MTAT.07.017
Applied Cryptography

Smart Cards (EstEID)

University of Tartu

Spring 2020
**Smart card**

Chip card or integrated circuit card (ICC)
Contains protected non-volatile memory and microprocessor

ISO/IEC 7816 defines dimensions and location of the contacts, electrical interface, transmission protocols, etc.

- Contact smart cards
- Contactless smart cards
- Dual interface cards
Smart card communication

APDU: Application Protocol Data Unit

terminal → card: command
terminal ← card: response

- Command APDU:
  \[[CLA][INS][P1][P2][L_c][C_{data}]...[L_e]\]
  \[\text{Header (5 bytes) + data (0 ... 255 bytes)}\]
  Case 1: 00 a4 00 0c[00]
  Case 2: 00 b2 01 0c ff
  Case 3: 00 a4 01 0c 02 ee ee
  Case 4: 00 a4 01 00 02 ee ee ff

- Response APDU:
  \[[R_{data}]...[SW1][SW2]\]
  \[\text{Data (0 ... 256 bytes) + status word (2 bytes)}\]
  62 00
  45 53 54 90 00
## Standard commands and responses

<table>
<thead>
<tr>
<th>ClaiNs P1 P2</th>
<th>Lc Send Data</th>
<th>Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0 04 00 00 00</td>
<td>3GPP TS 11.11</td>
<td>INVALIDATE</td>
<td></td>
</tr>
<tr>
<td>84 16 00 00 xx MAC</td>
<td>VSDC</td>
<td>CARD BLOCK</td>
<td></td>
</tr>
<tr>
<td>A0 20 00 xx 08 CHV Value</td>
<td>3GPP TS 11.11</td>
<td>VERIFY</td>
<td></td>
</tr>
<tr>
<td>00 82 00 xx 06 Manual</td>
<td>GEMPLUS MPCOS-EMV</td>
<td>EXTERNAL AUTHENTICATE</td>
<td></td>
</tr>
<tr>
<td>00 84 xx 08 Rnd Num</td>
<td>GEMPLUS MPCOS-EMV</td>
<td>INTERNAL AUTHENTICATE</td>
<td></td>
</tr>
<tr>
<td>00 88 XX xx 0A Manual</td>
<td>3GPP TS 11.11</td>
<td>RUN GSM ALGORITHM</td>
<td></td>
</tr>
<tr>
<td>A0 88 00 00 10 RAND : Rnd num xx SRES( 4B)</td>
<td>3GPP TS 11.11</td>
<td>SEEK</td>
<td></td>
</tr>
<tr>
<td>A0 A4 00 00 xx Pattern xx</td>
<td>GlobalPlatform</td>
<td>SELECT</td>
<td></td>
</tr>
<tr>
<td>A0 A4 00 00 xx AID</td>
<td>3GPP TS 11.11</td>
<td>SELECT</td>
<td></td>
</tr>
<tr>
<td>A0 B0 xx xx</td>
<td>3GPP TS 11.11</td>
<td>READ BINARY</td>
<td></td>
</tr>
<tr>
<td>00 B2 xx 00</td>
<td>VSDC</td>
<td>READ RECORD</td>
<td></td>
</tr>
<tr>
<td>00 C0 1C Key Info</td>
<td>GlobalPlatform</td>
<td>GET RESPONSE</td>
<td></td>
</tr>
<tr>
<td>80 CA xx xx</td>
<td>VSDC</td>
<td>GET DATA</td>
<td></td>
</tr>
<tr>
<td>80 D0 xx xx xx Data to be written in EEPROM</td>
<td>VSDC</td>
<td>LOAD STRUCTURE</td>
<td></td>
</tr>
<tr>
<td>A0 D6 xx xx xx Data to be written in EEPROM</td>
<td>3GPP TS 11.11</td>
<td>UPDATE BINARY</td>
<td></td>
</tr>
<tr>
<td>00 DA xx xx xx Data</td>
<td>VSDC</td>
<td>PUT DATA</td>
<td></td>
</tr>
<tr>
<td>00 DC xx xx xx Data (and MAC)</td>
<td>VSDC</td>
<td>UPDATE RECORD</td>
<td></td>
</tr>
<tr>
<td>80 E0 xx xx xx PCI length</td>
<td>3GPP TS 11.11</td>
<td>CREATE FILE</td>
<td></td>
</tr>
<tr>
<td>00 E2 00 00 xx Record</td>
<td>3GPP TS 11.11</td>
<td>APPEND RECORD</td>
<td></td>
</tr>
<tr>
<td>A0 E4 00 00 02 xx xx</td>
<td>3GPP TS 11.11</td>
<td>DELETE FILE</td>
<td></td>
</tr>
</tbody>
</table>

### SW1 SW2 Message

| ‘6X XX’ | Transmission protocol related codes |
| ‘61 XX’ | SW2 indicates the number of response bytes still available |
| ‘62 00’ | No information given |
| ‘62 82’ | The end of the file has been reached before the end of reading |
| ‘62 83’ | Invalid DF |
| ‘62 84’ | Selected file is not valid. File descriptor error |
| ‘6A 00’ | Bytes P1 and/or P2 are incorrect. |
| ‘6A 82’ | File not found |
| ‘9F XX’ | Success, XX bytes of data available to be read via “Get_Response” task. |

http://web.archive.org/web/20090623030155/http://cheef.ru/docs/HowTo/SW1SW2.info
### Smart card file system

- **Adressable objects:**
  - MF – Master File (root directory)
  - DF – Dedicated File (directory)
  - EF – Elementary File (data file)
- 2-byte file identifier (FID)
- There is no `ls`/`dir` command!
- Legacy
Using SELECT FILE

To change pointer to Dedicated File EEEE:
[0x00, 0xA4, 0x01, 0x0C, 0x02, 0xEE, 0xEE]

- **CLA** - 0x00
- **INS** - 0xA4 (command - SELECT FILE)
- **P1** - what type of object to select
  - 0x00 - Master File (root)
  - 0x01 - Dedicated File (directory)
  - 0x02 - Elementary File (data file)
  - 0x04 - Card Application (chip applet)
- **P2** - type of response
  - 0x00 - Include object description FCI (FCP+FMD)
  - 0x04 - Include object description FCP (file control parameters)
  - 0x08 - Include object description FMD (file management data)
  - 0x0C - Do not respond with description
- **Lc** - length of file identifier
- **Data** - file identifier for EF, DF or AID for application
Answer To Reset (ATR)

“Bytes returned by a contact smart card on power up. Conveys information about the parameters proposed by the card.”

```
$ pcsc_scan
ATR: 3B FA 18 00 00 80 31 FE 45 FE 65 49 44 20 2F 20 50 4B 49 03
+ TS = 3B --> Direct Convention
+ T0 = FA, Y(1): 1111, K: 10 (historical bytes)
  TA(1) = 18 --> Fi=372, Di=12, 31 cycles/ETU
  129032 bits/s at 4 MHz, fMax for Fi = 5 MHz => 161290 bits/s
TB(1) = 00 --> VPP is not electrically connected
TC(1) = 00 --> Extra guard time: 0
TD(1) = 80 --> Y(i+1) = 1000, Protocol T = 0
TD(2) = 31 --> Y(i+1) = 0011, Protocol T = 1
 TA(3) = FE --> IFSC: 254
TB(3) = 45 --> Block Waiting Integer: 4 - Character Waiting Integer: 5
+ Historical bytes: FE 65 49 44 20 2F 20 50 4B 49
  Category indicator byte: FE (proprietary format)
+ TCK = 03 (correct checksum)
Possibly identified card (using /home/user/.cache/smartcard_list.txt):
3B FA 18 00 00 80 31 FE 45 FE 65 49 44 20 2F 20 50 4B 49 03
  Estonian Identity Card (EstEID v3.5 (10.2014) cold) (eID)
```

Historical bytes can be used to identify the card:

```python
>>> "FE654944202F20504B49".decode('hex')
'\xfeeID / PKI'
```

Some cards can return two different ATRs:

- Cold ATR – when power is supplied to the card
- Warm ATR – when reset signal is sent
Transmission protocols (T=0/T=1)

T=0 (byte-oriented protocol):

- APDU is sent over the wire as it is
- In one round data can be sent only in one direction:
  - It must be known in which direction the data will be sent
  - Data sent/received must be exactly P3 bytes in length

To return data if terminal also sent data:
1. Card responds with SW: 61 XX
2. Terminal sends GET RESPONSE: 00 C0 00 00 XX
3. Card returns XX bytes + SW

If APDU contains incorrect $L_e$ value:
1. Card responds with SW: 6C XX
2. Terminal resends the command with $L_e$ set to XX
3. Card returns XX bytes + SW
Transmission protocols (T=0/T=1)

T=1 (block-oriented protocol):

- APDUs are encapsulated in blocks forming Transport Protocol Data Units
- In one round data can be sent in both directions
- Supports extended APDU (max 65,535 bytes)
- More advanced error detection

The same APDU can be sent over both T=0 and T=1 electrical protocols (unless using T=1 extended APDU feature).
Electrical communication

- **VCC** – Positive supply voltage (1.8V, 3.0V, 5.0V)
- **GND** – Ground
- **RST** – Reset signal to boot and restart the card
- **I/O** – Synchronous data transmission
- **CLK** – Clock signal for I/O sync (1 MHz to 5MHz)
  - Used by older cards to clock CPU
- **C4, C8, C6** – not used
Sniffing electrical communication

- Saleae USB Logic Analyzer 8CH 24MHz – $220
  - Chinese clone on eBay – EUR 6
- Wires soldered to the reader contacts (I/O, VCC, RST, CLK)
Sniffing electrical communication
Estonian Electronic Identity Card (EstEID)

Nationwide PKI:

- Subjects – Estonian (e-)residents
- Certificate Authority (QTSP) – SK ID Solutions AS
- Registration Authority – Police and Border Guard Board
- Manufacturer – Gemalto / IDEMIA
  - In addition generates key pair for you
Electronic functionality of Estonian ID card

- Two RSA/ECC key pairs:
  - Authentication (and decryption) key
  - Digital signature key
- Corresponding certificates
- PIN1/PIN2/PUK
- PIN retry counters
- Personal data file (16 records)
- Key usage counters
- Card management operations
  - PIN reset
  - Certificate overwrite
  - Applet reinstall
ID card chip platforms

(a) MICARDO

(b) MULTOS

(c) jTOP SLE66/SLE78 JavaCard

(d) Oberthur IAS-ECC JavaCard

EstEID specifications:

Preparation: hardware

- Get a smart card reader
  - OMNIKEY CardMan 1021 – EUR 6 (Swedbank)
  - Gemalto IDBridge CT710 – EUR 17 (SEB)
  - Pluss ID (+iD) – EUR 13 (Klick)

- Built-in readers may have issues
- Plug the reader into the USB port
Preparation: hardware

- If using VirtualBox forward USB to guest Ubuntu
  - For USB 2.0/3.0 support install VirtualBox Extension Pack
  - Uninstall USBPcap (from Wireshark) from guest OS
  - Add USB filter if host OS fails to release the USB device
    - After filter is created, new USB devices attached will be automatically redirected to guest OS

- Check if the smart card reader is detected by Ubuntu:

  $ dmesg
  [ 1599.744116] usb 4-2: new full-speed USB device number 3 using uhci_hcd
  [ 1599.921740] usb 4-2: New USB device found, idVendor=08e6, idProduct=3437
  [ 1599.921751] usb 4-2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
  [ 1599.921760] usb 4-2: Product: USB SmartCard Reader
  [ 1599.921767] usb 4-2: Manufacturer: Gemplus

  $ lsusb
  Bus 004 Device 003: ID 08e6:3437 Gemplus GemPC Twin SmartCard Reader  <-- external USB
  Bus 005 Device 002: ID 03f0:0324 Hewlett-Packard SK-2885 keyboard
  Bus 002 Device 004: ID 0b97:7762 02 Micro Inc. Oz776 SmartCard Reader  <-- DELL's built-in
Preperation: software

- Install pcscd:
  
  ```
  $ sudo apt install pcscd pcsc-tools
  $ pcsc_scan -n
  Scanning present readers...
  0: O2 Micro Oz776 00 00
  1: Gemalto PC Twin Reader 01 00
  Reader 0: O2 Micro Oz776 00 00
  Card state: Card removed,
  Reader 1: Gemalto PC Twin Reader 01 00
  Card state: Card inserted,
  ATR: 3B DE 18 FF C0 80 B1 FE 45 1F 03 45 73 74 45 49 44 20 76 65 72 20 31 2E 30 2B
  Possibly identified card (using /usr/share/pcsc/smartcard_list.txt):
  Estonian Identity Card (EstEID v1.0 2006 cold)
  ```

- Install pyscard:
  
  ```
  $ sudo apt install python3-pyscard
  $ python3
  >>> import smartcard
  >>> smartcard.System.readers()
  ['O2 Micro Oz776 00 00', 'Gemalto PC Twin Reader 01 00']
  >>> connection = smartcard.System.readers()[1].createConnection()
  >>> connection.connect()
  >>> connection.getATR()
  >>> connection.transmit([0x0a, 0xa4, 0x00, 0x00, 0x02])
  ([], 110, 0)
  ```
import sys
from smartcard.CardType import AnyCardType
from smartcard.CardRequest import CardRequest
from smartcard.CardConnection import CardConnection
from smartcard.util import toHexString

# this will wait until card inserted in any reader
channel = CardRequest(timeout=10, cardType=AnyCardType()).waitforcard().connection

# using T=0 for compatibility and simplicity
channel.connect(CardConnection.T0_protocol)

print("[+] Selected reader:", channel.getReader())

# detect and print EstEID card type (EstEID spec page 15)
atr = channel.getATR()
if atr == [0x3B,0xFE,0x18,0x00,0x00,0x80,0x31,0xFE,0x45,0x45,0x73,...]:
    print("[+] EstEID v3.x on JavaCard")
elif atr == [0x3B,0xFA,0x18,0x00,0x00,0x80,0x31,0xFE,0x45,0xFE,0x65,...]:
    print("[+] EstEID v3.5 (10.2014) cold (eID)")
elif atr == [0x3B,0xDB,0x96,0x00,0x80,0xB1,0xFE,0x1F,0x83,0x00,...]:
    print("[+] Estonian ID card (2018)")
else:
    print("[-] Unknown card:", toHexString(atr))
sys.exit(1)
from smartcard.util import toHexString

def send(apdu):
    data, sw1, sw2 = channel.transmit(apdu)

    # success
    if [sw1, sw2] == [0x90, 0x00]:
        return data

    # (T=0) card signals how many bytes to read
    elif sw1 == 0x61:
        return send([0x00, 0xC0, 0x00, 0x00, sw2])  # GET RESPONSE of sw2 bytes

    # (T=0) card signals incorrect Le
    elif sw1 == 0x6C:
        return send(apdu[0:4] + [sw2])  # resend APDU with Le = sw2

    # probably error condition
    else:
        print("Error: %02x %02x, sending APDU: %s" % (sw1, sw2, toHexString(apdu)))
        sys.exit(1)

• APDU commands and responses are lists containing integers (e.g., [0, 50, 199, 255])
• For pretty-printing, a list of integers can be converted to hex string with spaces (e.g., toHexString([0, 50, 199, 255]) == "00 32 C7 FF")
• To convert list of integers to bytes, use: bytes([97, 98, 67]) == b"abC".
Task 1: EstEID info – 2p

Implement a utility that reads personal data file and PIN retry counters from the ID card:

$ ./esteid_info.py
[+] Selected reader: Gemalto PC Twin Reader 00 00
[+] EstEID v3.5 (10.2014) cold (eID)
[+] Personal data file:
[1]Surname: PARŠOVS
[2]First name line 1: ARNIS
[3]First name line 2:
[4]Sex: M
[6]Birth date: 05.08.1986
[7]Personal ID code: 38608050013
[9]Expiry date: 27.08.2020
[10]Place of birth: LÄTI / LVA
[11]Date of issuance: 27.08.2015
[12]Type of residence permit:
[13]Notes line 1: EL KODANIK / EU CITIZEN
[14]Notes line 2: ALALINE ELAMISÖIGUS
[15]Notes line 3: PERMANENT RIGHT OF RESIDENCE
[16]Notes line 4: LUBATUD TÖÖTADA

[+] PIN retry counters:
PIN1: 3 left
PIN2: 3 left
PUK: 3 left

$ ./esteid_info.py
[+] Selected reader: Gemalto PC Twin Reader 00 00
[+] Estonian ID card (2018)
[+] Personal data file:
[1]Surname: PARŠOVS
[2]First name: ARNIS
[3]Sex:
[4]Citizenship:
[5]Date & place of birth:
[6]Personal ID code: 38608050013
[8]Expiry date: 02 01 2024
[9]Date & place of issuance: 02 01 2019
[10]Type of residence permit:
[11]Notes line 1:
[12]Notes line 2:
[13]Notes line 3:
[14]Notes line 4:
[15]Notes line 5:

[+] PIN retry counters:
PIN1: 3 left
PIN2: 3 left
PUK: 3 left
Task 1: Personal data file

EstEID spec page 24:

- Select MF/EEEE/5044
- With READ RECORD read all 16 records from the file
- Decode them to unicode using CP1252 codepage

```python
send([0x00, 0xA4, 0x00, 0x0C]) # SELECT FILE (MF)
send([0x00, 0xA4, 0x01, 0x0C]+[0x02, 0xEE, 0xEE]) # MF/EEEE
send([0x00, 0xA4, 0x02, 0x0C, 0x02, 0x50, 0x44]) # MF/EEEE/5044
r = send([0x00, 0xB2, 0x07, 0x04]) # READ RECORD 7
print("Personal ID code:", bytes(r).decode("cp1252"))
```

EstEID 2018 spec page 30:

- Select AID: A000000077010800070000FE00000100
- Select MF/5000/50XX (XX – personal data file record 1 to 15)
- With READ BINARY read the contents of the file

```python
send([0x00, 0xA4, 0x04, 0x00, 0x10, 0xA0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x77, 0x01, ...
send([0x00, 0xA4, 0x00, 0x0C]) # SELECT FILE (MF)
send([0x00, 0xA4, 0x01, 0x0C]+[0x02, 0x50, 0x00]) # MF/5000
send([0x00, 0xA4, 0x02, 0x0C]+[0x02, 0x50, 0x06]) # MF/5000/5006
r = send([0x00, 0xB0, 0x00, 0x00, 0x00, 0x00]) # READ BINARY
print("Personal ID code:", bytes(r).decode("utf8"))
```
Task 1: PIN retry counters

EstEID spec page 28:
- Select MF/0016
- With READ RECORD read records 1, 2 and 3 (for PIN1, PIN2 and PUK, respectively)
- Record’s 6th byte will contain integer value of how many tries are left

EstEID_{2018} spec page 25:
- Select MF for PIN1 and PUK
- Select MF/ADF2 for PIN2
- Send an empty VERIFY in P2 specifying 1, 2 or 133 (for PIN1, PUK and PIN2, respectively)
- VERIFY will return status word 63CX (X - remaining tries)
Task 2: EstEID getcert – 2p

Implement a utility that downloads the certificates stored on the ID card:

```
$ ./esteid_getcert.py --cert auth --out auth.pem
[+] Selected reader: Gemalto PC Twin Reader 00 00
[+] EstEID v3.5 (10.2014) cold (eID)
[=] Retrieving auth certificate...
[+] Certificate size: 1505 bytes
[+] Certificate stored in auth.pem
$ openssl x509 -in auth.pem -text | grep 'X509v3 Key Usage' -A 1
X509v3 Key Usage: critical
  Digital Signature, Key Agreement

$ ./esteid_getcert.py --cert sign --out sign.pem
[+] Selected reader: Gemalto PC Twin Reader 00 00
[+] Estonian ID card (2018)
[=] Retrieving sign certificate...
[+] Certificate size: 973 bytes
[+] Certificate stored in sign.pem
$ openssl x509 -in sign.pem -text | grep 'X509v3 Key Usage' -A 1
X509v3 Key Usage: critical
  Non Repudiation

$ wget https://c.sk.ee/esteid2018.pem.crt
$ openssl verify -partial_chain -CAfile ESTEID-SK_2015.pem.crt sign.pem
error sign.pem: verification failed
$ openssl verify -partial_chain -CAfile esteid2018.pem.crt sign.pem
sign.pem: OK
```
Task 2: certificate reading

- Select AID: A000000077010800070000FE00000100\textsubscript{2018}
- Select MF/EEEE/AACE or MF/ADF1/3401\textsubscript{2018} (authentication)
- Select MF/EEEE/DDCE or MF/ADF2/341F\textsubscript{2018} (digital signature)
- Certificate is stored in a DER form in a file of fixed size
  - With READ BINARY read the first 10 bytes of certificate
  - Calculate the length of certificate by parsing the length byte(s) of certificate’s outer ASN.1 SEQUENCE
    - All possible DER length values must be correctly handled
- Read the whole certificate (in a loop) using READ BINARY
  - With one READ BINARY maximum 231 bytes can be read
  - The offset is two byte integer. The most significant byte must be specified in P1, the least significant byte in P2.
    - Two byte integer \(i\) can be split into [MSB, LSB] using [\(i\gg8, i\&0xFF\)]
  - Make sure that only the minimum number of bytes are sent to/from the card (!)
Biometric passport (e-passport)

- Contactless smart card chip
- Stores information printed on the data page
  - Including facial image – 20KB JPG 480x640
- Data digitally signed by country signing CA
- Can be read only using the key encoded in MRZ
- Fingerprint reading requires terminal authentication
- Possibility for automated border clearance or “E-gates”