165C591FC3EE3F9408FD69869EA41E74DC77C6EC4C1513DDA92CCC011
          95131F0AE96C06D304744EF90AD7A68BE73CB08DECDEBC13D4CD022C6FB15D211AAFCA414D4A008402
          61B03B121110C940046A0C82FF4AF8EEBED3D367BFB8062FED3ED7010F54BE937092F5E25B6AADC51D
          C1567B3CDEC6E24BFA7E4D768E24E4DFEF3961B9A89F5B83A3388EFF15439BBF1221DF22DFB3EDB592
          3B6773E9B9D5F2C398B63BA242FF74FB47E37535F021501DD662687B2278227E6417B86F40E628A9FE
          463F56659929160ED89495B8984F8AF7C436CE4B830D688BF3CC4D99BA310AF9A9E6C03CF59338592F
          9A19509AAEED3F58CBC14394FD0AA8557ED5FF1CA17B8FB3C703DFBEF4397D8E962CF38E27F31919A2
          644A065D07226E676D31ED1FE95625AB10D82D8A1BCDA8D0E918304410A2BFF61D75EA6B8865E6BD60
```

BAF21AF30E39452C4CBDA4E72E74DCFAD7F8808D13BF190BE1540497D5A557397D619706735EB28AF4
242009B72CA4C1562903491310538DC74FCACA59CA102466DF01B313C5527243ADE312CD2A91759599
ABC7B7E0AC3D4CC21FBA38AD42BA93A5C70E91818CC1754541790CFDC0092FD008CA45499C8D928AE0
8878BCD606445848408FC8DF7DFED6D5B1257B335869F319BDD68E99117C4DBA5DC2D6EA1778EB97C8
5162D7453C1B7B7B550C332E3B1EA4A12880DE5DB0B74B3FFB49A81C59146AA49908CD6393C1B9250E
5E2BFBF1D74B461E7D2E4C21D2CDE939D5AB0FF624C6CB8A263776515F6E3746DF25CD8989CEEF0ABF
63158F3F742B9D24E74948C209C03BC2587C0C38D3824B2A595D787DD5951448411EC1A0FA0F18959F
5AD91D89D32F2DE77B1C0411A91DC94BDCC00B908669B6B4872756CF3F1116A03B5F180493F10BCD84
7CEF9BCDA2E7CAFB27DB539720A0D51FBC28CB5521FE2F243E4F43FFB7009E991558948043492ECCEA
0C28E9C2A9E76E80CB493A0420599AC30524EEA1065B20C03C5AA4ABBD75438DB2E6DAFFB3B078754D
BADDB2A8785879C14BFA30C032C1B9FBECE8C0A002485474A3CF15CE323DE9FFAA1AC1748737EDB96D
FC59F6FACD6004CFD8D6925642B00368D0B9E02B6AA4C7FF19773C2B1B36A3C7DF507159E2A1A51A7D
F2BD68631B592D81DCCD2718F111FDD7271214F3F7FF1F9180069567DCE91C28D317CAE57B1C784306
1A5616239B386BA4797D0D1FEC411A57A49B455943B6AB0C5DA1511C784C84BE702A0AE9DDCBA6515B
BE58102B0A4F367F350D156A138C9F1D03B7EFB60129220F79CE95C165B028C75C83CC5C48E48914EC
33265C7A1352AF6D4FBAB901463998DB9FBB56EF5648C8AFF4A4CBD2F35BA4733EE4C2ECAB8EA60B9D
092311DDBF423468564D3115CFC33AE9998EED2FDBD809F8972E6CC25597576919D3B00C41815AB3E8
AB5A5BF7D113C093F7D22E787B0B4F221C29FAEF9F0670E3D6035DCE27403AAD13CFB2DE95C372EA4E
FA26D3BC9DBF5E234BF8E418260887A964E2459EBB1D27F27011C62FDF5258E623F270F1413FC89082
A50A60075400541FAE022838342EB860BA7D7BEB98574A43202FFF3FA8E42058B2F85CF7E0C2EE3CCD
22FF172A61C278512FC11834DFFC9793AD8A60FECDB0247F610F55482FA0D76B09674D5C4FA2401620
4A1186FB300D84169676C90288730733BCC8E325FB36955A75145BB657F2B0E37FEDCDC34D8DDF57EE
CCBDD4EAE919395E132D078FC266879996226104D42A38DA0C1EF99C789F92CA1595B9E0B7CCB8D1CF
F38FF2251F6BCF7DD92A9F90C1B36CC3BA972E4770981279BB7D6723419AC9C8597BA3208FE7307D93
C6C879737425999FB5804022FC2E66A7073A52EFFD5E8522B58DAE5DE408AD717A575CE22FA2591D13
FF4740E2E2F792496D4B6012B636AEED4A3ED2432B422CF46CAED09C7CBB9E69

MACData = Append ISO9797 method 2 padding:

0C07020080000000087820609019EFFA8536486573A8F48FECB8E9E20A1599578AB59ED04725A42FBEA
CD60E64DB1E645A6E495CFC84A79145D94123B8A49DFB395E502AFBD65780F8E3DA459E7A32375DC75
987AF5C39A8BE4B0BB9020E2B6396C16B5E9DC48ADDD0A107C9A2047D06D3399B30CAD17A129C7609A
F7EAFC0DA41EEF81E113DE977C2E905D578B1DA8DE04D7D706E7481F0948CF15DF688EE4039A78BFF2
0065FCAAA88E969897AEF86CCE00D35AF4EFB8F0D165C591FC3EE3F9408FD69869EA41E74DC77C6EC4
C1513DDA92CCC01195131F0AE96C06D304744EF90AD7A68BE73CB08DECDEBC13D4CD022C6FB15D211A
AFCA414D4A00840261B03B121110C940046A0C82FF4AF8EEBED3D367BFB8062FED3ED7010F54BE9370
92F5E25B6AADC51DC1567B3CDEC6E24BFA7E4D768E24E4DFEF3961B9A89F5B83A3388EFF15439BBF12
21DF22DFB3EDB5923B6773E9B9D5F2C398B63BA242FF74FB47E37535F021501DD662687B2278227E64
17B86F40E628A9FE463F56659929160ED89495B8984F8AF7C436CE4B830D688BF3CC4D99BA310AF9A9
E6C03CF59338592F9A19509AAEED3F58CBC14394FD0AA8557ED5FF1CA17B8FB3C703DFBEF4397D8E96
2CF38E27F31919A2644A065D07226E676D31ED1FE95625AB10D82D8A1BCDA8D0E918304410A2BFF61D
75EA6B8865E6BD60BAF21AF30E39452C4CBDA4E72E74DCFAD7F8808D13BF190BE1540497D5A557397D
619706735EB28AF4242009B72CA4C1562903491310538DC74FCACA59CA102466DF01B313C5527243AD
E312CD2A91759599ABC7B7E0AC3D4CC21FBA38AD42BA93A5C70E91818CC1754541790CFDC0092FD008
CA45499C8D928AE08878BCD606445848408FC8DF7DFED6D5B1257B335869F319BDD68E99117C4DBA5D
C2D6EA1778EB97C85162D7453C1B7B7B550C332E3B1EA4A12880DE5DB0B74B3FFB49A81C59146AA499
08CD6393C1B9250E5E2BFBF1D74B461E7D2E4C21D2CDE939D5AB0FF624C6CB8A263776515F6E3746DF
25CD8989CEEF0ABF63158F3F742B9D24E74948C209C03BC2587C0C38D3824B2A595D787DD595144841
1EC1A0FA0F18959F5AD91D89D32F2DE77B1C0411A91DC94BDCC00B908669B6B4872756CF3F1116A03B
5F180493F10BCD847CEF9BCDA2E7CAFB27DB539720A0D51FBC28CB5521FE2F243E4F43FFB7009E9915
58948043492ECCEA0C28E9C2A9E76E80CB493A0420599AC30524EEA1065B20C03C5AA4ABBD75438DB2
E6DAFFB3B078754DBADDB2A8785879C14BFA30C032C1B9FBECE8C0A002485474A3CF15CE323DE9FFAA
1AC1748737EDB96DFC59F6FACD6004CFD8D6925642B00368D0B9E02B6AA4C7FF19773C2B1B36A3C7DF
507159E2A1A51A7DF2BD68631B592D81DCCD2718F111FDD7271214F3F7FF1F9180069567DCE91C28D3
17CAE57B1C7843061A5616239B386BA4797D0D1FEC411A57A49B455943B6AB0C5DA1511C784C84BE70
2A0AE9DDCBA6515BBE58102B0A4F367F350D156A138C9F1D03B7EFB60129220F79CE95C165B028C75C
83CC5C48E48914EC33265C7A1352AF6D4FBAB901463998DB9FBB56EF5648C8AFF4A4CBD2F35BA4733E
E4C2ECAB8EA60B9D092311DDBF423468564D3115CFC33AE9998EED2FDBD809F8972E6CC25597576919
D3B00C41815AB3E8AB5A5BF7D113C093F7D22E787B0B4F221C29FAEF9F0670E3D6035DCE27403AAD13
CFB2DE95C372EA4EFA26D3BC9DBF5E234BF8E418260887A964E2459EBB1D27F27011C62FDF5258E623
F270F1413FC89082A50A60075400541FAE022838342EB860BA7D7BEB98574A43202FFF3FA8E42058B2
F85CF7E0C2EE3CCD22FF172A61C278512FC11834DFFC9793AD8A60FECDB0247F610F55482FA0D76B09
674D5C4FA24016204A1186FB300D84169676C90288730733BCC8E325FB36955A75145BB657F2B0E37F

EDCDC34D8DDF57EECCBDD4EAE919395E132D078FC266879996226104D42A38DA0C1EF99C789F92CA15
95B9E0B7CCB8D1CFF38FF2251F6BCF7DD92A9F90C1B36CC3BA972E4770981279BB7D6723419AC9C859
7BA3208FE7307D93C6C879737425999FB5804022FC2E66A7073A52EFFD5E8522B58DAE5DE408AD717A
575CE22FA2591D13FF4740E2E2F792496D4B6012B636AEED4A3ED2432B422CF46CAED09C7CBB9E6980
0000

SK2Key1 = SK2[0..7]: DE 4F 90 FE 96 FA 22 CA

SK2Key1 = SK2[8..15]: 92 56 89 80 B9 71 F4 E1

MAC = SK2Key1.CBC_encrypt(MACData, IV(SSC)): F2 75 C6 7C 5B 75 484E

MAC = SK2Key2.decrypt(MAC): 3A0A30975C11C5BF

MAC = SK2Key1.encrypt(MAC): 50E263557CCBAE5D

Wrap MAC into TLV with tag 8E.

MAX = Tag(8E) || Length || Value(MAC):

87820609019EFFA8536486573A8F48FECB8E9E20A1599578AB59ED04725A42FBEACD60E64DB1E645A6E495
CFC84A79145D94123B8A49DFB395E502AFBD65780F8E3DA459E7A32375DC75987AF5C39A8BE4B0BB9020E2
B6396C16B5E9DC48ADDD0A107C9A2047D06D3399B30CAD17A129C7609AF7EAFC0DA41EEF81E113DE977C2E
905D578B1DA8DE04D7D706E7481F0948CF15DF688EE4039A78BFF20065FCAAA88E969897AEF86CCE00D35A
F4EFB8F0D165C591FC3EE3F9408FD69869EA41E74DC77C6EC4C1513DDA92CCC01195131F0AE96C06D30474
4EF90AD7A68BE73CB08DECDEBC13D4CD022C6FB15D211AAFCA414D4A00840261B03B121110C940046A0C82
FF4AF8EEBED3D367BFB8062FED3ED7010F54BE937092F5E25B6AADC51DC1567B3CDEC6E24BFA7E4D768E24
E4DFEF3961B9A89F5B83A3388EFF15439BBF1221DF22DFB3EDB5923B6773E9B9D5F2C398B63BA242FF74FB
47E37535F021501DD662687B2278227E6417B86F40E628A9FE463F56659929160ED89495B8984F8AF7C436
CE4B830D688BF3CC4D99BA310AF9A9E6C03CF59338592F9A19509AAEED3F58CBC14394FD0AA8557ED5FF1C
A17B8FB3C703DFBEF4397D8E962CF38E27F31919A2644A065D07226E676D31ED1FE95625AB10D82D8A1BCD
A8D0E918304410A2BFF61D75EA6B8865E6BD60BAF21AF30E39452C4CBDA4E72E74DCFAD7F8808D13BF190B
E1540497D5A557397D619706735EB28AF4242009B72CA4C1562903491310538DC74FCACA59CA102466DF01
B313C5527243ADE312CD2A91759599ABC7B7E0AC3D4CC21FBA38AD42BA93A5C70E91818CC1754541790CFD
C0092FD008CA45499C8D928AE08878BCD606445848408FC8DF7DFED6D5B1257B335869F319BDD68E99117C
4DBA5DC2D6EA1778EB97C85162D7453C1B7B7B550C332E3B1EA4A12880DE5DB0B74B3FFB49A81C59146AA4
9908CD6393C1B9250E5E2BFBF1D74B461E7D2E4C21D2CDE939D5AB0FF624C6CB8A263776515F6E3746DF25
CD8989CEEF0ABF63158F3F742B9D24E74948C209C03BC2587C0C38D3824B2A595D787DD5951448411EC1A0
FA0F18959F5AD91D89D32F2DE77B1C0411A91DC94BDCC00B908669B6B4872756CF3F1116A03B5F180493F1
0BCD847CEF9BCDA2E7CAFB27DB539720A0D51FBC28CB5521FE2F243E4F43FFB7009E991558948043492ECC
EA0C28E9C2A9E76E80CB493A0420599AC30524EEA1065B20C03C5AA4ABBD75438DB2E6DAFFB3B078754DBA
DDB2A8785879C14BFA30C032C1B9FBECE8C0A002485474A3CF15CE323DE9FFAA1AC1748737EDB96DFC59F6
FACD6004CFD8D6925642B00368D0B9E02B6AA4C7FF19773C2B1B36A3C7DF507159E2A1A51A7DF2BD68631B
592D81DCCD2718F111FDD7271214F3F7FF1F9180069567DCE91C28D317CAE57B1C7843061A5616239B386B
A4797D0D1FEC411A57A49B455943B6AB0C5DA1511C784C84BE702A0AE9DDCBA6515BBE58102B0A4F367F35
0D156A138C9F1D03B7EFB60129220F79CE95C165B028C75C83CC5C48E48914EC33265C7A1352AF6D4FBAB9
01463998DB9FBB56EF5648C8AFF4A4CBD2F35BA4733EE4C2ECAB8EA60B9D092311DDBF423468564D3115CF
C33AE9998EED2FDBD809F8972E6CC25597576919D3B00C41815AB3E8AB5A5BF7D113C093F7D22E787B0B4F
221C29FAEF9F0670E3D6035DCE27403AAD13CFB2DE95C372EA4EFA26D3BC9DBF5E234BF8E418260887A964
E2459EBB1D27F27011C62FDF5258E623F270F1413FC89082A50A60075400541FAE022838342EB860BA7D7B
EB98574A43202FFF3FA8E42058B2F85CF7E0C2EE3CCD22FF172A61C278512FC11834DFFC9793AD8A60FECD
B0247F610F55482FA0D76B09674D5C4FA24016204A1186FB300D84169676C90288730733BCC8E325FB3695
5A75145BB657F2B0E37FEDCDC34D8DDF57EECCBDD4EAE919395E132D078FC2668799962261O4D42A38DA0C
1EF99C789F92CA1595B9E0B7CCB8D1CFF38FF2251F6BCF7DD92A9F90C1B36CC3BA972E4770981279BB7D67
23419AC9C8597BA3208FE7307D93C6C879737425999FB5804022FC2E66A7073A52EFFD5E8522B58DAE5DE4
08AD717A575CE22FA2591D13FF4740E2E2F792496D4B6012B636AEED4A3ED2432B422CF46CAED09C7CBB9E
698E0850E263557CCBAE5D

Data = Data || MAC:

87820609019EFFA8536486573A8F48FECB8E9E20A1599578AB59ED04725A42FBEACD60E64DB1E645A6E495
CFC84A79145D94123B8A49DFB395E502AFBD65780F8E3DA459E7A32375DC75987AF5C39A8BE4B0BB9020E2
B6396C16B5E9DC48ADDD0A107C9A2047D06D3399B30CAD17A129C7609AF7EAFC0DA41EEF81E113DE977C2E
905D578B1DA8DE04D7D706E7481F0948CF15DF688EE4039A78BFF20065FCAAA88E969897AEF86CCE00D35A
F4EFB8F0D165C591FC3EE3F9408FD69869EA41E74DC77C6EC4C1513DDA92CCC01195131F0AE96C06D30474
4EF90AD7A68BE73CB08DECDEBC13D4CD022C6FB15D211AAFCA414D4A00840261B03B121110C940046A0C82
FF4AF8EEBED3D367BFB8062FED3ED7010F54BE937092F5E25B6AADC51DC1567B3CDEC6E24BFA7E4D768E24
E4DFEF3961B9A89F5B83A3388EFF15439BBF1221DF22DFB3EDB5923B6773E9B9D5F2C398B63BA242FF74FB

```
47E37535F021501DD662687B2278227E6417B86F40E628A9FE463F56659929160ED89495B8984F8AF7C436
CE4B830D688BF3CC4D99BA310AF9A9E6C03CF59338592F9A19509AAEED3F58CBC14394FD0AA8557ED5FF1C
A17B8FB3C703DFBEF4397D8E962CF38E27F31919A2644A065D07226E676D31ED1FE95625AB10D82D8A1BCD
A8D0E918304410A2BFF61D75EA6B8865E6BD60BAF21AF30E39452C4CBDA4E72E74DCFAD7F8808D13BF190B
E1540497D5A557397D619706735EB28AF4242009B72CA4C1562903491310538DC74FCACA59CA102466DF01
B313C5527243ADE312CD2A91759599ABC7B7E0AC3D4CC21FBA38AD42BA93A5C70E91818CC1754541790CFD
C0092FD008CA45499C8D928AE08878BCD606445848408FC8DF7DFED6D5B1257B335869F319BDD68E99117C
4DBA5DC2D6EA1778EB97C85162D7453C1B7B7B550C332E3B1EA4A12880DE5DB0B74B3FFB49A81C59146AA4
9908CD6393C1B9250E5E2BFBF1D74B461E7D2E4C21D2CDE939D5AB0FF624C6CB8A263776515F6E3746DF25
CD8989CEEF0ABF63158F3F742B9D24E74948C209C03BC2587C0C38D3824B2A595D787DD5951448411EC1A0
FA0F18959F5AD91D89D32F2DE77B1C0411A91DC94BDCC00B908669B6B4872756CF3F1116A03B5F180493F1
0BCD847CEF9BCDA2E7CAFB27DB539720A0D51FBC28CB5521FE2F243E4F43FFB7009E991558948043492ECC
EA0C28E9C2A9E76E80CB493A0420599AC30524EEA1065B20C03C5AA4ABBD75438DB2E6DAFFB3B078754DBA
DDB2A8785879C14BFA30C032C1B9FBECE8C0A002485474A3CF15CE323DE9FFAA1AC1748737EDB96DFC59F6
FACD6004CFD8D6925642B00368D0B9E02B6AA4C7FF19773C2B1B36A3C7DF507159E2A1A51A7DF2BD68631B
592D81DCCD2718F111FDD7271214F3F7FF1F9180069567DCE91C28D317CAE57B1C7843061A5616239B386B
A4797D0D1FEC411A57A49B455943B6AB0C5DA1511C784C84BE702A0AE9DDCBA6515BBE58102B0A4F367F35
0D156A138C9F1D03B7EFB60129220F79CE95C165B028C75C83CC5C48E48914EC33265C7A1352AF6D4FBAB9
01463998DB9FBB56EF5648C8AFF4A4CBD2F35BA4733EE4C2ECAB8EA60B9D092311DDBF423468564D3115CF
C33AE9998EED2FDBD809F8972E6CC25597576919D3B00C41815AB3E8AB5A5BF7D113C093F7D22E787B0B4F
221C29FAEF9F0670E3D6035DCE27403AAD13CFB2DE95C372EA4EFA26D3BC9DBF5E234BF8E418260887A964
E2459EBB1D27F27011C62FDF5258E623F270F1413FC89082A50A60075400541FAE022838342EB860BA7D7B
EB98574A43202FFF3FA8E42058B2F85CF7E0C2EE3CCD22FF172A61C278512FC11834DFFC9793AD8A60FECD
B0247F610F55482FA0D76B09674D5C4FA24016204A1186FB300D84169676C90288730733BCC8E325FB3695
5A75145BB657F2B0E37FEDCDC34D8DDF57EECCBDD4EAE919395E132D078FC2668799962261O4D42A38DA0C
1EF99C789F92CA1595B9E0B7CCB8D1CFF38FF2251F6BCF7DD92A9F90C1B36CC3BA972E4770981279BB7D67
23419AC9C8597BA3208FE7307D93C6C879737425999FB5804022FC2E66A7073A52EFFD5E8522B58DAE5DE4
08AD717A575CE22FA2591D13FF4740E2E2F792496D4B6012B636AEED4A3ED2432B422CF46CAED09C7CBB9E
698E0850E263557CCBAE5D
```

```
>>  0C 07 02 00 00 06 17 87 82 06 09 01 9E FF A8 53 64 86 57 3A 8F 48 FE CB 8E 9E 20 A1 59
    95 78 AB 59 ED 04 72 5A 42 FB EA CD 60 E6 4D B1 E6 45 A6 E4 95 CF C8 4A 79 14 5D 94 12
    3B 8A 49 DF B3 95 E5 02 AF BD 65 78 0F 8E 3D A4 59 E7 A3 23 75 DC 75 98 7A F5 C3 9A 8B
    E4 B0 BB 90 20 E2 B6 39 6C 16 B5 E9 DC 48 AD DD 0A 10 7C 9A 20 47 D0 6D 33 99 B3 0C AD
    17 A1 29 C7 60 9A F7 EA FC 0D A4 1E EF 81 E1 13 DE 97 7C 2E 90 5D 57 8B 1D A8 DE 04 D7
    D7 06 E7 48 1F 09 48 CF 15 DF 68 8E E4 03 9A 78 BF F2 00 65 FC AA A8 8E 96 98 97 AE F8
    6C CE 00 D3 5A F4 EF B8 F0 D1 65 C5 91 FC 3E E3 F9 40 8F D6 98 69 EA 41 E7 4D C7 7C 6E
    C4 C1 51 3D DA 92 CC C0 11 95 13 1F 0A E9 6C 06 D3 04 74 4E F9 0A D7 A6 8B E7 3C B0 8D
    EC DE BC 13 D4 CD 02 2C 6F B1 5D 21 1A AF CA 41 4D 4A 00 84 02 61 B0 3B 12 11 10 C9 40
    04 6A 0C 82 FF 4A F8 EE BE D3 D3 67 BF B8 06 2F ED 3E D7 01 0F 54 BE 93 70 92 F5 E2 5B
    6A AD C5 1D C1 56 7B 3C DE C6 E2 4B FA 7E 4D 76 8E 24 E4 DF EF 39 61 B9 A8 9F 5B 83 A3
    38 8E FF 15 43 9B BF 12 21 DF 22 DF B3 ED B5 92 3B 67 73 E9 B9 D5 F2 C3 98 B6 3B A2 42
    FF 74 FB 47 E3 75 35 F0 21 50 1D D6 62 68 7B 22 78 22 7E 64 17 B8 6F 40 E6 28 A9 FE 46
    3F 56 65 99 29 16 0E D8 94 95 B8 98 4F 8A F7 C4 36 CE 4B 83 0D 68 8B F3 CC 4D 99 BA 31
    0A F9 A9 E6 C0 3C F5 93 38 59 2F 9A 19 50 9A AE ED 3F 58 CB C1 43 94 FD 0A A8 55 7E D5
    FF 1C A1 7B 8F B3 C7 03 DF BE F4 39 7D 8E 96 2C F3 8E 27 F3 19 19 A2 64 4A 06 5D 07 22
    6E 67 6D 31 ED 1F E9 56 25 AB 10 D8 2D 8A 1B CD A8 D0 E9 18 30 44 10 A2 BF F6 1D 75 EA
    6B 88 65 E6 BD 60 BA F2 1A F3 0E 39 45 2C 4C BD A4 E7 2E 74 DC FA D7 F8 80 8D 13 BF 19
    0B E1 54 04 97 D5 A5 57 39 7D 61 97 06 73 5E B2 8A F4 24 20 09 B7 2C A4 C1 56 29 03 49
    13 10 53 8D C7 4F CA CA 59 CA 10 24 66 DF 01 B3 13 C5 52 72 43 AD E3 12 CD 2A 91 75 95
    99 AB C7 B7 E0 AC 3D 4C C2 1F BA 38 AD 42 BA 93 A5 C7 0E 91 81 8C C1 75 45 41 79 0C FD
    C0 09 2F D0 08 CA 45 49 9C 8D 92 8A E0 88 78 BC D6 06 44 58 48 40 8F C8 DF 7D FE D6 D5
    B1 25 7B 33 58 69 F3 19 BD D6 8E 99 11 7C 4D BA 5D C2 D6 EA 17 78 EB 97 C8 51 62 D7 45
    3C 1B 7B 7B 55 0C 33 2E 3B 1E A4 A1 28 80 DE 5D B0 B7 4B 3F FB 49 A8 1C 59 14 6A A4 99
    08 CD 63 93 C1 B9 25 0E 5E 2B FB F1 D7 4B 46 1E 7D 2E 4C 21 D2 CD E9 39 D5 AB 0F F6 24
    C6 CB 8A 26 37 76 51 5F 6E 37 46 DF 25 CD 89 89 CE EF 0A BF 63 15 8F 3F 74 2B 9D 24 E7
    49 48 C2 09 C0 3B C2 58 7C 0C 38 D3 82 4B 2A 59 5D 78 7D D5 95 14 48 41 1E C1 A0 FA 0F
    18 95 9F 5A D9 1D 89 D3 2F 2D E7 7B 1C 04 11 A9 1D C9 4B DC C0 0B 90 86 69 B6 B4 87 27
    56 CF 3F 11 16 A0 3B 5F 18 04 93 F1 0B CD 84 7C EF 9B CD A2 E7 CA FB 27 DB 53 97 20 A0
    D5 1F BC 28 CB 55 21 FE 2F 24 3E 4F 43 FF B7 00 9E 99 15 58 94 80 43 49 2E CC EA 0C 28
    E9 C2 A9 E7 6E 80 CB 49 3A 04 20 59 9A C3 05 24 EE A1 06 5B 20 C0 3C 5A A4 AB BD 75 43
    8D B2 E6 DA FF B3 B0 78 75 4D BA DD B2 A8 78 58 79 C1 4B FA 30 C0 32 C1 B9 FB EC E8 C0
```

```
    A0 02 48 54 74 A3 CF 15 CE 32 3D E9 FF AA 1A C1 74 87 37 ED B9 6D FC 59 F6 FA CD 60 04
    CF D8 D6 92 56 42 B0 03 68 D0 B9 E0 2B 6A A4 C7 FF 19 77 3C 2B 1B 36 A3 C7 DF 50 71 59
    E2 A1 A5 1A 7D F2 BD 68 63 1B 59 2D 81 DC CD 27 18 F1 11 FD D7 27 12 14 F3 F7 FF 1F 91
    80 06 95 67 DC E9 1C 28 D3 17 CA E5 7B 1C 78 43 06 1A 56 16 23 9B 38 6B A4 79 7D 0D 1F
    EC 41 1A 57 A4 9B 45 59 43 B6 AB 0C 5D A1 51 1C 78 4C 84 BE 70 2A 0A E9 DD CB A6 51 5B
    BE 58 10 2B 0A 4F 36 7F 35 0D 15 6A 13 8C 9F 1D 03 B7 EF B6 01 29 22 0F 79 CE 95 C1 65
    B0 28 C7 5C 83 CC 5C 48 E4 89 14 EC 33 26 5C 7A 13 52 AF 6D 4F BA B9 01 46 39 98 DB 9F
    BB 56 EF 56 48 C8 AF F4 A4 CB D2 F3 5B A4 73 3E E4 C2 EC AB 8E A6 0B 9D 09 23 11 DD BF
    42 34 68 56 4D 31 15 CF C3 3A E9 99 8E ED 2F DB D8 09 F8 97 2E 6C C2 55 97 57 69 19 D3
    B0 0C 41 81 5A B3 E8 AB 5A 5B F7 D1 13 C0 93 F7 D2 2E 78 7B 0B 4F 22 1C 29 FA EF 9F 06
    70 E3 D6 03 5D CE 27 40 3A AD 13 CF B2 DE 95 C3 72 EA 4E FA 26 D3 BC 9D BF 5E 23 4B F8
    E4 18 26 08 87 A9 64 E2 45 9E BB 1D 27 F2 70 11 C6 2F DF 52 58 E6 23 F2 70 F1 41 3F C8
    90 82 A5 0A 60 07 54 00 54 1F AE 02 28 38 34 2E B8 60 BA 7D 7B EB 98 57 4A 43 20 2F FF
    3F A8 E4 20 58 B2 F8 5C F7 E0 C2 EE 3C CD 22 FF 17 2A 61 C2 78 51 2F C1 18 34 DF FC 97
    93 AD 8A 60 FE CD B0 24 7F 61 0F 55 48 2F A0 D7 6B 09 67 4D 5C 4F A2 40 16 20 4A 11 86
    FB 30 0D 84 16 96 76 C9 02 88 73 07 33 BC C8 E3 25 FB 36 95 5A 75 14 5B B6 57 F2 B0 E3
    7F ED CD C3 4D 8D DF 57 EE CC BD D4 EA E9 19 39 5E 13 2D 07 8F C2 66 87 99 96 22 61 04
    D4 2A 38 DA 0C 1E F9 9C 78 9F 92 CA 15 95 B9 E0 B7 CC B8 D1 CF F3 8F F2 25 1F 6B CF 7D
    D9 2A 9F 90 C1 B3 6C C3 BA 97 2E 47 70 98 12 79 BB 7D 67 23 41 9A C9 C8 59 7B A3 20 8F
    E7 30 7D 93 C6 C8 79 73 74 25 99 9F B5 80 40 22 FC 2E 66 A7 07 3A 52 EF FD 5E 85 22 B5
    8D AE 5D E4 08 AD 71 7A 57 5C E2 2F A2 59 1D 13 FF 47 40 E2 E2 F7 92 49 6D 4B 60 12 B6
    36 AE ED 4A 3E D2 43 2B 42 2C F4 6C AE D0 9C 7C BB 9E 69 8E 08 50 E2 63 55 7C CB AE 5D
    00 00
<<  6E 00
Card error SW1/SW2=6e00 - Checking error: Class not supported
```

Abbreviations

Table of Abbreviations	
Abbreviation	Definition
AID	Application identifier – sequence of bytes (5_{dec}-16_{dec}) to identify the application on the card.
ANSI	American National Standards Institute
APDU	Application Protocol Data Unit - The application protocol data unit of the chip.
ASCII	American Standard Code for Information Interchange - The standard 7-bit code table to present digitally the English alphabet and other keyboard symbols.
ASN.1 BER	Abstract Syntax Notation One Basic Encoding Rules. Often called as Tag Length Value (TLV) formatting rules.
ASN.1 DER	Abstract Syntax Notation One Distinguished Encoding Rules.
HEX	The symbol for the hexadecimal numeral system.
BCD	The presentation of numbers in a way that the first 4 and last 4 bits of each byte could be viewed as separate digits ranging 0 through 9 in the hexadecimal numeral system.

Table of Abbreviations	
Abbreviation	Definition
CPLC	Card Production Life Cycle – Data containing information about chip fabricator, operating system, chip serial number, personalisers, personalising equipment, personalising dates, etc.
BIN	The symbol for the binary number system.
DEC	The symbol for the decimal number system.
ICC	Integrated Circuit Card
IFD	Interface Device
K	Key
KID	The key identifier byte in key reference data.
KST	Key search type
KV	The key version byte in key reference data.
LSB	The least significant bit.
MSB	The most significant bit.
FID	The file identifier.
FCI	File Control Information. Data FCP+FMD. (ISO 7816-4)
FCP	File Control Parameters. The given parameters include a list of logical, structural and security attributes. (ISO 7816-4)
FMD	File Management Data (ISO 7816-4)
POS	Point of sale
RND	Random
TLV	Tag Length Value – Data formatting method.

Terms

Table of Terms	
Term	Definition
Authentication	The procedure for confirming the origin of somebody or something.
Authorisation	The procedure during which it is established whether the given person has the rights for the particular operation.
Digital signing	The procedure that results as an unique verifiable data.
Hash	A unique set of bits corresponding to a specific set of data.
PIN – code	Personal Identification Number - A code consisting of letters and/or digits used to authenticate the user identity.
Verification	The procedure for verifying the data validity.

References

Kaliski, B., "PKCS #1: RSA Encryption Version 1.5", RFC 2313, March 1998.

ISO/IEC 7816-3 (2006): “Identification cards - Integrated circuit cards - Part 3: Cards with contacts — Electrical interface and transmission protocols".

ISO/IEC 7816-4 (2013): “Identification cards - Integrated circuit cards - Part 4: Organization, security and commands for interchange”.

ISO/IEC 7816-8 (2014): “Identification cards - Integrated circuit cards - Part 8: Commands for security operations”.

ISO/IEC 18033-3 (2010): “Information technology - Security techniques - Encryption algorithms - Part 3: Block ciphers”.

ISO/IEC 9797-1 (1999): “Information technology - Security techniques - Message Authentication Codes (MACs) - Part 1: Mechanisms using a block cipher”.

Java Card ™ Specifications Version 2.2.2, March 2006

Table of tables

Table of figures

Document version history

Table for Document Version History		
Date	Version	Changes/notices
03.07.2012	0.1	Draft and document barebones
24.07.2012	0.2	APDU commands and responses
31.07.2012	0.3	Card application objects, their details and started with general operations
02.08.2012	0.4	Card application general operations
21.08.2012	0.5	Card application management operations and APPENDIX with APDU log.
10.09.2012	0.6	Clarifications
03.10.2012	0.7	▪ Removed chapter T0 and T1 protocols ▪ Improved chapter 1.4 Card application file system structure ▪ Grammar and wording corrections ▪ Few procedure parameters corrections ▪ Added references ▪ Changed name of chapter Decrypting session key to Decrypting public key encrypted data and improved given chapter ▪ Clarified using of RSA exponent in case of different JavaCard platforms

Table for Document Version History		
Date	Version	Changes/notices
21.01.2013	0.8	▪ Added chapters 1.3 Identifying the card application, 7.1 Card possible response in case of protocol T0 and 7.2.4 GET RESPONSE. ▪ Grammar and wording corrections ▪ Improved examples ▪ Fixed secret keys reference numbers
13.02.2013	0.81	▪ SSL is obsolete. Changed to TLS. ▪ Changed key reference identifiers to be backward compatible
02.04.2013	0.9	▪ Term TLS mistyping fixed. ▪ Improved chapter 1.3 Identifying the card application. ▪ Improved description for MANAGE SECURITY ENVIRONMENT command. ▪ Fixed chapters 3.1, 3.2.1, 3.2.2 and 3.3 by according to previous mentioned improvement.
15.05.2013	0.91	▪ Clarifications for protocol T0. ▪ Clarifications for application identification.
21.05.2013	1.0	Specification release
10.06.2013	1.05	▪ Document reference links fixed ▪ Wording improvements
29.09.2013	1.10	▪ Clarification about ATR historical bytes
11.03.2014	1.20	▪ Added introductory section " Differentiation to previous versions of EstEID card application