Document: List of APDU commands for Fiscal Smart Card		
Version: 1.5	Date: 10/02/2017	Created: J.G.

Table of Contents

Index of Tables

1 OVERVIEW

In this document the protocol that enables the correct communication between the SDC and the Fiscal Smart Card (SAM) is described. The commands in this document are the ones exchanged during the regular operation, that is, after the card has been enrolled and provided with a valid identity: these other commands are supposed to be known only by the Avatar backbone (ATAX), and have higher security requirements: they are sent in encrypted form and some of them are authenticated with digital certificates.

2 PROTOCOL DESCRIPTION

2.1 Physical aspects

The interface between the SDC and the SAM follows the standard ISO 7816-3. Avatar SAM has been tested and is known to work with communication protocol T1, as defined in section 11 in the aforementioned norm.

2.2 Communication SDC-SAM

The communication between the SDC and the SAM follows a Command-Response model. All the commands are initiated by the SDC and no other command is sent until a response (or a timeout) is received from the SAM.

More specifically, the SDC and the SAM exchange commands and responses by using Application Protocol Data Units (APDU) in command-response pairs. The format of the APDU's is defined in ISO specifications 78163-3 and 7816-4.

2.3 Authentication

All the commands issued by the SDC must be authenticated with a Pin, that is an alphanumeric sequence. Currently (i.e. December 2016), the sequence has 6 characters.

The pin authentication mechanism works like this:

1. SDC sends to the SAM the command VERIFY_PIN, including the pin sequence. It is up to the SDC developer to decide how frequently it asks the user for the PIN.
2. If the pin is correct, the SAM returns a random 8 Byte sequence (NONCE) in response to the command; otherwise the SAM throws an exception and returns the error code SW_UNAUTHORIZED (0x9C06).
3. After successful authentication, the SDC will append the nonce to all the subsequent commands.

2.4 Protocol overview

Before sending a command to the applet; the Avatar Applet has to be selected through the SELECT_AVATAR_APPLET command; several applications might be running inside the same physical secure element. If there is a power cycle of the card in the middle of a regular operation, the nonce is undefined, and therefore the authentication process has to be initiated again.

The SAM has two main functions:

- Digitally sign an piece of data provided by the SDC.
- Return to the SDC some internal data.

The set of commands described in COMMAND DESCRIPTION allow the SDC to ask the card to perform these two functions in different ways: there are commands that ask the SAM to sign an invoice; others that ask the SAM to provide its internal data, and also there are other commands for performing both functions at the same time.

Finally, there are other commands involved in the audit procedure. There is a limit in the card for the number of regular signing operations that it is allowed to perform with no further checks (except some basic ones). If this limit is reached, the card will stop signing and will reject all the signing command with exception SW_MAX_UNAUDITED_TRANSACTIONS (0x9C60). Therefore, some procedure is required for resuming the regular operations. The audit procedure works like this:

-The SDC sends START_AUDIT to the card. The SAM returns a random token together with its internal data.

-ATAX signs this token its private key. The SAM decodes it, and checks if the encrypted data contains the random token that it has previously issued.

3 COMMAND DESCRIPTION

In this section a description of all the commands that the SDC can send to the SAM is given. Any other combination will throw an exception, and the card will return an error code. Each command is divided into three components:

- COMMAND: APDU sent from the SDC to the SAM
- RESPONSE: Response from the SAM to the SDC when no error has arisen during the processing of the COMMAND.
- ERRORS: List of possible errors that can occur while processing the COMMAND.

The commands follow the APDU structure defined in ISO7816-3 and ISO7816-4. The type of command, according to section 12 in ISO7816-3, is also provided.

Notation: In the following sections, the operator* || *is used several times. It has to be interpreted as "Concatenation". As an example:

0x0102*||*0x0304 = 0x01020304

3.1 SELECT_AVATAR_APPLET

3.1.1 Description.

This command tells the SAM Card Manager to relay all the subsequent commands to the Avatar Applet. The Card Manager can be viewed as the resident operative system in the card. Different applications, each one having its own AID and cryptographic keys, can be present in the SAM.

3.1.2 Fields.

Command type = 3S.

COMMAND

CLA	INS	P1	P2	Lc	Data
0x00	0xA4	0x04	0x00	0x09	AID

Table 3.1: SELECT_AVATAR_APPLET Command

RESPONSE

SW1	SW2
0x90	0x00

Table 3.2: SELECT_AVATAR_APPLET Response

3.1.3 Formal description

```
SELECT_AVATAR_APPLET = {
COMMAND = {
        CLA SELECT_CLA,
        INS  SELECT
        SELECT_BY_NAME,
        0x00,
        Blength(AID),
        AID
```

},

RESPONSE = {

SUCCESS

},

ERRORS = {

ISO7816_EXCEPTIONS

}

}

3.1.4 Example

COMMAND =

Cla:

00

Ins:

A4

Select by Name:

04

P2:

00

Length:

09

Avatar AID:

41 76 61 74 61 72 00 00 00

RESPONSE =

Success:

90 00

3.2 VERIFY_PIN

3.2.1 Description.

Login command, it’s the mechanism whereby the SDC tries to establish an authenticated session with the secure element.

3.2.2 Fields

Command type = 4S.

COMMAND

CLA	INS	P1	P2	Lc	Data
0xB0	0x42	0x00	0x00	0x06	PIN

Table 3.3: VERIFY_PIN Command

RESPONSE

Data	SW1	SW2
NONCE	0x90	0x00

Table 3.4: VERIFY_PIN Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x02	SW_AUTH_FAILED	Incorrect pin
0x9C	0x0C	SW_IDENTITY_BLOCKED	Max number of login attempts reached

Table 3.5: VERIFY_PIN Errors

3.2.3 Formal description

```
VERIFY_PIN = {
COMMAND = {
        CLA PIN_CLA,
```

INS VERIFY_PIN,

IDENTITY_NUMBER,

0x00,

Blength(PIN),

PIN

},

RESPONSE = {

NONCE,

SUCCESS

},

ERRORS = {

IDENTITY_EXCEPTIONS,

ISO7816_EXCEPTIONS

}

}

3.2.4 Example

This is the APDU for a login attempt with Pin 123456.

COMMAND =

Cla:

B0

Ins:

42

P1:

00

P2:

00

Length:

06

Pin:

31 32 33 34 35 36

RESPONSE =

Nonce:

B5 F1 B1 C3 BB 80 DF F3

Success:

90 00

3.3 SIGN_INVOICE

3.3.1 Description

This is one of the two methods for requesting the SAM to sign an invoice. The response to this command only contains the signature of the invoice, no other internal data present in the SAM is added to the response. The Lc in the command has the extended length format (3 Bytes long).

The data in the command is formed by concatenating several values.

- 0x000501 : Fixed values.
- 2 Bytes Length of the subsequent request
- request: also a composed value with these fields:
  1. Transaction type (sale or refund)
  2. Transaction mode (normal, training or proforma)
  3. Transaction values (amount and tax)
  4. Invoice.
- Nonce

3.3.2 Fields

Command type = 4E.

COMMAND

CLA	INS	Key Number	Op Type	Extended Lc	Data
0xB0	0x38	0x00	0x04	Variable	0x000501||SLENGTH(REQUEST)|| REQUEST||NONCE

Table 3.6: SIGN_INVOICE Command

RESPONSE

Data	SW1	SW2
LSIMPLE_SIGNATURE	0x90	0x00

Table 3.7: SIGN_INVOICE Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x06	SW_UNAUTHORIZED	Incorrect nonce
0x9C	0x45	SW_PENDING_SIGNATURE	A previous signature has not been read yet.
0x9C	0x60	SW_MAX_UNAUDITED_TRANSACTIONS	No more signatures allowed, audit procedure needed.
0x9C	0x52	SW_REFUND_NOT_ALLOWED	The amount to be refunded exceed the total sales amount.
0x9C	0x51	SW_OVERFLOW	Maximum value for a counter exceeded.

Table 3.8: SIGN_INVOICE Errors

3.3.3 Formal description

```
SIGN_INVOICE = {
COMMAND = {
        CLA  = PIN_CLA,
        INS   = SIGN_INVOICE,
        KEY_NUMBER = 0x00,
        OPERATION_TYPE = ONE_STEP,
        Elength (DATA),
        DATA = {
                CIPHER_MODE  = CM_RSA_NOPAD
                OPERATION = SIGN,
                DATA_LOCATION = DL_APDU,
                SLENGTH(REQUEST),
                REQUEST,
                NONCE
        }
},
RESPONSE = {
```

LSIMPLE_SIGNATURE,

SUCCESS

},

ERRORS {

SIGNING_EXCEPTIONS

CRYPTOGRAPHIC_EXCEPTIONS,

TRANSACTION_EXCEPTIONS,

ARITHMETIC_EXCEPTIONS,

APPLET_EXCEPTIONS,

ISO7816_EXCEPTIONS

}

}

3.3.4 Example

This is a SIGN_INVOICE command example having with the following arguments:

TRANSACTION_TYPE = SALE

TRANSACTION_MODE = NORMAL

TRANSACTION_AMOUNT = 10.05

TAX_AMOUNT = 1.25

INVOICE = “invoice”

NONCE = 0xB90EF2E40D7C8F5D

COMMAND =

Cla:

B0

Ins:

38

Key Number:

00

One Step Operation:

04

Extended Length:

00 00 21

Fixed data header:

00 05 01

Request length:

00 14

Request:

Sale:

00

Normal mode:

00

Transaction value length:

0A

Amount:

04 01 10 01 05

Tax:

04 01 01 01 25

Invoice:

69 6E 76 6F 69 63 65

Nonce:

B9 0E F2 E4 0D 7C 8F 5D

RESPONSE =

Signature length:

01 00

Signature:

4C 53 8C 6D 96 BC BD EF C9 EE C2 45 79 47 74 3D A9 21 C8 2A 11 6A 6C
19 CA 12 2F 94 AA DD 4F D3 A7 32 70 82 5C DF 8C 1E 91 99 46 1E 4E 4F
7F B2 7A 4C A2 C1 55 7C 2F D7 0F A7 D6 92 0A D1 DA 89 25 98 06 F6 A1
EB 87 88 EF 50 5D D2 3C 71 09 C0 C0 F8 D6 D0 30 29 06 83 B6 76 52 E8
43 11 92 07 5D F3 74 BA AF 2E 93 5F A6 5D F5 3C 7C 2E A8 74 0D 82 FA
02 96 1E 3A 64 4A 80 18 5C A6 AF A2 08 BA 28 68 AC 14 85 DC 84 D8 5D
46 C6 92 F6 F5 65 9A 94 D1 E1 A8 26 5C 48 5C 20 C3 FB 10 1B 2A 20 05
DA D9 9A EE F9 0E 0C 66 6E EE 97 2B F0 22 0A 6F DE 02 AC 9E 1B 71 27
D6 1E B8 52 42 A5 EF 99 F0 FA C8 B4 1D AA 9A 6E 5A 04 97 C4 F4 54 18
8F 49 E5 2B 05 DC C0 9F B3 D8 7E E1 7D 6E 01 20 9B 58 21 A7 76 A5 AF
86 8D 44 59 1F 04 60 31 00 24 9B CF C3 23 B9 62 4A 3C 9F 12 15 FD 64
7B B3 E1

Success:

90 00

3.4 SIGN_INVOICE_SHORT

3.4.1 Description

This is a variant of SIGN_INVOICE for devices not supporting extended length APDU's. The only difference in the COMMAND APDU is the LC field, that now is a single Byte long. The response however is completely different: instead of the signature itself, it returns the length of the signature. The latter can be retrieved afterwards with the command READ_SIGNATURE. No further signature command are processed until the signature reading is complete. In that case the exception SW_PENDING_SIGNATURE is thrown by the applet.

3.4.2 Fields

Command type = 4S.

COMMAND

CLA	INS	Key Number	Op Type	Lc	Data
0xB0	0x38	0x00	0x04	Variable	0x000501||SLENGTH(REQUEST)|| REQUEST||NONCE

Table 3.9: SIGN_INVOICE Command

RESPONSE

Data	SW1	SW2
SLENGTH (LSIMPLE_SIGNATURE)	0x90	0x00

Table 3.10: SIGN_INVOICE Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x06	SW_UNAUTHORIZED	Incorrect nonce
0x9C	0x45	SW_PENDING_SIGNATURE	A previous signature has not been read yet.
0x9C	0x60	SW_MAX_UNAUDITED_TRANSACTIONS	No more signatures allowed, audit procedure needed.
0x9C	0x52	SW_REFUND_NOT_ALLOWED	The amount to be refunded exceed the total sales amount.
0x9C	0x51	SW_OVERFLOW	Maximum value for a counter exceeded.

Table 3.11: SIGN_INVOICE Errors

3.4.3 Formal description

```
SIGN_INVOICE_SHORT = {
COMMAND = {
        CLA  = PIN_CLA,
        INS   = SIGN_INVOICE_SHORT,
        KEY_NUMBER = 0x00,
        OPERATION_TYPE = ONE_STEP,
        Blength (DATA),
        DATA = {
                CIPHER_MODE  = CM_RSA_NOPAD
                OPERATION = SIGN,
                DATA_LOCATION = DL_APDU,
                SLENGTH(REQUEST),
                REQUEST,
                NONCE
        }
},
RESPONSE = {
```

SLENGTH(LSIMPLE_SIGNATURE),

SUCCESS

},

ERRORS {

SIGNING_EXCEPTIONS

CRYPTOGRAPHIC_EXCEPTIONS,

TRANSACTION_EXCEPTIONS,

ARITHMETIC_EXCEPTIONS,

APPLET_EXCEPTIONS,

ISO7816_EXCEPTIONS

}

}

3.4.4 Example

This is a COMMAND with the following arguments:

INVOICE = “invoice”

TRANSACTION_TYPE = SALE

TRANSACTION_MODE = NORMAL

TRANSACTION_AMOUNT = 10.05

TAX_AMOUNT = 1.25

NONCE = 0xB90EF2E40D7C8F5D

COMMAND =

CLA:

B0

INS:

38

Key Number:

00

One Step Operation:

04

Length:

21

Fixed Data Header:

00 05 01

Request Length:

00 14

Request:

Sale:

00

Normal:

00

Transaction value length:

0A

Amount:

04 01 10 01 05

Tax:

04 01 01 01 25

Invoice:

69 6E 76 6F 69 63 65

Nonce:

B9 0E F2 E4 0D 7C 8F 5D

RESPONSE =

Signature length:

01 02

Success:

90 00

3.5 SIGN_INVOICE_T

3.5.1 Description

This is a variant of the SIGN_INVOICE command. The only difference in the command is the INS field (0x39 instead of 0x38). This instruction code indicates the SAM that it has to perform a signature as in the previous command, but the response structure is completely different. The response contains a concatenation of the following fields:

- Global Counters (see GLOBAL_COUNTERS)
- SHA1(Counters) (see COUNTERS)
- Signature(Invoice||SHA1(Counters)) . The signature is done with the private key of the SAM.
- Encryption(Counters) (see COUNTERS)

3.5.2 Fields

Command type = 4E.

COMMAND

CLA	INS	Key Number	Op Type	Extended Lc	Data
0xB0	0x39	0x00	0x04	Variable	0x000501||SLENGTH(REQUEST)|| REQUEST||NONCE

Table 3.12: SIGN_INVOICE_T Command

RESPONSE

Data	SW1	SW2
SLENGTH(FULL_SIGNATURE)||FULL_SIGNATURE	0x90	0x00

Table 3.13: SIGN_INVOICE_T Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x06	SW_UNAUTHORIZED	Incorrect nonce
0x9C	0x45	SW_PENDING_SIGNATURE	A previous signature has not been read yet.
0x9C	0x60	SW_MAX_UNAUDITED_TRANSACTIONS	No more signatures allowed, audit procedure needed.
0x9C	0x52	SW_REFUND_NOT_ALLOWED	The amount to be refunded exceed the total sales amount.
0x9C	0x51	SW_OVERFLOW	Maximum value for a counter exceeded.

Table 3.14: SIGN_INVOICE_T Errors

3.5.3 Formal description

```
SIGN_INVOICE_T = {
COMMAND = {
        CLA PIN_CLA,
        INS  SIGN_INVOICE_T,
        0x00,
        ONE_STEP,
        Elength (DATA),
        DATA = {
                CIPHER_MODE  = CM_RSA_NOPAD
                OPERATION = SIGN,
                DATA_LOCATION = DL_APDU,
                SLENGTH (REQUEST),
                REQUEST,
                NONCE
        }
},
```

RESPONSE = {

SLENGTH(FULL_SIGNATURE),

FULL_SIGNATURE ,

SUCCESS

},

ERRORS = {

SIGNING_EXCEPTIONS,

CRYPTOGRAPHIC_EXCEPTIONS,

TRANSACTION_EXCEPTIONS,

ARITHMETIC_EXCEPTIONS,

APPLET_EXCEPTIONS,

ISO7816_EXCEPTIONS

}

}

3.5.4 Example

This example has with the following arguments:

INVOICE = “invoice”

TRANSACTION_TYPE = SALE

TRANSACTION_MODE = NORMAL

TRANSACTION_AMOUNT = 10.05

TAX_AMOUNT = 1.25

NONCE = 0xB90EF2E40D7C8F5D

COMMAND =

Cla:

B0

Ins:

39

Key Number:

00

One Step Operation:

04

Extended Length:

00 00 21

Fixed Data Header:

00 05 01

Request length:

00 14

Request:

Sale:

00

Normal:

00

Transaction value length:

0A

Amount:

04 01 10 01 05

Tax:

04 01 01 01 25

Invoice:

69 6E 76 6F 69 63 65

Nonce:

B9 0E F2 E4 0D 7C 8F 5D

RESPONSE =

Response length:

02 28

Global Counters length:

00 0C

Global Counters:

00 00 00 00 00 02 00 00 00 00 00 02

Counters Hash length:

00 14

Counters Hash:

EB 1B 62 27 1A D9 CC C2 02 F6 82 C7 CE 8E 30 72 FE DF B7 07

Signature length:

01 00

Signature:

51 B8 E0 3F 81 FF F6 A4 C1 1B 69 6A F5 39 E2 46 D8 D1 33 27 8E 10 37
6D D7 61 88 3F 84 96 BD EA 17 70 E7 F1 4B 1C 3F 2E 23 7F F2 01 C4 CE
43 2B 5B 28 99 87 C3 52 19 2C BE 79 1B 1A 16 B8 C5 9A 2E 9A 15 8B 36
0B F4 95 AC AF 5A 74 0A 21 C2 1F EE 1C 19 6A B5 AE A8 CF D5 5A CE 24
45 A6 88 8F 06 77 5E EA 94 CE 17 66 4D AF 63 7E 0D AB 3F 11 E2 5D 9D
FF E0 EA 44 96 F5 42 A6 9E 58 41 CB C7 37 63 32 54 99 B6 E6 7C D7 90
E2 C5 55 7D AD 5E 77 25 DD 3D B8 C1 14 CD 1C B3 69 C4 38 D7 92 1F 05
1C 1B BB 48 CD B2 13 BD 93 F6 59 59 8A 65 89 28 CA 55 01 AA 31 AD C0
AF B4 D0 40 CE 92 A4 E1 5C 13 A6 8C E5 78 08 2B 58 7C D3 FF 79 78 59
17 2B 70 62 B2 CE 06 F9 C0 54 20 E3 C8 C5 F3 FC 54 AD F6 41 62 5A 65
7B C8 83 C0 7F C0 57 8A 32 FA AD F4 3A AF AB 2C D2 5E 3D 0F 6F 42 FD
0C 56 2C

Encrypted Counters Length:

01 00

Encrypted Counters:

70 E1 DB D1 15 CF 81 A3 7F 39 9B 17 A9 BF 60 9E C2 4B 06 54 F9 38 7A
23 C6 5D 68 4B D4 A1 15 B2 DC 11 9E 4B 6C 8B EC E5 B6 A4 38 61 4F BA
B9 FA 0A E1 71 02 B4 C3 EA F3 75 CA EE 0F 19 21 C2 2F 66 91 91 4C 30
2B C2 24 48 76 98 1E 11 8A 4F D9 D4 EE BC 65 0D F8 B6 D3 9E 21 BC 11
88 46 79 C6 F3 BF 63 95 A8 20 82 E5 72 27 BF 34 C8 9A BF 1F 3A 97 6F
52 65 7A 01 59 5E 77 C4 95 71 D7 2B EB F8 00 BF 39 A9 61 36 C7 D6 95
89 35 F3 83 49 22 6A B3 A6 38 B1 20 CE B0 DE D0 3A F9 28 99 C0 47 F7
D9 A9 FD 48 85 41 F0 E4 92 D8 B0 8E E0 4D 1B EA 64 90 AE 9D BE B3 22
CF F7 02 40 F3 05 DA F1 EF D6 97 30 CD B7 D6 36 B7 ED A9 0F BD B4 47
95 86 15 B1 5C E3 31 CA 92 30 C6 3A 5D F3 BD 8F D6 C1 ED A8 86 97 42
41 9A 3E 69 1E 23 17 57 D2 DC 7B F8 22 07 DF 4E A2 DE 5A 76 76 F1 59
31 92 6F

Success:

90 00

3.6 SIGN_INVOICE_T_SHORT

3.6.1 Description

This command is the short variant of SIGN_INVOICE_T for devices not supporting extended APDU's. See command SIGN_INVOICE_SHORT for details. The only difference with the SIGN_INVOICE_T command is the Lc field: in this command the field is only one Byte long. The difference comes is in the response: only the length of the full signature is provided; the SDC has to read it from the SAM by issuing one or more READ_SIGNATURE commands.

3.6.2 Fields

Command type = 4S.

COMMAND

CLA	INS	Key Number	Op Type	Lc	Data
0xB0	0x39	0x00	0x04	Variable	0x000501||SLENGTH(REQUEST)|| REQUEST|| NONCE

Table 3.15: SIGN_INVOICE_T_SHORT Command

RESPONSE

Data	SW1	SW2
SLENGTH(FULL_SIGNATURE)	0x90	0x00

Table 3.16: SIGN_INVOICE_T_SHORT Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x06	SW_UNAUTHORIZED	Incorrect nonce
0x9C	0x45	SW_PENDING_SIGNATURE	A previous signature has not been read yet.
0x9C	0x60	SW_MAX_UNAUDITED_TRANSACTIONS	No more signatures allowed, audit procedure needed.
0x9C	0x52	SW_REFUND_NOT_ALLOWED	The amount to be refunded exceed the total sales amount.
0x9C	0x51	SW_OVERFLOW	Maximum value for a counter exceeded.

Table 3.17: SIGN_INVOICE_T_SHORT Errors

3.6.3 Formal description

```
SIGN_INVOICE_T_SHORT = {
COMMAND = {
        CLA PIN_CLA,
        INS  SIGN_INVOICE_T_SHORT,
        0x00,
        ONE_STEP,
        Blength(DATA),
        DATA = {
                CIPHER_MODE  = CM_RSA_NOPAD
                OPERATION = SIGN,
                DATA_LOCATION = DL_APDU,
                SLENGTH (REQUEST),
                REQUEST,
                NONCE
        }
},
RESPONSE = {
```

SLENGTH(FULL_SIGNATURE),

SUCCESS

},

ERRORS = {

SIGNING_EXCEPTIONS,

CRYPTOGRAPHIC_EXCEPTIONS,

TRANSACTION_EXCEPTIONS,

ARITHMETIC_EXCEPTIONS,

APPLET_EXCEPTIONS,

ISO7816_EXCEPTIONS

}

}

3.6.4 Example

This is a COMMAND with the following arguments:

RSA_KEY_SIZE = 256 (0x0100)

INVOICE = “invoice”

TRANSACTION_TYPE = SALE

TRANSACTION_MODE = NORMAL

TRANSACTION_AMOUNT = 10.05

TAX_AMOUNT = 1.25

NONCE = 0xB5F1B1C3BB80DFF3

COMMAND =

Cla:

B0

Ins:

39

Key Number:

00

One Step Operation:

04

Length:

21

Fixed Data Header:

00 05 01

Request Length:

00 14

Request:

Sale:

00

Normal:

00

Transaction Value Length:

0A

Amount:

04 01 10 01 05

Tax:

04 01 01 01 25

Invoice:

69 6E 76 6F 69 63 65

Nonce:

B5 F1 B1 C3 BB 80 DF F3

RESPONSE =

Full Signature Length:

02 2A

Success:

90 00

3.7 READ_SIGNATURE

3.7.1 Description

Some devices do not support the Extended APDU format; therefore some method is needed that enables the SDC to read the signature requested by the previous commands in small chunks. Reading of the last fragment is signaled by setting the field LAST_CHUNK to 0x01. In the Data field of the command, the SDC indicates the offset (expressed as a short) starting from which it wants to read a CHUNK_SIZE of the signature object. The SAM does not track this operation, it's the SDC responsibility to read all the data in the signature.

3.7.2 Fields

Command type = 4S.

COMMAND

CLA	INS	LAST_CHUNK	CHUNK_SIZE	Lc	Data
0xB0	0x55	True: 0x01 False: 0x00	Variable	0x0A	(short)(Offset)||NONCE

Table 3.18: READ_SIGNATURE Command

RESPONSE

Data	SW1	SW2
SIGNATURE_CHUNK	0x90	0x00

Table 3.19: READ_SIGNATURE Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x06	SW_UNAUTHORIZED	Incorrect nonce
0x9C	0x46	SW_SIGNATURE_READ_FORBIDDEN	There is no signature to be read.

Table 3.20: READ_SIGNATURE Errors

3.7.3 Formal description

READ_SIGNATURE = {

COMMAND = {

CLA PIN_CLA,

INS READ_SIGNATURE,

LAST_CHUNK,

CHUNK_SIZE,

Blength(DATA),

DATA = {

SIGNATURE_OFFSET,

NONCE

}

},

RESPONSE = {

SIGNATURE_CHUNK,

SUCCESS

},

ERRORS {

APPLET_EXCEPTIONS,

SIGNING_EXCEPTIONS,

ISO7816_EXCEPTIONS

}

3.7.4 Example

In this example, the first step is to send a signature COMMAND using the same COMMAND as in the example for command SIGN_INVOICE_SHORT. The first response indicates the length of the signature object. Then the signature is retrieved in two chunks.

COMMAND 1 (SIGN_INVOICE_SHORT):

B0 38 00 04 21 00 05 01 00 14 00 00 0A 04 01 10 01 05 04 01 01 01 25 69 6E 76 6F 69 63 65 E9 6D D1 40 62 31 63 44

RESPONSE 1:
01 02 90 00

COMMAND 2 (READ_SIGNATURE; Initial chunk):

Cla:
B0

Ins:

55

No Last Chunk:

00

Chunk Size:

FA

Length:

0A

Offset:

00 00

Nonce:

E9 6D D1 40 62 31 63 44

RESPONSE 2:

Signature Chunk:
01 00

92 CE A0 4E F0 32 72 3B 07 CC B8 28 EB 41 B5 D6 EE 60 55 FA 4F 0A CC
E0 71 F8 98 9A 80 C8 0A 2E 73 5C DF E6 14 AC 9F 7E 6D 1C D5 C2 6F F0
2C B6 81 B3 BD 9E 03 5D 6A 17 5A D8 1A F7 3D 99 74 4B DB 1D 96 BE 41
4C CF E0 7E 8B C7 DE 11 64 8C 01 F2 16 5F 4B 8B 96 07 67 F8 4C 02 D5
87 F7 11 FB 3F DB B0 2A 35 54 4E 11 38 AA BE F7 1A EF 6B 50 AE 40 73
2C 32 C8 39 41 E5 C6 BB 07 2C 0F 3E 2F B7 A1 99 BC 8F 5C 19 3C 50 F4
B5 32 39 59 C1 10 03 BC 69 76 77 4E B5 87 64 CC E1 E5 83 09 75 56 A0
62 2B 63 50 78 B7 29 19 B6 FD 9A 89 46 78 46 30 78 4D 71 19 8C D4 10
FB 80 F6 88 94 74 1E B3 F7 6F 7A 23 A0 8A 3D 1D E0 74 4A 66 A2 97 74
B3 87 6E 09 1E BD 4D A8 8B 37 5E 01 34 73 9A C4 01 05 C3 AE 53 66 8E
9E 92 BA 16 41 28 98 CC 8E 15 86 77 94 C2 D7 3E 59 81

Success:

90 00

COMMAND 3 (READ_SIGNATURE; Final chunk):

Cla:
B0

Ins:

55

Last Chunk:

01

Chunk Size:

08

Length:

0A

Offset:

00 FA

Nonce:

E9 6D D1 40 62 31 63 44

RESPONSE 3:

Signature Chunk:
D4 95 F8 0B 3E BD CC 6B

Success:

90 00

3.8 START_AUDIT

3.8.1 Description

This is the command that initiates a Proof Of Audit (PoA). Once the SAM receives the command, and after making some initial checks, it starts building an object (auditData), composed by the following elements:

-The SerialID of the certificate present in the card.

-TRANSACTIONS_COUNTER.

-LAST_AUDITED_TRANSACTION_COUNTER.

-The COUNTERS object.

-A random token of 32 Bytes.

This piece of information is returned to the card, although not directly. For privacy and integrity reasons it is encrypted and signed; the precise format is detailed in the next sections.

3.8.2 Fields

Command type = 4E.

COMMAND

CLA	INS	P1	P2	Extended LC	Data
0xB0	0x78	0x00	0x00	0x000008	NONCE

Table 3.21: START_AUDIT Command

RESPONSE

Data	SW1	SW2
Slength(AUDIT_DATA_RESPONSE)|| AUDIT_DATA_RESPONSE	0x90	0x00

Table 3.22: START_AUDIT Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x06	SW_UNAUTHORIZED	Incorrect nonce
0x9C	0x61	SW_START_AUDIT_FORBIDDEN	There is an ongoing audit data read operation.

Table 3.23: START_AUDIT Errors

3.8.3 Formal description

START_AUDIT = {

COMMAND = {

CLA PIN_CLA,

INS START_AUDIT

00,

00,

Elength(NONCE),

NONCE

},

RESPONSE = {

Slength(AUDIT_DATA_RESPONSE)||AUDIT_DATA_RESPONSE,

SUCCESS

},

ERRORS {

ISO7816_EXCEPTIONS,

AUDIT_EXCEPTIONS

}

}

3.8.4 Example

COMMAND =

CLA:

B0

INS:

78

P1/P2:

00 00

Extended Length:

00 00 08

Nonce:

BF 30 99 E5 E4 F8 D9 C5 02 06

RESPONSE =

Total Audit Response Length:
02 04

Length of Encrypted Audit Data:

01 00

Encrypted Audit Data:

55 24 DB 61 F1 A9 66 66 73 5B 5D 1D A2 22 5A B3 E7 8B FE 3E C1 67 A4
41 A3 E0 35 A0 52 D0 51 82 13 34 27 DF 9D 0F 7B 14 42 19 E5 0B 26 6D
28 10 38 D6 B9 7F 8D 7B D6 4C 39 34 E0 37 36 60 FD 5A 7F A9 A7 DE E9
13 7A 6A 55 31 19 24 CD 79 31 F0 93 F7 6E 56 2E 02 D1 A4 0F 1E FE FA
95 B2 01 5D CD 49 EA A6 3B D6 95 C3 9F 50 10 7D DA 59 5A 2D F0 55 60
01 5F 4A 31 23 29 8D B0 9C 3E 25 F4 6F 2F 55 6B A5 3A D2 4B 0D F7 D5
88 3F 7E 55 FB 66 3A BB 92 F5 7B C5 38 2A 76 CE F9 F5 28 A6 39 01 FB
36 2D 12 6B 8A 14 43 97 4F 2D EB D8 9A D3 54 95 8C 2D 7A 63 B7 6C A2
C2 37 F2 2F 5D A4 53 E4 C7 1A B7 DF 51 48 47 21 32 A2 82 9F AD FE 82
E5 C5 B1 65 29 C7 70 20 CE E2 8C F9 8D 4C E2 24 F1 0A E3 FB B0 4D 3A
87 B2 31 AD 4F 7F E9 3F B3 43 17 B6 41 06 4D F9 60 A7 AE D3 BB 54 6E
B9 27 66

Length of Signed Audit Data:

01 00

Signed Audit Data:

```
28 52 70 EF C1 2C 46 04 78 90 74 07 A7 73 43 6C 7F 5F E4 96 30 6D 83
44 8F 3F 42 C4 68 65 D5 32 FE 10 24 30 76 13 FE C5 98 1E 17 4F 6C 95
F7 BA 8E 40 CE 16 50 66 76 72 FA C2 10 E2 12 97 56 45 C9 CD 11 C4 FD
D3 45 B2 83 A0 F5 45 93 68 E2 21 23 76 0C B2 32 9C B6 0D 0B A4 74 43
81 00 70 79 9C F1 D4 41 91 72 C5 42 ED EB C4 92 20 B5 BB 0D DA 5A 21
72 48 E9 70 EF 92 46 CF AE 42 CE 55 40 81 4F B8 2E F0 9B 6B 01 16 00
D2 3E 25 77 A7 68 A5 E7 15 8F 44 2E 4C C2 DC E4 C8 44 0A AA FC B3 BB
87 B1 5B 47 2F FB 83 7E AD 04 73 97 99 BB 76 5A FF 1D 96 29 6C 19 E0
71 70 E1 88 03 63 5A 12 2A 36 02 EB 1B 91 00 28 72 7F EC 39 21 AB B9
16 16 80 C8 FA C6 87 66 C0 31 4A 20 E5 AB 95 12 0F F5 FF 67 E7 68 39
F1 72 4C 4E 4F FF 72 CB BD 6E 2A 6D 1A 7D 37 23 02 52 8D 57 FF 65 F5
C9 20 ED
```

Success:

```
90 00
```

3.9 START_AUDIT_SHORT

3.9.1 Description

This is the short version of START_AUDIT for devices that do not support the extended APDU format. It works in a similar way as SIGN_INVOICE_T_SHORT . This command, instead of returning the full audit data, returns the its length. The SDC will be responsible of rebuilding it by sending to the SAM the required number of READ_AUDIT commands.

3.9.2 Fields

Command type = 4S.

COMMAND

CLA	INS	P1	P2	LC	Data
0xB0	0x80	0x00	0x00	0x08	NONCE

Table 3.24: START_AUDIT_SHORT Command

RESPONSE

Data	SW1	SW2
SHORT_AUDIT_DATA_RESPONSE	0x90	0x00

Table 3.25: START_AUDIT_SHORT Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x06	SW_UNAUTHORIZED	Incorrect nonce
0x9C	0x61	SW_START_AUDIT_FORBIDDEN	There is an ongoing audit data read operation.

Table 3.26: START_AUDIT Errors

3.9.3 Formal description

START_AUDIT_SHORT = {

COMMAND = {

CLA PIN_CLA,

INS START_AUDIT_SHORT

00,

00,

Blength(NONCE),

NONCE

},

RESPONSE = {

SHORT_AUDIT_DATA_RESPONSE,

SUCCESS

},

ERRORS {

ISO7816_EXCEPTIONS,

AUDIT_EXCEPTIONS

}

}

3.9.4 Example

COMMAND =

CLA

B0

INS

80

P1/P2

00 00

Lc

08

NONCE

7E 77 89 31 7D D5 2C B0

RESPONSE =

AuditData Length

02 06

SUCCESS

90 00

3.10 READ_AUDIT

3.10.1 Description

This command is equivalent to READ_SIGNATURE, but instead of reading the signature of a transaction, it reads the audit data built by the card after a successful START_AUDIT_SHORT command. This command has to be sent a number of times until the full audit data is retrieved from the SAM. The SDC is responsible of asking for the right piece of data at each request. It also has to indicate to the SAM whether or not it is asking for the last piece of audit data. No other START_AUDIT_SHORT or START_AUDIT is allowed while there is some piece of audit data yet unread. In that case, the command is rejected with the exception SW_START_AUDIT_FORBIDDEN. This command is not allowed when no previous START_AUDIT_SHORT has been sent to the SAM; in that case the exception thrown is SW_AUDIT_READ_FORBIDDEN.

3.10.2 Fields

Command type = 4S.

COMMAND

CLA	INS	LAST_CHUNK	CHUNK_SIZE	LC	Data
0xB0	0x81	True: 0x01 False: 0x00	Variable	0x0A	AUDITDATA_OFFSET||NONCE

Table 3.27: READ_AUDIT Command

RESPONSE

Data	SW1	SW2
AUDIT_DATA_CHUNK	0x90	0x00

Table 3.28: READ_AUDIT Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x06	SW_UNAUTHORIZED	Incorrect nonce
0x9C	0x62	SW_AUDIT_READ_FORBIDDEN	No audit data in place.

Table 3.29: READ_AUDIT Errors

3.10.3 Formal description

READ_AUDIT = {

COMMAND = {

CLA PIN_CLA,

INS READ_AUDIT,

LAST_CHUNK,

CHUNK_SIZE,

Blength(AUDITDATA_OFFSET||NONCE),

AUDITDATA_OFFSET,

NONCE

},

RESPONSE = {

AUDIT_DATA_CHUNK,

SUCCESS

},

ERRORS {

ISO7816_EXCEPTIONS,

AUDIT_EXCEPTIONS

}

}

3.10.4 Example

COMMAND 1: START_AUDIT_SHORT

B0 80 00 00 08 DD 4D E4 3E 5A 9B 3D 74
RESPONSE 1:

Audit Data Length
02 06

Success

90 00

COMMAND 2: READ_AUDIT_DATA (Last Chunk = False; Offset = 0; Chunk size = 250)
B0 81 00 FA 0A 00 00 DD 4D E4 3E 5A 9B 3D 74

RESPONSE 2:

Audit Data Chunk:
02 04 01 00 66 5C 6C DF A3 9F C2 93 BB 1E 29 F4 BB 94 96 53 10 1C 91
E6 30 3C FA 9A F5 41 9C 1C CE EB 2B 91 B6 45 E7 42 10 B2 C1 5C 34 10
1C B5 C8 93 95 FE 84 2F 91 17 89 E2 C2 A1 C0 51 7B D6 AE 66 35 BC FF
F6 9C 0D DF B2 80 86 32 6B 27 BE E3 AD 43 6E E0 F7 4D F8 F6 A7 F6 59
EC F0 07 EA 7D 82 0F 52 89 D2 B4 24 41 B9 FD E1 25 BC 12 8D AA 60 14
25 69 EF 11 F6 A0 58 A7 97 B4 D1 ED 22 E0 09 2D 43 D2 F8 6F 18 CD 4C
9B 72 52 DF 08 59 71 D0 68 13 C1 38 FD E9 7A 09 55 8B D7 0E 05 4A C7
26 5E F0 52 40 68 CE F2 C4 FA 93 9E 1F F2 AC 21 76 80 05 8C 63 60 CE
C1 6B 56 B0 FD B0 57 64 F6 4E 4C 6A 33 87 07 50 41 5C 35 8A 69 59 83
DC 7E D1 B0 FD EA 77 BA A6 26 FF A5 0E D9 5E C2 1C CC 70 B2 E1 BF FF
39 CF AE C3 82 22 1C 2A FD 65 4E A0 A1 27 BC DB FE AD 39 E2

Success:

90 00

COMMAND 2: READ_AUDIT_DATA (Last Chunk = False; Offset = 250; Chunk size = 250)
B0 81 00 FA 0A 00 FA DD 4D E4 3E 5A 9B 3D 74

RESPONSE 2:

Audit Data Chunk:
03 3F BB F0 97 F5 BA 85 A0 27 01 00 62 EF 05 5F 12 A7 F8 3F AF C5 99
65 1A 01 E3 36 2D E4 C1 91 4A BD F4 57 EB 68 84 F8 16 6D A8 7C 63 EB
07 5E 04 0B A8 10 D9 49 1A 37 84 CA F2 B4 FB 48 AD BF 2D 3B 3D 5F 10
DC 0B 9A BB 52 AC 82 CB 23 F2 49 34 1B 07 9C 1D 0D 17 AE EE 78 49 BA
08 78 2C 9B C1 C2 DC D3 F4 D8 2C 4F 3F 43 BD C6 E3 90 3D F1 36 C5 87
F5 CD 1C BA 14 CF 09 49 B0 6A F8 05 E2 A5 96 4C A4 7F C3 9A AE FB CA
65 74 F5 09 4F F4 D8 3B BA 7B A9 00 1C 85 92 B5 56 FF A1 C3 B3 1F A0
AC 67 64 C3 85 0D 62 0B 17 5E 97 79 05 9B 1A FF 8A DB 16 2A 8E 62 D0
26 D3 4B 69 91 AA CF B6 43 80 FE CB E7 19 C8 94 CF FF 18 E9 0B 3B F2
C2 12 3E 24 13 A7 07 B5 03 5B 5D 15 B7 E9 E7 18 E0 9E 68 77 E9 E7 5B
78 FB 8B 29 E1 D3 02 D2 1A 4D 9D E2 16 64 E7 19 B3 88 9B CB

Success:

90 00

COMMAND 3: READ_AUDIT_DATA (Last Chunk = True; Offset = 500; Chunk size = 18)
B0 81 01 12 0A 01 F4 DD 4D E4 3E 5A 9B 3D 74

RESPONSE 3:

Audit Data Chunk:
CE 2D 6C 16 07 A8 27 EB EC 04 08 03 1C 0C 98 30 2E 0C

Success:

90 00

3.11 VERIFY_AUDIT

3.11.1 Description

This command has to be initiated by ATAX, and it is the latest stage in a Proof of Audit (PoA) procedure. ATAX has received an AUDIT_DATA_RESPONSE from the SAM. It makes some validations; and finally it sends this command to report the card the the audit data has been properly validated. The content of the command is the received TOKEN from the SAM in a response to a previous AUDIT_DATA_RESPONSE, but signed with ATAX private key. It may happen that this validation process takes some time; and that during that lapse another audit process has been initiated; in that case the command will fail with exception SW_WRONG_PROOF.

This command will also fail if it is sent before a START_AUDIT, in this case the exception will be SW_AUDIT_FORBIDDEN.

3.11.2 Fields

Command type = 2E.

COMMAND

CLA	INS	P1	P2	Extended Lc	Data
0xB0	0x79	0x00	0x00	Variable	STOKEN||NONCE

Table 3.30: VERIFY_AUDIT Command

RESPONSE

SW1	SW2
0x90	0x00

Table 3.31: VERIFY_AUDIT Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x06	SW_UNAUTHORIZED	Incorrect nonce
0x9C	0x66	SW_AUDIT_FORBIDDEN	No pending audit exists
0x9C	0x64	SW_WRONG_PROOF	The challenge presented is incorrect.

Table 3.32: VERIFY_AUDIT Errors

3.11.3 Formal description

VERIFY_AUDIT =

COMMAND = {

CLA PIN_CLA,

INS VERIFY_AUDIT,

0x00,

0x00,

Elength(STOKEN||NONCE),

STOKEN,

NONCE

},

RESPONSE = {

SUCCESS

},

ERRORS {

AUDIT_EXCEPTIONS,

ISO7816_EXCEPTIONS

}

3.11.4 Example

CLA:

B0

INS:

79

P1/P2:

00 00

Extended Length:

00 01 08

Signed Audit Token:

0A 86 E9 D3 91 1F 6B 72 B5 B3 74 36 FF 85 E1 1D 18 39 F0 39 BA BF F9
E4 B0 8E 58 40 47 E3 2F 54 C4 2F B9 A1 F5 FF 3D 2E 71 67 48 ED 38 24
D6 50 AB 3E 37 53 44 50 28 DF 21 BC 20 25 9A 89 77 96 87 25 1E 3B BB
04 CF 9D 99 BD 78 FF A6 BB CB 3C 4E B4 48 F7 98 7B 2D 0F 4A 55 12 B5
40 BD DE D0 BA FB 04 FD 40 9D 88 55 BD 33 AE CE 7F 96 D7 C9 E9 28 FB
06 A5 0B 14 6A 7A 1C DA 50 5D D8 85 91 2C 73 12 C4 49 B4 08 72 A9 F0
8D 2B 3C E8 E3 04 B4 ED AC A7 1B 49 01 44 0E FA 15 6D 25 7F E6 2D E3
84 42 33 B2 6E 9F D9 90 DC 36 AA C5 C0 42 A2 B0 89 AC 57 73 1E 7D A8
D4 D4 D2 B3 AB 91 17 0C 18 9B 59 BB D9 20 48 37 4C 0B F6 11 94 BC 42
0B 1A 4E 23 4D C7 2F 8B 3F B0 6F 68 A8 41 1F 1D AD 87 CE 7A 2F 57 3F
5B B0 B4 0B 1A 9D DB 73 E7 91 E1 BA DF 3A 0F 4E DA 3C DB AB 4E 1E 08
02 62 C2

Nonce:

67 24 71 1F 16 E1 50 55

RESPONSE =

Success:

90 00

3.12 VERIFY_AUDIT_SHORT

3.12.1 Description

This is the short version of the VERIFY_AUDIT command, for devices that do not support the extended length APDU format. Instead of sending STOKEN in a unique chunk, ATAX fragments it into several smaller ones. A simple mechanism is provided for indicating whether or not the current chunk in the last one.

3.12.2 Fields

Command type = 2S.

COMMAND

CLA	INS	LAST_CHUNK	CHUNK_SIZE	Lc	Data
0xB0	0x80	LAST_TOKEN_CHUNK	TOKEN_CHUNK_SIZE	Variable	STOKEN_CHUNK||NONCE

Table 3.33: VERIFY_AUDIT_SHORT Command

RESPONSE

SW1	SW2
0x90	0x00

Table 3.34: VERIFY_AUDIT_SHORT Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x06	SW_UNAUTHORIZED	Incorrect nonce
0x9C	0x66	SW_AUDIT_FORBIDDEN	No pending audit exists
0x9C	0x64	SW_WRONG_PROOF	The challenge presented is incorrect.

Table 3.35: VERIFY_AUDIT_SHORT Errors

3.12.3 Formal description

VERIFY_AUDIT_SHORT =

COMMAND = {

CLA PIN_CLA,

INS VERIFY_AUDIT_SHORT,

LAST_TOKEN_CHUNK,

TOKEN_CHUNK_SIZE,

Blength(STOKEN_CHUNK||NONCE),

STOKEN_CHUNK,

NONCE

},

RESPONSE = {

SUCCESS

},

ERRORS {

AUDIT_EXCEPTIONS,

ISO7816_EXCEPTIONS

}

3.12.4 Example

COMMAND 1: Verify_Audit_Short (Last = false, offset = 0, length = 240)

CLA:

B0

INS:

82

Last Chunk = False

00

Signed Token Chunk Length:

F0

APDU Length:

FA

Signed Token Chunk Offset:

00 00

Signed Token Chunk:

0A 86 E9 D3 91 1F 6B 72 B5 B3 74 36 FF 85 E1 1D 18 39 F0 39 BA BF F9
E4 B0 8E 58 40 47 E3 2F 54 C4 2F B9 A1 F5 FF 3D 2E 71 67 48 ED 38 24
D6 50 AB 3E 37 53 44 50 28 DF 21 BC 20 25 9A 89 77 96 87 25 1E 3B BB
04 CF 9D 99 BD 78 FF A6 BB CB 3C 4E B4 48 F7 98 7B 2D 0F 4A 55 12 B5
40 BD DE D0 BA FB 04 FD 40 9D 88 55 BD 33 AE CE 7F 96 D7 C9 E9 28 FB
06 A5 0B 14 6A 7A 1C DA 50 5D D8 85 91 2C 73 12 C4 49 B4 08 72 A9 F0
8D 2B 3C E8 E3 04 B4 ED AC A7 1B 49 01 44 0E FA 15 6D 25 7F E6 2D E3
84 42 33 B2 6E 9F D9 90 DC 36 AA C5 C0 42 A2 B0 89 AC 57 73 1E 7D A8
D4 D4 D2 B3 AB 91 17 0C 18 9B 59 BB D9 20 48 37 4C 0B F6 11 94 BC 42
0B 1A 4E 23 4D C7 2F 8B 3F B0 6F 68 A8 41 1F 1D AD 87 CE 7A 2F 57 3F
5B B0 B4 0B 1A 9D DB 73 E7 91

Nonce:

DD 4D E4 3E 5A 9B 3D 74

RESPONSE 1:

Success:

90 00

COMMAND 2 Verify_Audit_Short (Last = true; offset = 240, length = 16)

CLA:
B0

INS:

82

Last Chunk = True:

01

Signed Token Chunk Length:

10

APDU Length:

1A

Signed Token Chunk Offset:

00 F0

Signed Token Chunk:

E1 BA DF 3A 0F 4E DA 3C DB AB 4E 1E 08 02 62 C2

Nonce:

DD 4D E4 3E 5A 9B 3D 74

RESPONSE 2:

Success:

90 00

3.13 GET COUNTERS

3.13.1 Description

This command is used to retrieve the internal counters stored in the secure element. The card signs the counters object with ATAX public encryption key, and returns them to the SDC. The ATAX public key is stored in the SAM during the enrollment stage. This command does not require any argument. Obviously, only ATAX will be able to read the actual value of the internal counters.

3.13.2 Fields

Command type = 2E.

COMMAND

CLA	INS	P1	P2	Lc	Data
0xB0	0x76	0x00	0x00	8	NONCE

Table 3.36: GET_COUNTERS Command

RESPONSE

Data	SW1	SW2
ENCRYPTED_COUNTERS	0x90	0x00

Table 3.37: GET_COUNTERS Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x06	SW_UNAUTHORIZED	Incorrect nonce

Table 3.38: GET_COUNTERS Errors

3.13.3 Formal description

GET_COUNTERS =

COMMAND = {

CLA PIN_CLA,

INS GET_COUNTERS,

0x00,

0x00,

0x08,

NONCE

},

RESPONSE = {

ENCRYPTED_COUNTERS,

SUCCESS

},

ERRORS = {

CRYPTOGRAPHIC_EXCEPTIONS,

APPLET_EXCEPTIONS,

ISO7816_EXCEPTIONS

}

}

3.13.4 Example

COMMAND =

Cla:

B0

Ins:

76

P1:

00

P2:

00

Length:

08

Nonce:

62 91 BD E2 7A 57 5A 3C 00

RESPONSE:

06 D6 A4 D9 15 78 32 E2 1D 5D 1C 2B 0A 06 D9 75 1B 77 E5 6E 0B 5F D1
B5 D7 01 DD E2 AB B5 CD 77 23 B9 48 0A C9 5C AF 3F 4B 46 71 7D 1F 30
EE 5E A3 67 A6 97 6F C8 4E A3 CF 58 DC 3A 13 AB 43 4D 25 1A A4 73 1E
FC 52 89 87 9D A5 18 9B 46 43 FE 11 16 67 1E 24 95 D1 B7 27 EB C3 88
14 48 31 F3 FE BC 77 73 6B 17 B0 E8 EF AA F7 91 12 B9 6F 5B 77 C3 0E
49 0D BE 05 EE AF BD AC EA 6E 88 8C 3E 1A E9 49 B3 A1 E6 9F 54 86 A8
35 2A E9 3E 86 63 8C 68 70 BD 99 F6 04 AD D1 03 B4 41 B5 39 DB 14 AF
4B F4 E0 21 63 88 95 72 A1 E5 24 93 52 A9 EE 80 51 30 0F E5 47 B3 AD
FB 4E CF 02 32 DB 06 10 F7 C9 9D C4 82 9A C0 72 AB 84 82 56 95 0D 11
AE F4 4E BC CE EC BE E1 54 DE EC 57 C7 30 0B 73 8A 78 7E 42 89 F3 C6
92 96 C2 C2 72 D5 EE D3 C3 92 4F 1A 21 1B 35 6C 14 CF 33 63 15 1B D9
B4 BB B8 90 00

3.14 SIGN_DATA

3.14.1 Description

This command ask the SAM to sign with its private key a piece of data contained in the APDU.

3.14.2 Fields

Command type = 4E.

COMMAND

CLA	INS	P1	P2	ExtendedLC	Data
0xB0	0x33	0x00	0x00	Variable	RAW_DATA||NONCE

Table 3.39: SIGN_DATA Command

RESPONSE

Data	SW1	SW2
Slength(SIGNED_DATA)|| SIGNED_DATA	0x90	0x00

Table 3.40: SIGN_DATA Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x06	SW_UNAUTHORIZED	Incorrect nonce
0x9C	0x70	SW_SIGN_DATA_FORBIDDEN	There is an ongoing reading process

Table 3.41: SIGN_DATA Errors

3.14.3 Formal description

```
SIGN_DATA = {
COMMAND =       {
        CLA PIN_CLA,
```

INS SIGN_DATA,

0x00,

0x00,

Elength(RAW_DATA||NONCE),

RAW_DATA,

NONCE

},

RESPONSE = {

Slength(SIGNED_DATA),

SIGNED_DATA

SUCCESS

},

ERRORS {

ISO7816_EXCEPTIONS,

DATA_SIGNING_EXCEPTIONS

}

}

3.14.4 Example

Assuming that the raw data is "000102030405060708090A0B0C0D0E0F":

COMMAND =

CLA

B0

INS

35

P1/P2

00 00

Extended Length

00 00 18

Raw Data

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

Nonce:

54 77 86 19 AF 9F FE 77

RESPONSE =

Signed Data Length:

01 00

Signed Data:

**B8 7C 82 96 C7 4C F1 CA 81 BD 90 2D ED 6D 72 C8 D6 1C 15 64 29 F0 34
1C 07 34 8F 70 F4 C1 CA BD 46 55 D3 CA 3C B8 50 71 72 4C A9 6F 5E B3
1A DF D8 22 97 63 3B 4A 12 C8 34 2A 69 E5 DD 13 D9 60 2F FE DF A5 7F
2E 51 F6 7A F0 97 0E 54 68 BE CE 08 4F D5 45 3A BB 6E 55 C4 03 49 51
53 92 1F 17 6C F5 9D 8B D8 81 71 CA FA 02 45 3A 21 B0 36 5F 69 0A 3B
7C 0D 61 96 B4 1E 88 09 45 D2 F2 9D 25 F3 24 F2 E1 33 7A 6B 29 BA 18
D4 11 24 BC CE 3C B1 CD F5 36 D2 75 71 29 57 F5 8A D9 41 E1 00 E3 8C
A8 A9 57 41 E3 B1 78 57 61 69 C4 C1 E1 EB DB 8E BE B6 31 2F 62 E2 63
EB EE 31 5F 91 8A D8 45 00 45 FE 1B 12 27 1E BD 54 35 BA 17 E9 20 C8
CC 7D 45 09 14 8C 78 90 82 28 F2 D5 48 E5 21 E4 FE 0C 7B 42 66 9B F9
DA 58 71 2C B3 CE 8A 91 17 ED 52 8C 6E 4D 74 E1 3E 68 E4 4F D3 1E 4B
2E 21 28**

Success:

90 00

3.15 SIGN_DATA_SHORT

3.15.1 Description

Short version of SIGN_DATA for devices that do not support the extended APDU format. The response does not contain the actual signed data but its length. The SDC may read the signed data by using the required number of READ_SIGNED_DATA commands.

3.15.2 Fields

Command type = 4S.

COMMAND

CLA	INS	P1	P2	LC	Data
0xB0	0x34	0x00	0x00	Variable	RAW_DATA||NONCE

Table 3.42: SIGN_DATA_SHORT Command

RESPONSE

Data	SW1	SW2
Slength(SIGNED_DATA)	0x90	0x00

Table 3.43: SIGN_DATA_SHORT Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x06	SW_UNAUTHORIZED	Incorrect nonce
0x9C	0x	SW_SIGN_DATA_FORBIDDEN	Another reading process is ongoing

Table 3.44: SIGN_DATA_SHORT Errors

3.15.3 Formal description

SIGN_DATA_SHORT = {

COMMAND = {

CLA PIN_CLA,

INS SIGN_DATA_SHORT,

0x00,

0x00,

Blength(RAW_DATA||NONCE),

RAW_DATA,

NONCE

},

RESPONSE = {

Slength(SIGNED_DATA),

SUCCESS

},

ERRORS {

ISO7816_EXCEPTIONS,

DATA_SIGNING_EXCEPTIONS

}

}

3.15.4 Example

In this example the piece of data intended to sign is "000102030405060708090A0B0C0D0E0F"

COMMAND : SIGN_DATA_SHORT

CLA:

B0

INS:

34

P1/P2:

00 00

APDU length:

18

Data:

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

Nonce:

C9 DE 69 B3 56 A7 F7 FF

RESPONSE 1:

Signed Data Length:

01 00

Success:

90 00

3.16 READ_SIGNED_DATA

3.16.1 Description

This command has to be sent a number of times until the full signed data resulting from a SIGN_DATA_SHORT command is retrieved. The SDC is responsible of asking for the right piece of data at each request. It also has to indicate to the SAM when it is asking for the last piece of signed data. No other SIGN_DATA_SHORT or SIGN_DATA is allowed while there is some piece of signed data unread. In that case, the command is rejected with the exception SW_READ_SIGNED_DATA_FORBIDDEN. No other SIGN_DATA or SIGN_DATA_SHORT is allowed while there is some piece of signed data not yet read. In that case, the command is rejected with the exception SW_SIGN_DATA_FORBIDDEN.

3.16.2 Fields

Command type = 4S.

COMMAND

CLA	INS	LAST_CHUNK	CHUNK_SIZE	LC	Data
0xB0	0x80	True: 0x01 False: 0x00	Variable	0x0A	SIGNED_DATA_OFFSET|| NONCE

Table 3.45: READ_SIGNED_DATA Command

RESPONSE

Data	SW1	SW2
SIGNED_DATA_CHUNK	0x90	0x00

Table 3.46: READ_SIGNED_DATA Response

ERRORS

SW1	SW2	Name	Info
0x9C	0x06	SW_UNAUTHORIZED	Incorrect nonce
0x9C	0x62	SW_AUDIT_READ_FORBIDDEN	No signed data in place.

Table 3.47: READ_SIGNED_DATA Errors

3.16.3 Formal description

READ_SIGNED_DATA = {

COMMAND = {

CLA PIN_CLA,

INS READ_SIGNED_DATA,

LAST_CHUNK,

CHUNK_SIZE,

Blength(SIGNED_DATA_OFFSET||NONCE),

SIGNED_DATA_OFFSET,

NONCE

},

RESPONSE = {

SIGNED_DATA_CHUNK,

SUCCESS

},

ERRORS {

ISO7816_EXCEPTIONS,

AUDIT_EXCEPTIONS

}

}

3.16.4 Example

In this example the piece of data intended to sign is “000102030405060708090A0B0C0D0E0F“

COMMAND 1: SIGN_DATA_SHORT

CLA:

B0

INS:

34

P1/P2:

00 00

Apdu length:

18

Data:

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

Nonce:

C9 DE 69 B3 56 A7 F7 FF

RESPONSE 1:

Signed Data Length:
01 00

Success:

90 00

COMMAND 2: READ_SIGNED_DATA (Last Chunk = false, offset = 0)

CLA:
B0

INS:

35

Last Chunk = False:

00

Requested Length:

FA

APDU Length:

0A

Offset:

00 00

Nonce:

C9 DE 69 B3 56 A7 F7 FF

RESPONSE 2:

Signed Data Chunk:

20 68 B4 CB BB 2B F7 EF E1 95 F2 DA 36 A5 7B 6D B0 69 03 C6 E9 27 A3
91 B4 1C DC 59 93 8B 29 D6 C3 D7 E4 2D F5 D4 F4 16 3C 2E EE 5D 4D EE
CF F5 64 CF B5 3C E0 4C B6 B5 D3 3E D8 11 57 9D A8 D3 B4 BB 6A B5 C7
48 86 4B BF D9 35 9B 85 5C 8C 06 B8 B9 1D 4D 0F 10 F9 59 43 1A 91 F8
2C 51 8A 59 38 80 5E 83 21 D3 84 7A 96 5A EE DB 36 21 7B 2F DA 7C 83
D2 76 F4 F8 14 74 98 65 73 10 68 0B 46 43 DC 0C 75 77 E2 F4 AC 05 66
73 70 4F F3 9F 14 A1 B8 83 D6 B1 5A B9 7F C1 4D 85 55 2C E9 ED 2C C5
40 65 FD AE 32 F2 7F 34 8B 12 E0 E7 69 00 99 C0 31 B6 A6 55 E7 80 12
6C E6 F2 2E 8C 88 87 52 C6 93 EA 45 41 30 28 68 1D BC 5A 57 C4 35 0A
92 3C E7 BD 7F 38 58 22 14 C7 29 29 13 8E 11 4F 0A 10 2B 53 82 50 FC
F5 4A CA 0D 7C 57 8B B6 EF 0C 20 38 D9 ED C2 31 07 50 7B 2C

Success:

90 00

COMMAND 3:

CLA
B0

INS:

35

Last Chunk = True

01

Requested Length:

06

APDU Length:

0A

Offset:

00 FA

Nonce:

C9 DE 69 B3 56 A7 F7 FF

RESPONSE 3

Signed Data Chunk:
99 97 76 B1 16 5A

Success:

90 00

4 TYPE DEFINITIONS

4.1 Constants

These are some constant predefined values.

4.1.1 MAX

MAX = 1024

4.1.2 SELECT_BY_NAME

SELECT_BY_NAME = 0x04

4.1.3 AID

Avatar Applet ID.

AID = 0x417661746172000000

4.1.4 SHA_SIZE

Size of a sha1 hash.

SHA_SIZE = 0x14

4.1.5 SHA256_SIZE

Size of a sha256 hash.

SHA256_SIZE = 0x20

4.1.6 NONCE_SIZE

Size of the nonce that the SAM replies after a successful login.

NONCE_SIZE = 8.

4.1.7 MAX_NUM_KEYS

MAX_NUM_KEYS = 32

4.1.8 SALE

A transaction of type sale.

SALE = 0x00

4.1.9 REFUND

A transaction of type refund.

REFUND = 0x01

4.1.10 CM_RSA_NOPAD

No padding.

CM_RSA_NOPAD = 0x00

4.1.11 CM_RSA_PAD_PKCS1

Padding of data according to PKCS1.

CM_RSA_PAD_PKCS1 = 0x01

4.1.12 NORMAL

Indicates a regular transaction.

NORMAL = 0x00

4.1.13 TRAINING

Indicates a training transaction.

TRAINING = 0x01

4.1.14 PROFORMA

Indicates a proforma transaction.

PROFORMA = 0x02

4.1.15 DL_APDU

Input data for a transaction is contained in the APDU

DL_APDU = 0x01

4.1.16 DL_IOBUF

Input data for a transaction is contained in the IO Buffer of the applet.

DL_IOBUF = 0x02

4.1.17 ENCRYPT

A cryptographic encryption is issued.

ENCRYPT = 0x03

4.1.18 DECRYPT

A cryptographic decryption is issued.

DECRYPT = 0x04

4.1.19 SIGN

A cryptographic signing is issued.

SIGN = 0x05

4.1.20 SUCCESS

An APDU COMMAND is performed without errors.

SUCCESS = 0x9000

4.1.21 IDENTITY_NUMBER

In the future, the applet might work for several identities simultaneously. As of now, it only support identity, therefore we define it as a constant.

IDENTITY_NUMBER = 0x00

4.1.22 CIPHER_INIT

CIPHER_INIT = 0x01

4.1.23 CIPHER_PROCESS

CIPHER_PROCESS = 0x02

4.1.24 CIPHER_FINAL

CIPHER_FINAL = 0x03

4.1.25 CIPHER_ONE_STEP

CIPHER_ONE_STEP = 0x04

4.1.26 ICOUNTER_SIZE

This is the fixed length of an integer counter.

ICOUNTER_SIZE = 6

4.1.27 GLOBAL_COUNTERS_SIZE

This is the fixed length of the GLOBALCOUNTERS field in the response of a signing COMMAND.

GLOBAL_COUNTERS_SIZE = (Short)**2x**ICOUNTER_SIZE

4.1.28 COUNTERS_HASH_SIZE

This is the size of the hash of the internal counters object hash. Currently the hash function used is sha1.

COUNTERS_HASH_SIZE = SHA_SIZE

4.1.29 SEED_SIZE

Field contained in the GEN_AUDIT_TOKEN command indicating the size of the seed that the SAM will use for generating a random audit token.

SEED_SIZE = 0x10

4.1.30 TOKEN_SIZE

Size of the token that the SDC command to the SAM.

TOKEN_SIZE = 0x20

4.1.31 IDENTITY_SIZE

Size of the identity field stored in the SAM.

IDENTITY_SIZE = SHA256_SIZE

4.1.32 MAX_UNAUDITED

Maximum number of unaudited transactions in the SAM.

MAX_UNAUDITED = 1000

MAX	Int	1024
SHA_SIZE	Byte	0x14
SHA256_SIZE	Byte	0x20
NONCE_SIZE	Integer	8
AID	Byte[14]	0x417661746172000000
SALE	Byte	0x00
REFUND	Byte	0x01
CM_RSA_NOPAD	Byte	0x00
CM_RSA_PAD_PKCS1	Byte	0x01
NORMAL	Byte	0x00
TRAINING	Byte	0x01
PROFORMA	Byte	0x02
DL_APDU	Byte	0x01
DL_IOBUF	Byte	0x02
ENCRYPT	Byte	0x03
DECRYPT	Byte	0x04
SIGN	Byte	0x05
SUCCESS	Byte[2]	0x9000
CIPHER_ONE_STEP	Byte	0x04
ICOUNTER_SIZE	Int	6
GLOBAL_COUNTERS_SIZE	Int	2xICOUNTER_SIZE
COUNTERS_HASH_SIZE	Int	SHA_SIZE
SEED_SIZE	Byte	0x10
TOKEN_SIZE	Byte	0x20
IDENTITY_SIZE	Byte	SHA256_SIZE

Table 4.1: CONSTANTS

4.2 Primitive Data Types

4.2.1 Byte

4.2.2 Byte[]

Array of Bytes of undefined length. The length might go from 0 to MAX .

4.2.3 Byte[N]

Array of Bytes of length N.

4.2.4 Short

The short data type is a 16-bit signed two's complement integer. The Bytes are ordered from most significant to least significant.

4.2.5 Integer

Data type whose value is either 0 or a positive Integer from 1-255 or 0 .

4.2.6 Int

Integer number.

4.2.7 Alphanumeric

Byte Alphanumeric = [0..9|a..z|A..Z].

4.3 Variable Data Types

4.3.1 CLA

This field represents the type of command sent to the SAM. These are the valid values.

- PIN_CLA: This type of command indicates that the user has to be logged in.
- SECURE_CLA: These commands require the existence of a Secure Channel between the SDC and the SAM. They are encrypted and Mac’ed.

- SELECT_CLA: Only used for selecting a particular AID.

Name	Value
PIN_CLA	0xB0
SECURE_CLA	0x84
SELECT_CLA	0x00

Table 4.2: CLA

4.3.2 INS

This field in an APDU COMMAND represents the instruction code. It can have one of these values.

COMMAND	CODE
SELECT_AVATAR_APPLET	0xA4
VERIFY_PIN	0x42
SIGN_INVOICE	0x38
SIGN_INVOICE_SHORT	0x38
SIGN_INVOICE_T	0x39
SIGN_INVOICE_T_SHORT	0x39
READ_SIGNATURE	0x55
START_AUDIT	0x78
START_AUDIT_SHORT	0x80
READ_AUDIT	0x81
VERIFY_AUDIT	0x79
VERIFY_AUDIT_SHORT	0x82
GET_COUNTERS	0x76
SIGN_DATA	0x33
SIGN_DATA_SHORT	0x34
READ_SIGNED_DATA	0x35

Table 4.3: INSTRUCTION CODES

4.3.3 PIN

An array of 6 ASCII Bytes.

Byte[6] PIN = Ascii[Byte[6] Alphanumeric]

1.1.1 RSA_KEY_SIZE

It is common practice to express the length of RSA keys in bits. The current supported values are 1024 bits or 2048 bits. For consistency with the rest of the values in this document, this value is expressed in Bytes.

Byte[2] RSA_KEY_SIZE = CHOICE {

0x0080,

0x0100

}

1.1.2 KEY_NUMBER

The SAM can have several RSA keys. The maximum number of keys is MAX_NUM_KEYS . The keys having an odd index are the private keys, and the ones having an even index are the public ones. Private key with index 0 is tied to public key with index 1; private key 2 with public key 3, and so on.

The SDC won't be able to generate new keys. Currently, the SAM's are shipped with a single key pair with indexes 0 and 1.

1.1.3 OPERATION_TYPE

Defined the type of cryptographic operation issued. It can have these values:

- CIPHER_INIT : Initializes cipher
- CIPHER_PROCESS : Processes more data
- CIPHER_FINAL : Processes last chunk of data.
- CIPHER_ONE_STEP : Same as Initialize and Final in one step.

Name	Value
CIPHER_INIT	CIPHER_INIT
CIPHER_PROCESS	CIPHER_PROCESS
CIPHER_FINAL	CIPHER_FINAL
CIPHER_ONE_STEP	CIPHER_ONE_STEP

Table 4.4: OPERATION_TYPE

1.1.4 CIPHER_MODE

This field defines the type of padding to be applied to the data on which a cryptographic operation is going to be applied.

Name	Value
CM_RSA_NO_PAD	CM_RSA_NOPAD
CM_RSA_PAD_PKCS1	CM_RSA_PAD_PKCS1

Table 4.5: CIPHER_MODE

1.1.5 OPERATION

Cryptographic operation issued. It can have these values:

- ENCRYPT: Encryption of some input data using a specific key.
- DECRYPT: Decryption of some input data using a specific key.
- SIGN: Signature of some input data using one of the SAM private keys.

Name	Value
ENCRYPT	ENCRYPT
DECRYPT	DECRYPT
SIGN	SIGN

Table 4.6: OPERATION

1.1.6 DATA_LOCATION

This field indicates the location of the input data to the cryptographic operation issued. It can have two values:

- DL_APDU: The input data is contained in the same APDU of the COMMAND.
- DL_IOBUF: The input data is the in I/O buffer of the SAM.

Name	Value
DL_APDU	DL_APDU
DL_IOBUF	DL_IOBUF

Table 4.7: DATA_LOCATION

1.1.7 TRANSACTION_TYPE

This type defines the type of transaction to be signed by the SAM. It can have these values:

Name	Value
SALE	SALE
REFUND	REFUND

Table 4.8: TRANSACTION_TYPE

1.1.8 TRANSACTION_MODE

This type defines the mode of the transaction to be signed. It can have three different values:

- NORMAL. Regular operation
- TRAINING. Test signature.
- PROFORMA.

Name	Value
NORMAL	NORMAL
TRAINING	TRAINING
PROFORMA	PROFORMA

Table 4.9: TRANSACTION_MODE

1.1.9 INTEGER_COUNTER

These counters lack a decimal part. They have a fix length ICOUNTER_SIZE, and they encode the Integer part in BCD format.

Example:

C1 = [0x10, 0x46, 0x00, 0x01, 0x90, 0x50] represents the decimal value 104600019050.

As a consequence, the maximum value that an Integer counter can store is 999999999999, which should be enough for the card lifetime.

INTEGER_COUNTER = Byte[ICOUNTER_SIZE]

1.1.10 DECIMAL_COUNTER

These are counters having both an Integer part and a decimal part. Each part uses the same encoding as an Integer Counter, but with variable length. The length is explicitly prepended to the value. The actual format is:

Byte[] DECIMAL_COUNTER = {

TotalLength	**Integer,**
IntegerPartLength	**Integer,**
IntegerPart	**Byte[IntegerPartLength],**
DecimalPartLength	**Integer,**
DecimalPart	**Byte[DecimalPartLength]**

}

1.1.11 TRANSACTION_AMOUNT

In a transaction signing COMMAND, this field represents the amount of money of the transaction. It is expressed as a DECIMAL_COUNTER .

DECIMAL_COUNTER **TRANSACTION_AMOUNT**

1.1.12 TAX_AMOUNT

In a transaction signing COMMAND, this field represents the amount of taxes applied to the transaction. It is expressed as a DECIMAL_COUNTER .

DECIMAL_COUNTER **TAX_AMOUNT**

1.1.13 TRANSACTION_VALUES

This fields represents the whole amount of the transaction, concatenating the TRANSACTION_AMOUNT and the TAX_AMOUNT.

Byte[] TRANSACTION_VALUES = {

(Byte)(Blength(TRANSACTION_AMOUNT) + Blength(TAX_AMOUNT)),

TRANSACTION_AMOUNT,

TAX_AMOUNT

}

1.1.14 INVOICE

From the point of view of the applet, the invoice is just an array of Bytes, and it's the actual piece of data that has to be signed using the private key present in the SAM.

Byte[] INVOICE

1.1.15 REQUEST

This is the full data that the SDC presents to the card for:

1) Signing

2) Updating the internal counters.

It is comprised of several fields.

Byte[] REQUEST =

{

TRANSACTION_TYPE,

TRANSACTION_MODE,

TRANSACTION_VALUES,

INVOICE

}

1.1.16 NONCE

The nonce is an eight-Byte random value that the card replies after a successful login via the VERIFY_PIN command. This values has to be appended to all the subsequent commands sent to the SAM.

Byte[8] NONCE

1.1.17 TOTAL_INVOICE_COUNTER

This is the internal counter that stores the total number of invoices signed, independently of the transactions types and modes. It is represented as an INTEGER_COUNTER.

INTEGER_COUNTER **TOTAL_INVOICE_COUNTER**

1.1.18 NORMAL_TRANSACTIONS_COUNTER

Internal counter that stores the total number of transactions signed whose mode was NORMAL. It is represented as an INTEGER_COUNTER.

INTEGER_COUNTER **NORMAL_TRANSACTIONS_COUNTER**

1.1.19 TRAINING_TRANSACTIONS_COUNTER

Internal counter that stores the total number of transactions signed whose mode was TRAINING. It is represented as an INTEGER_COUNTER.

INTEGER_COUNTER **TRAINING_TRANSACTIONS_COUNTER**

1.1.20 PROFORMA_TRANSACTIONS_COUNTER

Internal counter that stores the total number of transactions signed whose mode was PROFORMA. It is represented as an INTEGER_COUNTER.

INTEGER_COUNTER **PROFORMA_TRANSACTIONS_COUNTER**

1.1.21 TRANSACTIONS_COUNTER

This field refers to different internal counters according to the mode of the transaction signed. Formally expressed:

INTEGER_COUNTER T**RANSACTIONS_COUNTER = {**

SWITCH(TRANSACTION_MODE**) {**

CASE NORMAL **:**

NORMAL_TRANSACTIONS_COUNTER;

CASE TRAINING **:**

TRAINING_TRANSACTIONS_COUNTER;

CASE PROFORMA **:**

PROFORMA_TRANSACTIONS_COUNTER

}

}

1.1.22 GLOBAL_COUNTERS

This is field is composed by concatenating two counters:

1) TOTAL_INVOICE_COUNTER

2) TRANSACTIONS_COUNTER

Byte[2*ICOUNTER_SIZE] GLOBAL_COUNTERS = {

TOTAL_INVOICE_COUNTER,

TRANSACTIONS_COUNTER

}

1.1.23 LGLOBAL_COUNTERS

Byte[2*ICOUNTER_SIZE + 2] LGLOBAL_COUNTERS = {

GLOBAL_COUNTERS_SIZE,

GLOBAL_COUNTERS

}

1.1.24 SALES_COUNTER

Number of valid sale transactions signed. It can't decrease.

INTEGER_COUNTER **SALES_COUNTER**

1.1.25 REFUNDS_COUNTER

Number of valid refund transactions signed. It can't decrease.

INTEGER_COUNTER **REFUNDS_COUNTER**

1.1.26 SALES_VALUE_COUNTER

Sum of all the amounts in valid sale transactions. It is decreased after a valid refund in the refund amount signed.

DECIMAL_COUNTER **SALES_VALUE_COUNTER**

1.1.27 REFUNDS_VALUE_COUNTER

Sum of all the amounts in valid refund transactions. It cant decrease.

DECIMAL_COUNTER **REFUNDS_VALUE_COUNTER**

1.1.28 SALES_TAX_VALUE_COUNTER

Sum of all the tax amounts in valid sale transactions. It is decreased after a valid refund in the refund tax amount signed.

DECIMAL_COUNTER **SALES_TAX_VALUE_COUNTER**

1.1.29 REFUNDS_TAX_VALUE_COUNTER

Sum of all the tax amounts in valid refund transactions. It can't decrease.

DECIMAL_COUNTER **REFUNDS_TAX_COUNTER**

1.1.30 LAST_AUDITED_TRANSACTION_COUNTER

Id of the highest transaction Id audited so far. It can't decrease.

INTEGER_COUNTER **REFUNDS_COUNTER**

1.1.31 CURRENTLY_AUDITED_TRANSACTION_COUNTER

If a proof of audit verification is pending, this counter stored the highest transaction ID for which an audit token was generated. Otherwise its value is 0.

INTEGER_COUNTER **REFUNDS_COUNTER**

1.1.32 INTERNAL_COUNTERS

This is an object composed by a group of counters. Its definition is:

Byte[] InternalCounters = {

NORMAL_TRANSACTIONS_COUNTER,

SALES_COUNTER,

REFUNDS_COUNTER,

LAST_AUDITED_TRANSACTION_COUNTER,

CURRENTLY_AUDITED_TRANSACTION_COUNTER,

SALES_VALUE_COUNTER,

REFUNDS_VALUE_COUNTER,

SALES_TAX_VALUE_COUNTER,

REFUNDS_TAX_VALUE_COUNTER

}

1.1.33 INTEGER_COUNTERS_LENGTH

This is the combined length of all the internal counters of type INTEGER_COUNTER, expressed in one byte.

1.1.34 DECIMAL_COUNTERS_LENGTH

This is the combined length of all the internal counters of type DECIMAL_COUNTER, expressed in one byte.

1.1.35 COUNTERS

Avatar applet builds this object and appends it to the signature; but not in plain. It is encoded with ATAX public encryption key. Therefore this object is never seen directly at the SDC. It has to be decrypted and interpreted in ATAX. The format of this object is the following:

Byte[] COUNTERS = {

INTEGER_COUNTERS_LENGTH,

TOTAL_INVOICE_COUNTER,

NORMAL_TRANSACTIONS_COUNTER,

SALES_COUNTER,

REFUNDS_COUNTER,

LAST_AUDITED_TRANSACTION_COUNTER,

CURRENTLY_AUDITED_TRANSACTION_COUNTER,

DECIMAL_COUNTERS_LENGTH,

SALES_VALUE_COUNTER,

REFUNDS_VALUE_COUNTER,

SALES_TAX_VALUE_COUNTER,

REFUNDS_TAX_VALUE_COUNTER

}

1.1.36 COUNTER_HASH_SIZE

Size of the hash of the COUNTERS object, expressed in two Bytes.

COUNTERS_HASH_SIZE = (Short)SHA1_SIZE

1.1.37 COUNTERS_HASH

This field contains the sha1 hash of the COUNTERS object.

Byte[SHA1_SIZE] COUNTERS_HASH = sha1(COUNTERS)

1.1.38 HCOUNTERS

Byte[SHA1_SIZE + 2] HCOUNTERS = {

COUNTER_HASH_SIZE,

COUNTER_HASH

}

1.1.39 SIMPLE_SIGNATURE

This is the signature of the INVOICE object obtained by using algorithm ALG_RSA_SHA_PKCS1. First the algorithm generates a 20-Byte SHA1 digest of the data, pads the digest according to the PKCS#1 (v1.5) scheme, and finally encrypts it using the RSA private key present in the SAM. The size of the signature is the same as the length of the key used.

1.1.40 LSIMPLE_SIGNATURE

Byte[RSA_KEY_SIZE + 2**] LSIMPLE_SIGNATURE = {**

RSA_KEY_SIZE,

SIMPLE_SIGNATURE

}

1.1.41 SIGNATURE

This is the signature of the INVOICE object concatenated with the COUNTERS_HASH object using algorithm ALG_RSA_SHA_PKCS1. First the algorithm generates a 20-Byte SHA1 digest of the data, pads the digest according to the PKCS#1 (v1.5) scheme, and finally encrypts it using the RSA private key present in the SAM. The size of the signature depends on the length of the keys.

Byte[] AUX = {

INVOICE,

COUNTERS_HASH

}

Byte[RSA_KEY_SIZE] SIGNATURE = ALG_RSA_SHA_PKCS1(AUX)

1.1.42 LSIGNATURE

Byte[RSA_KEY_SIZE + 2**] LSIGNATURE = {**

RSA_KEY_SIZE,

SIGNATURE

}

1.1.43 ENCRYPTED_COUNTERS_SIZE

This is the length of the ENCRYPTED_COUNTERS object, expressed in two Bytes.

Short ENCRYPTED_COUNTERS_SIZE

1.1.44 ENCRYPTED_COUNTERS

This object is created by encrypting the COUNTERS object using ATAX public key.

Byte[RSA_KEY_SIZE] ENCRYPTED_COUNTERS = $Enc_{KATAXPub}$(COUNTERS)

1.1.45 ECOUNTERS

Byte[RSA_KEY_SIZE + 2] ECOUNTERS = {

ENCRYPTED_COUNTERS_SIZE,

ENCRYPTED_COUNTERS

}

1.1.46 FULL_SIGNATURE

FULL_SIGNATURE = {

LGLOBAL_COUNTERS,

HCOUNTERS,

LSIGNATURE,

ECOUNTERS

}

1.1.47 LAST_CHUNK

Field that indicates whether or nor the SDC is requesting to read the last piece of a piece of data.

0x00 : Not last chunk

0x01 : Last chunk

1.1.48 CHUNK_SIZE

Field indicating the number of Bytes requested from a piece of data.

Integer CHUNK_SIZE

1.1.49 SIGNATURE_OFFSET

Field present in the READ_SIGNATURE command that indicates the initial Byte of the slice of the signature that the SDC wants to read. The offset starts at 0 and is incremented in CHUNK_SIZE Bytes until all the Bytes if the signature are read.

Short SIGNATURE_OFFSET

1.1.50 SIGNATURE_CHUNK

Field present in the READ_SIGNATURE response that contains the piece of the signature object issued.

Byte[] SIGNATURE_CHUNK

1.1.51 NOW

Now is the current timestamp expressed in unix time (**milliseconds** `since 1970-01-01 00:00:00 UTC).`

1.1.52 SEED

The seed is sent in the GEN_AUDIT_TOKEN command, and it is used internally by the SAM as the seed for the generating a random sequence of Bytes of size TOKEN_SIZE. The formula for obtaining this value is:

Byte[SEED_SIZE] SEED =

{

Ascii(Split(Now)),

0x000000

}

1.1.53 CURRENTLY_AUDITED_TID

This field in sent in the GEN_AUDIT_TOKEN command, and it indicates the number of transaction that the backbone wants to audit. That is, this transaction and all the unaudited previous ones are intended to be audited.

INTEGER_COUNTER CURRENTLY_AUDITED_TID

1.1.54 CertID

Serial Number of the X509 certificate present in the SAM.

Byte[4] CertID

1.1.55 AUDIT_DATA

This is the object built by the SAM when a PoA process is initiated by an external agent.

Byte[] = {

SLENGTH(CertID) ,

CertID ,

SLENGTH(TID),

TID,

SLENGTH(LAST_AUDITED_TID),

LAST_AUDITED_TID,

SLENGTH(COUNTERS),

COUNTERS,

SLENGTH(TOKEN),

TOKEN

}

1.1.56 EAUDIT_DATA

Encrypted AUDIT_DATA with the public key of ATAX.

Byte[RSA_KEY_SIZE] = $Enc_{KATAXPub}$(AUDIT_DATA)

1.1.57 SAUDIT_DATA

Signed AUDIT_DATA with the private RSA key present in the SAM. The signing algorithm is **RSA_SHA_PKCS1.**

Byte[RSA_KEY_SIZE] = $Sign_{SAMPrivate}$(AUDIT_DATA)

1.1.58 TOKEN

A token is a random array of TOKEN_SIZE length.

Byte[TOKEN_SIZE] TOKEN

1.1.59 AUDIT_DATA_RESPONSE

This is the response from the SAM to the START_AUDIT command.

Byte[] AUDIT_DATA_RESPONSE = {

SLENGTH(EAUDIT_DATA),

EAUDIT_DATA,

SLENGTH(SAUDIT_DATA),

SAUDIT_DATA

}

1.1.60 SHORT_AUDIT_DATA_RESPONSE

This is the response from the SAM to the START_AUDIT_SHORT command.

Byte[] AUX = {

SLENGTH(EAUDIT_DATA),

EAUDIT_DATA,

SLENGTH(SAUDIT_DATA),

SAUDIT_DATA

}

Byte[] AUDIT_DATA_RESPONSE = {

SLENGTH(AUX)

}

1.1.61 AUDIT_DATA_OFFSET

In the READ_AUDIT command, this parameter represents the offset of the piece of AUDIT_DATA_RESPONSE requested by the SDC.

1.1.62 AUDIT_DATA_CHUNK

Field present in the READ_AUDIT response that contains the piece of the AUDIT_DATA_RESPONSE object issued.

Byte[] AUDIT_DATA_CHUNK

1.1.63 STOKEN

TOKEN signed by ATAX, using algorithm RSA_SHA_PKCS1.

Byte[RSA_KEY_SIZE**] STOKEN = Sign$_{KATAXPRIVATE}$(**TOKEN)

1.1.64 IDENTITY

A SAM has an identity given to it by the Avatar backbone during the enrollment stage. The identity is not stored itself, but its sha256 hash. The filed is an ASN1 encoded string by formed by the concatenations of these fields, also encoded in ASN1:

- userid: the user to to which the SAM is delivered.
- serial: the serial number of the card
- certid: the serial number of the certificate stored in the card.
- Issuer: “Avatar Inc.”
- Creation Date: The timestamp at the enrollment stage.

Byte[SHA256_SIZE] IDENTITY

1.1.65 CHALLENGE

This field in sent in the VERIFY_AUDIT_PROOF command. The challenge is obtained by encrypting with ATAX private key an array of Bytes obtained after concatenating the following fields:

PLAIN_CHALLENGE = {

TOKEN,

IDENTITY,

CURRENTLY_AUDITED_TID

}

Byte[RSA_KEY_SIZE] CHALLENGE = $Enc_{KATAXPri}$(PLAIN_CHALLENGE)

1.1.66 LAST_TOKEN_CHUNK

In VERIFY_AUDIT_SHORT, this parameter indicates whether or not this chunk of the STOKEN is the last one.

Byte[1] LAST_TOKEN_CHUNK := CHOICE {

0x00 False,

0x01 True

}

1.1.67 TOKEN_CHUNK_SIZE

In VERIFY_AUDIT_SHORT, this parameter indicates the length of the STOKEN fragment that is carried as data.

Byte[1] = TOKEN_CHUNK_SIZE

1.1.68 STOKEN_CHUNK

Fragment of STOKEN

Byte[] STOKEN_CHUNK

1.1.69 RAW_DATA

In SIGN_DATA, it is the piece of data to be signed.

Byte[] RAW_DATA

1.1.70 SIGNED_DATA

Present in SIGN_DATA, it is the signature of RAW_DATA.

Byte[RSA_KEY_SIZE] RAW_DATA

1.1.71 SIGNED_DATA_OFFSET

In the READ_SIGNED_DATA command, this parameter represents the offset of the piece of SIGNED_DATA requested by the SDC.

1.1.72 SIGNED_DATA_CHUNK

Fragment of SIGNED_DATA

Byte[] SIGNED_DATA_CHUNK

1.2 Exceptions

All the exceptions generated within the card or the applet are conveyed to the SDC with a Status Code different from 0x9000 in response to an APDU COMMAND. The exceptions are divided into different groups.

1.2.1 ISO7816_EXCEPTIONS

See:
http://www.win.tue.nl/pinpasjc/docs/apis/jc222/javacard/framework/ISO7816.html

1.2.2 CRYPTOGRAPHIC_EXCEPTIONS

Exceptions related to cryptographic operations. These are the possible values.

Name	Value
SW_CRYPTO_ILLEGAL_VALUE	0x9C30
SW_CRYPTO_ILLEGAL_USE	0x9C31
SW_CRYPTO_INVALID_INIT	0x9C32
SW_CRYPTO_NO_SUCH_ALGORTIHM	0x9C33
SW_CRYPTO_UNINITIALIZED_KEY	0x9C34
SW_CRYPTO_INCONSISTENT_KEY_ALG_PAIR	0x9C35
SW_CRYPTO_UNDEFINED	0x9C36
SW_DIRECTION_UNSUPPORTED	0x9C13
SW_LOCATION_INVALID	0x9C14
SW_KEY_SIZE_UNSUPPORTED	0x9C16
SW_KEY_TYPE_UNSUPPORTED	0x9C17
SW_KEY_TYPE_INVALID	0x9C18
SW_INVALID_KEY_ID	0x9C19
SW_CIPH_MODE_INVALID	0x9C1C
SW_INCONSTANT_KEYPAIRING	0x9C1E
SW_CRYPTO_OPERATION_UNSUPPORTED	0x9C27

Table 4.10: CRYPTOGRAPHIC_EXCEPTIONS

1.2.3 AUDIT_EXCEPTIONS

These is a group of exceptions that might arise when asking for a new audit token or in the verification stage.

Name	Value
SW_START_AUDIT_FORBIDDEN	0x9C61
SW_AUDIT_READ_FORBIDDEN	0x9C62
SW_WRONG_PROOF	0x9C64
SW_AUDIT_FORBIDDEN	0x9C66

Table 4.11: AUDIT_EXCEPTIONS

1.2.4 TRANSACTION_EXCEPTIONS

Some of the operations in the applet run inside a transaction in order to preserve the integrity of the data inside the card. This group of exceptions are related to these transactions.

Name	Value
SW_TRANSACTION_IN_PROGRESS	0x9C41
SW_TRANSACTION_NOT_IN_PROGRESS	0x9C42
SW_TRANSACTION_PROBLEM	0x9C43
SW_TRANSACTION_UNDEFINED	0x9C44

Table 4.12: TRANSACTION_EXCEPTIONS

1.2.5 APPLET_EXCEPTIONS

Group of custom exceptions throw-able by the applet during its regular operation upon executing an APDU COMMAND.

Name	Value
SW_NO_MEMORY_LEFT	0x9C01
SW_AUTH_FAILED	0x9C02
SW_OPERATION_NOT_ALLOWED	0x9C03
SW_UNSUPPORTED_FEATURE	0x9C05
SW_UNAUTHORIZED	0x9C06
SW_OBJECT_NOT_FOUND	0x9C07
SW_OBJECT_EXISTS	0x9C08
SW_INVALID_PARAMETER	0x9C0F
SW_INCORRECT_P1	0x9C10
SW_INCORRECT_P2	0x9C11
SW_SEQUENCE_END	0x9C12

Table 4.13: APPLET_EXCEPTIONS

1.2.6 IDENTITY_EXCEPTIONS

Exception thrown when the maximum number of pin attempts has been reached.

Name	Value
SW_IDENTITY_BLOCKED	0x9C0C
SW_AUTH_FAILED	0x9C02

Table 4.14: IDENTITY_EXCEPTIONS

1.2.7 ARITHMETIC_EXCEPTIONS

Exceptions thrown when an arithmetic operations overflows.

Name	Value
SW_OVERFLOW	0x9C51

Table 4.15: ARITHMETIC_EXCEPTIONS

1.2.8 SIGNING_EXCEPTIONS

Exceptions thrown by the applet when trying to perform the signature of a piece of data.

Name	Value
SW_MAX_UNAUDITED_TRANSACTIONS	0x9C60
SW_INVALID_COMMAND_LENGTH	0x9C28
SW_INVALID_TRANSACTION_TYPE	0x9C29
SW_PENDING_SIGNATURE	0x9C45
SW_SIGNATURE_READ_FORBIDDEN	0x9C46

Table 4.16: SIGNING_EXCEPTIONS

1.2.9 DATA_SIGNING_EXCEPTIONS

Name	Value
SW_SIGN_DATA_FORBIDDEN	0x9C70
SW_READ_SIGNED_DATA_FORBIDDEN	0x9C71

Table 4.17: DATA_SIGNING_EXCEPTIONS

1.3 Operations

In this section, operations that can be applied to the data types defined in the previous sections are described.

1.3.1 Blength(Byte[] A)

Length of an array of Bytes expressed in a single Byte.

Integer Blength(Byte[] A) = (Byte)A.length

1.3.2 Slength(Byte[] A)

Length of an array of Bytes expressed in a short type.

Short Slength(Byte[] A) = (short)A.length

1.3.3 Elength(Byte[] A)

Extended length. It calculates Slength(A) and then Byte 0x00 is concatenated with the result.

```
Byte[3] ELength(A)  = {
        0x00,
        Slength(A)
}
```

1.3.4 Ascii(Byte[] A)

This operations takes an array of characters and returns an array of the same length with the ASCII value of each character.

Example: Ascii([1,2]) = [31,32]

1.3.5 Split(S)

This operations takes a string as input and return and array containing all the characters in the string.

Byte[S.length] = Split(S)

Example: Split(“1234”) = [1,2,3,4]