A-Trust Gesellschaft für Sicherheitssysteme
im elektronischen Datenverkehr GmbH
Landstraßer Hauptstraße 1b
A-1030 Wien

https://www.a-trust.at
E-Mail: office@a-trust.at

A-Trust cash register smart card
Developer Manual

Version: 0.3
Date: December 21, 2021

Contents

Datum	Rev	Autor	Änderungen
15.01.2016	0.1	Joel Chinnow	Initial release
06.04.2016	0.2	Joel Chinnow	Converted to LaTeX and added Card OS 5.3 smart card
14.12.2021	0.3	Joel Chinnow	Added Acos ID smart card

Table 1: Document history

1 Overview

This Document describes the APDU Interface for cash register smart cards offered by A-Trust.

The first section explains the required APDUs. The following two sections explain the details for the ACOS as well as the Card OS smart card.

Source code examples can be found on the A-Trust website: `https://labs.a-trust.at/developer/ShowSource.aspx?id=114`

2 Smart card commands

2.1 Smart card commands

2.1.1 SELECT

The SELECT command is used to select AIDs or FIDs on the smart card.

CLA	INS	P1	P2	LC	DATA
0x00	0xA4	0xSC	0xSO	0xLC	FID or AID

Table 2: SELECT command

2.1.1.1 SC

- For a SELECT with a FID set SC to 0x00
- For a SELECT with an AID set SC to 0x04

2.1.1.2 SO

The following values are allowed:

- 0x00 FCI optional in response
- 0x02 Select next
- 0x0C No File Control Information

2.1.1.3 LC

The length of FID or AID

2.1.1.4 Return codes

- 0x9000 Normal processing
- 0x6700 Wrong length
- 0x6A81 Function not supported; wrong parameters P1-P2
- 0x6A82 File not found
- 0x6D00 Instruction code not supported or invalid
- 0x6E00 Class not supported

2.1.2 MANAGE SECURITY ENVIRONMENT

The MANAGE SECURITY ENVIRONMENT command is used for ACOS smart cards to prepare a signature. Before sending the command, DF_SIG should be selected.

CLA	INS	P1	P2	LC	DATA
0x00	0x22	0x41	0xB6	0x06	0x840188800144

Table 3: MANAGE SECURITY ENVIRONMENT command

2.1.2.1 Return codes

- 0x9000 Normal processing
- 0x6700 Wrong length
- 0x6983 Key locked
- 0x6985 Conditions of use not satisfied (incorrect key type)
- 0x6A80 Incorrect parameter in data field
- 0x6A81 Function not supported; wrong parameters P1-P2
- 0x6A88 Key not found
- 0x6D00 Instruction code not supported or invalid
- 0x6E00 Class not supported
- 0x6F05 No security environment

2.1.3 VERIFY

The VERIFY command is used to authenticate the signature creation.

CLA	INS	P1	P2	LC	DATA
0x00	0x20	0x00	0x81	0x08	PIN

Table 4: VERIFY command

2.1.3.1 PIN

The format of the PIN depends on the smart card type. See the corresponding section of the concrete smart card.

2.1.3.2 Return codes

- 0x9000 Normal processing
- 0x63Cx Incorrect PIN value, x retries left
- 0x9615 Cleartext Pin not allowed
- 0x6700 Wrong length
- 0x6982 Security Status not satisfied
- 0x6983 PIN blocked
- 0x6985 Incorrect key type (no PIN)
- 0x6A81 Function not supported; wrong parameters P1-P2
- 0x6A00 KID not found
- 0x6B00 Wrong parameter P1-P2
- 0x6D00 Instruction code not supported or invalid
- 0x6E00 Class not supported

2.1.4 PUT HASH

The PUT HASH command is used to load the SHA256 hash into the smart card.

CLA	INS	P1	P2	LC	DATA
0x00	0x2A	0x90	0x81	0x20	DATA

Table 5: PUT HASH command

2.1.4.1 DATA

Data contains 32 bytes with SHA256 hash.

2.1.4.2 Return codes

- 0x9000 Normal processing
- 0x6700 Wrong length
- 0x6883 Command chain not finished

- 0x6982 Security status not satisfied
- 0x6A00 Function not supported; wrong parameters P1-P2
- 0x6A80 Wrong parameter in data field
- 0x6D00 Instruction code not supported or invalid
- 0x6E00 Class not supported
- 0x6F02 Data Overflow
- 0x6F05 No security environment

2.1.5 COMPUTE DIGITAL SIGNATURE

The COMPUTE DIGITAL SIGNATURE command is used to generate the signature after the hash is loaded on the smart card. If the command returns 0x9000, the

CLA	INS	P1	P2	LE
0x00	0x2A	0x9E	0x9A	0x00

Table 6: COMPUTE DIGITAL SIGNATURE command

computation was successful. Then, the answer contains the computed signature.

2.1.5.1 Return codes

- 0x9000 Normal processing
- 0x6700 Wrong length
- 0x6982 Security Status not satisfied
- 0x6983 Key locked
- 0x6985 Incorrect key type
- 0x6A00 Incorrect parameters P1-P2
- 0x6A80 Invalid hash length
- 0x6D00 Instruction code not supported or invalid
- 0x6E00 Class not supported
- 0x6F05 No security environment
- 0x6F03 Command sequence not correct

2.1.6 READ BINARY

The READ BINARY command is used to read data like certificates and CIN/CSN from the smart card. The data can be read either implicitly or explicitly.

CLA	**INS**	**P1**	**P2**	**LE**
0x00	0xB0	OH	OL	LE

Table 7: READ BINARY command

2.1.6.1 Explicit selection

For explicit selection the exact offset is given.

- OH Offset High Byte
- OL Offset Low Byte

2.1.6.2 Implicit selection

When using implicit selection, the data of the EF with the given Short Identifier (SID) is read. The SID always refers to an EF in the DF just selected before. Therefore, 3 bits of OH are used to indicate the implicit selection.

B8	B7	B6	B5	B4	B3	B2	B1
1	0	0	x	x	x	x	x

Table 8: OH for implicit selection

XXXXX is the Short File Identifier. The Lower Offset OL is always set to 0x00.

2.1.6.3 Le

Le (length expected), 0x00 all digits until the end of the file but max. 256 bytes.

2.1.6.4 Return codes

- 0x9000 Normal processing
- 0x6282 End of file/record reached before reading Le bytes
- 0x6700 Wrong length

- 0x6981 Command incompatible with file structure; incorrect file type
- 0x6982 Security status not satisfied
- 0x6984 Referenced data invalidated; file locked
- 0x6986 Command not allowed (no current EF)
- 0x6A81 Function not supported
- 0x6A86 Wrong parameters P1-P2 (Offset out of file)
- 0x6A00 File not found
- 0x6D00 Instruction code not supported or invalid
- 0x6E00 Class not supported

3 ACOS smart card

3.1 Workflow

For the different use cases, the following commands should be sent to the smart card in the given order. Communication with the smart card should use the T1 protocol. The commands are described in the following sections. For the commands, it is assumed that the MF is selected. If a different DF is selected, the MF has to be selected first. If the required DF is already selected, the select DF is not required.

Sign hash

1. SELECT DF_SIG
2. MANAGE SECURITY ENVIRONMENT
3. VERIFY
4. PUT HASH
5. COMPUTE DIGITAL SIGNATURE

Read certificate

1. SELECT DF_SIG
2. SELECT EF_C_CH_DS
3. READ BINARY

Read CIN / CSN

1. SELECT DF_DEC
2. SELECT EF_CIN_CSN
3. READ BINARY

3.2 Data structures

The smart card contains two dedicated files (DF), DF_SIG and DF_DEC.

3.2.1 DF_SIG

DF_SIG is used for making signatures and reading the certificate. The following application identifier (AID) or file id (FID) can be used to select it.

- AID = { 0xA0, 0x00, 0x00, 0x01, 0x18, 0x45, 0x43 }
- FID = { 0xDF, 0x70 }

DF_SIG contains the elementary file (EF) of the certificate.

- FID = { 0xc0, 0x02 }

3.2.2 DF_DEC

DF_DEC is used for reading the CIN / CSN.

- AID = { 0xA0, 0x00, 0x00, 0x01, 0x18, 0x45, 0x4E }
- FID = { 0xDF, 0x71 }

The EF of the CIN / CSN can be selected with the following values:

- FID = { 0xD0, 0x01 }
- SID = { 0x06 }

3.3 PIN format

The PIN consists of 8 bytes data. For each digit of the pin, the corresponding ASCII code (e.g. “1” = 0x31) should be used. The last bytes should be set to 0x00. As an example, the PIN "123456" would result in the following byte array: { 0x31 0x32 0x33 0x34 0x35 0x36 0x00 0x00}

3.4 Examples

3.4.1 Signature process

The first example shows the signature process.

```
byte[] DF_SIG = new byte[] { 0xDF, 0x70 };
byte[] TLV = new byte[] { 0x84, 0x01, 0x88, 0x80, 0x01, 0x44 };

sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, DF_SIG);
sendCase3APDU(0x00, 0x22, 0x41, 0xb6, TLV);
sendCase3APDU(0x00, 0x20, 0x00, 0x81, pin);
sendCase3APDU(0x00, 0x2A, 0x90, 0x81, hash);
Response r = sendCase2APDU(0x00, 0x2A, 0x9E, 0x9A, 0x00);
byte[] signature = r.GetData();
```

Listing 1: Signature process

After selecting the right DF, it is required to set the required parameter with MANAGE SECURITY ENVIRONMENT command. Now, an authentication via the VERIFY command with the PIN is necessary. Finally, the hash can be submitted with the PUT HASH command and the signature can be retrieved with the SIGN command.

If the card is used to make further signatures, it is not necessary to repeat these selections

```
sendCase3APDU(0x00, 0x20, 0x00, 0x81, pin);
sendCase3APDU(0x00, 0x2A, 0x90, 0x81, hash);
Response r = sendCase2APDU(0x00, 0x2A, 0x9E, 0x9A, 0x00);
byte[] signature = r.GetData();
```

Listing 2: Signature process without selection

This results in a faster signature calculation.

3.4.2 Certificate reading

The next example shows how the certificate can be read from the smart card.

```
byte[] DF_SIG = new byte[] { 0xDF, 0x70 };
byte[] EF_C_CH_DS = new byte[] { 0xc0, 0x02 };

sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, DF_SIG);
sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, EF_C_CH_DS);
int offset = 0;
byte[] certificate;
while (true) {
  Response r = sendCase2APDU(0x00, 0xB0, offset >> 8, offset);
  if (r.StatusWord != 0x9000) {
    break;
  }
    byte[] part = r.GetData();
    certificate += part;
  offset += 256;
}
```

Listing 3: Reading the certificate

After selecting the right DF / EF, the certificate can be read sequentially in 256 Byte Blocks.

3.4.3 Reading CIN / CSN

The last example shows how the CIN / CSN can be read from the smart card.

```
byte[] DF_DEC = new byte[] { 0xDF, 0x71 };
byte[] EF_CIN_CSN = new byte[] { 0xD0, 0x01 };

sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, DF_DEC);
sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, EF_CIN_CSN);
Response r = sendCase2APDU(0x00, 0xB0, 0x00, 0x00, 0x00);
byte[] cin = r.GetData();
```

Listing 4: Reading the certificate

After selecting the right DF / EF, the CIN / CSN can be read.

4 Card OS 5.3 smart card

The Card OS 5.3 is the second generation smart card for cash register. It offers a simplified workflow for signatures, as well as faster signature generation times.

4.1 Workflow

For the different use cases, the following commands should be sent to the smart card in the given order. Communication with the smart card should use the T1 protocol. The commands are described in the following sections. For the commands, it is assumed that the MF is selected. If a different DF is selected, the MF has to be selected first. If the required DF is already selected, the select DF is not required.

Sign hash

1. SELECT DF_SIG
2. VERIFY
3. COMPUTE DIGITAL SIGNATURE

Read certificate

1. SELECT DF_SIG
2. SELECT EF_C_CH_DS
3. READ BINARY

Read CIN / CSN

1. SELECT DF_DEC
2. SELECT EF_CIN_CSN
3. READ BINARY

4.2 Data structures

The smart card contains the master file MF as well as the dedicated file DF_SIG.

4.2.1 DF_SIG

DF_SIG is used for making signatures and reading the certificate. The following application identifier (AID) or file id (FID) can be used to select it.

- AID = { 0xD0, 0x40, 0x00, 0x00, 0x22, 0x00, 0x01 }
- FID = { 0xDF, 0x01 }

DF_SIG contains the elementary file (EF) of the certificate.

- FID = { 0xc0, 0x00 }

4.2.2 MF

MF is used for reading the CIN / CSN. It can be selected with the value:

- MF = { 0x3F, 0x00 }

The EF of the CIN / CSN can be selected with the following value:

- FID = { 0xD0, 0x01 }
- SID = { 0x06 }

4.3 PIN format

The Card OS smart card uses the format 2 PIN block. It is constructed, using 4Bit nibbles, in the following way:

- 1 nibble with the value of 2, which identifies this as a format 2 block
- 1 nibble encoding the length N of the PIN
- N nibbles, each encoding one PIN digit
- 14-N nibbles, each holding the "fill" value 15

For instance, the PIN "123456" means, that the data {0x26 0x12 0x34 0x56 0xFF 0xFF 0xFF 0xFF} should be sent to the smart card.

4.4 Examples

4.4.1 Signature process

The first example shows the signature process.

```
byte[] DF_SIG = new byte[] { 0xDF, 0x01 };

sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, DF_SIG);
sendCase3APDU(0x00, 0x20, 0x00, 0x81, pin);
Response r = sendCase4APDU(0x00, 0x2A, 0x9E, 0x9A, hash, 64);
byte[] signature = r.GetData();
```

Listing 5: Signature process

After selecting the right DF, an authentication via the VERIFY command with the PIN is necessary. Then, the signature can be calculated with the SIGN command, which returns the hash as data.

If the card is used to make more than one signature, it is not necessary to make the selections a second time.

```
sendCase3APDU(0x00, 0x20, 0x00, 0x81, pin);
Response r = sendCase4APDU(0x00, 0x2A, 0x9E, 0x9A, hash, 64);
byte[] signature = r.GetData();
```

Listing 6: Signature process without selection

This results in a faster signature calculation.

4.4.2 Certificate reading

The next example shows how the certificate can be read from the smart card.

```
byte[] DF_SIG = new byte[] { 0xDF, 0x01 };
byte[] EF_C_CH_DS = new byte[] { 0xc0, 0x00 };

sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, DF_SIG);
sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, EF_C_CH_DS);
int offset = 0;
byte[] certificate;
while (true) {
  Response r = sendCase2APDU(0x00, 0xB0, offset >> 8, offset);
  if (r.StatusWord != 0x9000) {
    break;
  }
    byte[] part = r.GetData();
    certificate += part;
  offset += 256;
}
```

Listing 7: Reading the certificate

After selecting the right DF / EF, the certificate can be read sequentially in 256 Byte Blocks.

4.4.3 Reading CIN / CSN

The last example shows how the CIN / CSN can be read from the smart card.

```
byte[] MF = new byte[] { 0x3F, 0x00 };
byte[] EF_CIN_CSN = new byte[] { 0xD0, 0x01 };

sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, MF);
sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, EF_CIN_CSN);
Response r = sendCase2APDU(0x00, 0xB0, 0x00, 0x00, 0x00);
byte[] cin = r.GetData();
```

Listing 8: Reading the certificate

After selecting the right DF / EF, the CIN / CSN can be read.

5 Acos ID smart card

The Acos ID is the next generation smart card for cash registers. It offers the same easy workflow as Card OS 5.3 cards, as well as even faster signature generation times.

The only important difference is, that the VERIFY command requires a different key ID. Instead of 0x81, 0x8A must be used.

Therefore, the PIN verification call has to be adapted from
SendCase3APDU(isoReader, 0x00, 0x20, 0x00, 0x81, formatedPIN);
to
SendCase3APDU(isoReader, 0x00, 0x20, 0x00, 0x8A, formatedPIN);.

5.1 Workflow

For the different use cases, the following commands should be sent to the smart card in the given order. Communication with the smart card should use the T1 protocol. The commands are described in the following sections. For the commands, it is assumed that the MF is selected. If a different DF is selected, the MF has to be selected first. If the required DF is already selected, the select DF is not required.

Sign hash

1. SELECT DF_SIG
2. VERIFY
3. COMPUTE DIGITAL SIGNATURE

Read certificate

1. SELECT DF_SIG
2. SELECT EF_C_CH_DS
3. READ BINARY

Read CIN / CSN

1. SELECT DF_DEC
2. SELECT EF_CIN_CSN
3. READ BINARY

5.2 Data structures

The smart card contains the master file MF as well as the dedicated file DF_SIG.

5.2.1 DF_SIG

DF_SIG is used for making signatures and reading the certificate. The following application identifier (AID) or file id (FID) can be used to select it.

- AID = { 0xD0, 0x40, 0x00, 0x00, 0x22, 0x00, 0x01 }
- FID = { 0xDF, 0x01 }

DF_SIG contains the elementary file (EF) of the certificate.

- FID = { 0xc0, 0x00 }

5.2.2 MF

MF is used for reading the CIN / CSN. It can be selected with the value:

- MF = { 0x3F, 0x00 }

The EF of the CIN / CSN can be selected with the following value:

- FID = { 0xD0, 0x01 }
- SID = { 0x06 }

5.3 PIN format

The Card OS smart card uses the format 2 PIN block. It is constructed, using 4Bit nibbles, in the following way:

- 1 nibble with the value of 2, which identifies this as a format 2 block
- 1 nibble encoding the length N of the PIN
- N nibbles, each encoding one PIN digit
- 14-N nibbles, each holding the "fill" value 15

For instance, the PIN "123456" means, that the data {0x26 0x12 0x34 0x56 0xFF 0xFF 0xFF 0xFF} should be sent to the smart card.

5.4 Examples

5.4.1 Signature process

The first example shows the signature process.

```
byte[] DF_SIG = new byte[] { 0xDF, 0x01 };

sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, DF_SIG);
sendCase3APDU(0x00, 0x20, 0x00, 0x8A, pin);
Response r = sendCase4APDU(0x00, 0x2A, 0x9E, 0x9A, hash, 64);
byte[] signature = r.GetData();
```

Listing 9: Signature process

After selecting the right DF, an authentication via the VERIFY command with the PIN is necessary. Then, the signature can be calculated with the SIGN command, which gets the hash as data.

If the card is used to make more than one signature, it is not necessary to make the selections a second time.

```
sendCase3APDU(0x00, 0x20, 0x00, 0x81, pin);
Response r = sendCase4APDU(0x00, 0x2A, 0x9E, 0x9A, hash, 64);
byte[] signature = r.GetData();
```

Listing 10: Signature process without selection

This results in a faster signature calculation.

5.4.2 Certificate reading

The next example shows how the certificate can be read from the smart card.

```
byte[] DF_SIG = new byte[] { 0xDF, 0x01 };
byte[] EF_C_CH_DS = new byte[] { 0xc0, 0x00 };

sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, DF_SIG);
sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, EF_C_CH_DS);
int offset = 0;
byte[] certificate;
while (true) {
  Response r = sendCase2APDU(0x00, 0xB0, offset >> 8, offset);
  if (r.StatusWord != 0x9000) {
    break;
  }
    byte[] part = r.GetData();
    certificate += part;
  offset += 256;
}
```

Listing 11: Reading the certificate

After selecting the right DF / EF, the certificate can be read sequentially in 256 Byte Blocks.

5.4.3 Reading CIN / CSN

The last example shows how the CIN / CSN can be read from the smart card.

```
byte[] MF = new byte[] { 0x3F, 0x00 };
byte[] EF_CIN_CSN = new byte[] { 0xD0, 0x01 };

sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, MF);
sendCase3APDU(0x00, 0xA4, 0x00, 0x0C, EF_CIN_CSN);
Response r = sendCase2APDU(0x00, 0xB0, 0x00, 0x00, 0x00);
byte[] cin = r.GetData();
```

Listing 12: Reading the certificate

After selecting the right DF / EF, the CIN / CSN can be read.

A Glossary

DF	dedicated file
EF	elementary file
MF	master file