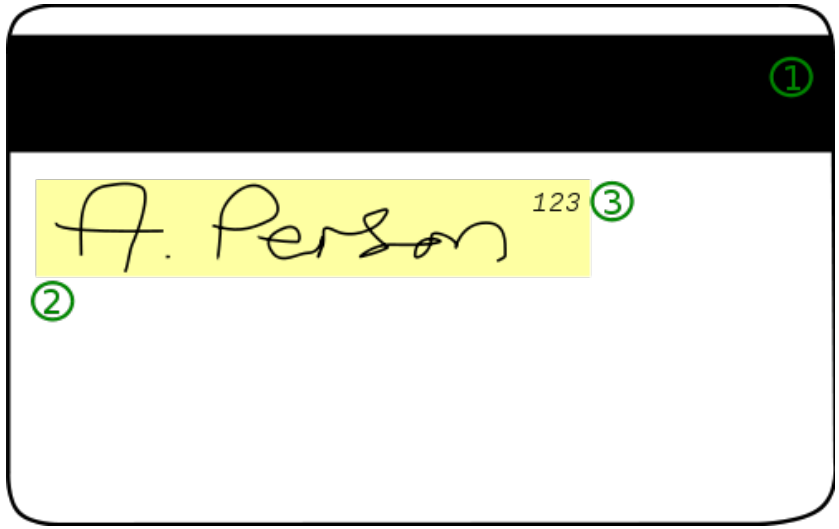
MTAT.07.017
Applied Cryptography

Smart Cards (EstEID)

University of Tartu

Spring 2017

Magnetic Stripe Card

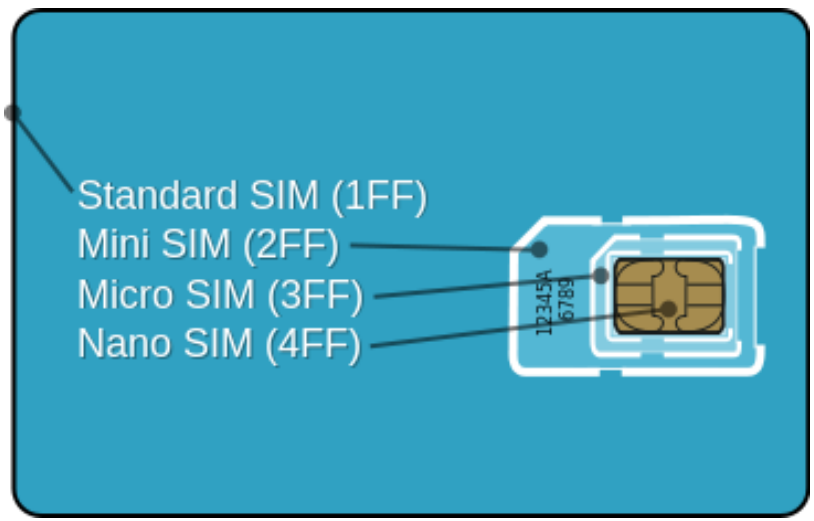


- Not a smart card!
- Three-track stripe:
  - Track 1 holds 79 6-bit plus parity bit characters
  - Track 2 holds 40 4-bit plus parity bit characters
  - Track 3 holds 107 4-bit plus parity bit characters
- Easily modifiable and cloneable
- Magnetic stripe and cryptography

Smart Card

A.K.A. chip card or integrated circuit card (ICC)
Contains protected non-volatile memory and microprocessor

ISO/IEC 7816 defines dimentions 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 $\longrightarrow$ card: command
terminal $\longleftarrow$ card: response

- Command APDU:
  `[CLA] [INS] [P1] [P2]` $[L_c]$ $[C_{data}] \ldots$ $[L_e]$
  Header (4 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` ~~`0a`~~
- Response APDU:
  $[R_{data}] \ldots$ `[SW1] [SW2]`
  Data (0 ... 256 bytes) + status word (2 bytes)

  `62 00`
  `45 53 54 90 00`

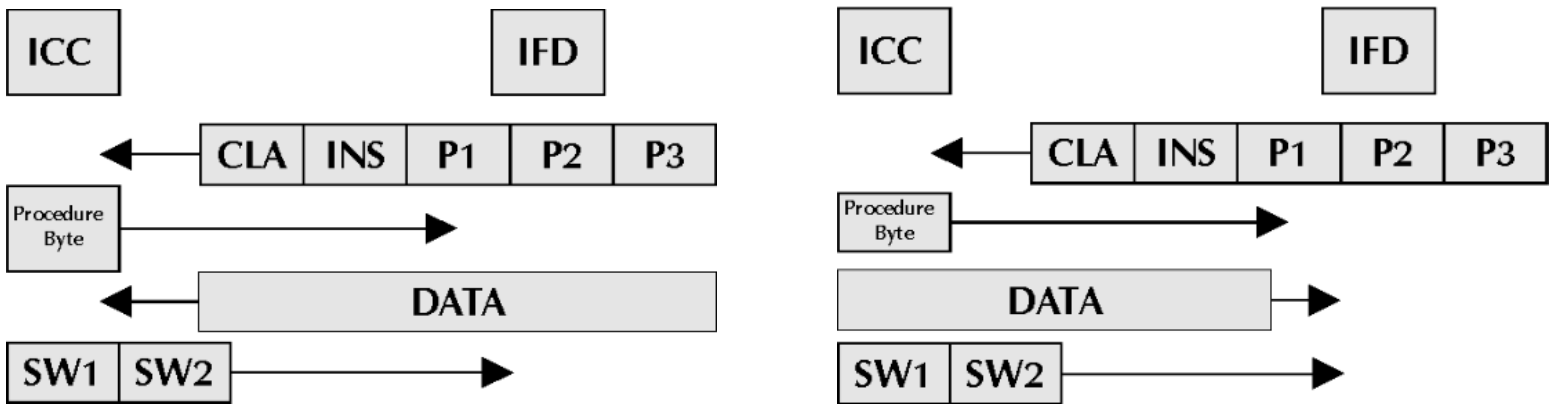
Electrical Transmission Protocols (T=0/T=1)

T=1 (block-oriented protocol):

- Supports extended APDU (max 65'535 bytes)

T=0 (byte-oriented protocol):

- Simplicity and minimal memory requirements
- Data can be sent only in one direction:



  - Reading data from card:
    1. Terminal sends data APDU (or APDU with incorrect L_e)
    2. Card responds with SW: `61 XX`
    3. Terminal sends `GET RESPONSE` command: `00 C0 00 00 XX`
    4. Card returns XX bytes + SW

Standard Commands

ClaIns P1 P2	Lc Send Data	Le Recv Data	Specification	Description
A0 04 00 00 00			3GPP TS 11.11	INVALIDATE
84 16 00 00 xx MAC			VSDC	CARD BLOCK
A0 20 00 xx 08 CHV Value			3GPP TS 11.11	VERIFY
00 82 00 xx 06 Manual			GEMPLUS MPCOS-EMV	EXTERNAL AUTHENTICATE
00 84 xx xx		08 Rnd Num	GEMPLUS MPCOS-EMV	GET CHALLENGE
00 88 XX xx 0A Manual			GEMPLUS MPCOS-EMV	INTERNAL AUTHENTICATE
A0 88 00 00 10 RAND : Rnd num		xx SRES(4B)	3GPP TS 11.11	RUN GSM ALGORITHM
A0 A2 00 xx xx Pattern		xx	3GPP TS 11.11	SEEK
00 A4 04 00 xx AID		00	GlobalPlatform	SELECT
00 A4 00 xx xx File ID || Name		00 Manual	VSDC	SELECT
A0 A4 00 00 02 File ID			3GPP TS 11.11	SELECT
A0 B0 xx xx		xx	3GPP TS 11.11	READ BINARY
00 B2 xx		00	VSDC	READ RECORD
A0 B2 xx xx		xx	3GPP TS 11.11	READ RECORD
00 C0		1C Key Info	GlobalPlatform	GET RESPONSE
A0 C0 00 00		xx	3GPP TS 11.11	GET RESPONSE
80 CA xx xx xx			VSDC	GET DATA
80 D0 xx xx xx Data to be written in EEPROM			VSDC	LOAD STRUCTURE
A0 D6 xx xx xx Data to be written in EEPROM			3GPP TS 11.11	UPDATE BINARY
00 DA xx xx xx Data			VSDC	PUT DATA
00 DC xx xx xx Data (and MAC)			VSDC	UPDATE RECORD
A0 DE 00 00 03 Data			3GPP TS 11.11	LOAD AoC(SICAP)
80 E0 xx xx xx FCI length			3GPP TS 11.11	CREATE FILE
00 E2 00 00 xx Record			3GPP TS 11.11	APPEND RECORD
A0 E4 00 00 02 xx xx			3GPP TS 11.11	DELETE FILE
...			...	...

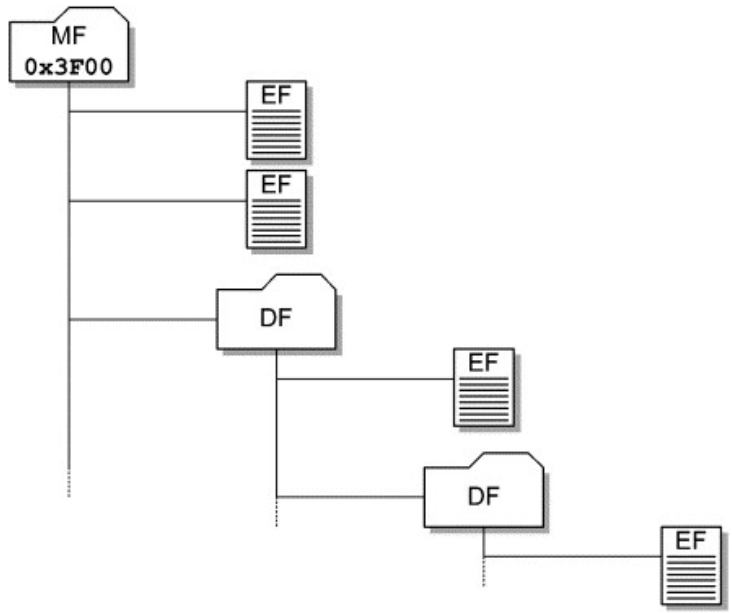
http://web.archive.org/web/20090630004017/http://cheef.ru/docs/HowTo/APDU.info

Standard Status Words

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 81'	Returned data may be corrupted
'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
'63 00'	Authentification failed. Invalid secret code or forbidden value
'63 81'	File filled up by the last write
'6A 00'	Bytes P1 and/or P2 are incorrect.
'6A 82'	File not found
'6A 83'	Record not found
'6A 84'	There is insufficient memory space in record or file
'6A 85'	Lc inconsistent with TLV structure
'6A 86'	Incorrect parameters P1-P2
'6A 87'	The P3 value is not consistent with the P1 and P2 values.
'6A 88'	Referenced data 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

Answer To Reset (ATR)

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

Historical bytes can be used to identify 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)

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

Some cards can return two different ATRs:

- Cold ATR – when power is supplied to the card
- Warm ATR – when signal on RST pin is given

Preparation: Hardware

- Get a smart card reader
  - Can buy one in Swedbank ~~or SEB~~ for EUR 5.75
  - May be problems with readers built-in DELL laptops
- Plug the reader into the USB port
  - If using VirtualBox forward USB to guest Ubuntu
  - Check if smart card reader 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 002 Device 002: ID 413c:a005 Dell Computer Corp. Internal 2.0 Hub
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 003: ID 0b97:7761 O2 Micro, Inc. Oz776 1.1 Hub
Bus 002 Device 004: ID 0b97:7762 O2 Micro, Inc. Oz776 SmartCard Reader <--- DELLs built-in
```

Preperation: Software

- Install pcscd (this will allow to send APDUs to smart card):

```
$ sudo apt-get install pcscd pcsc-tools
$ dpkg --list | grep -i pcsc
ii  libpcsc-perl   Perl interface to the PC/SC smart card library
ii  libpcsclite1   Middleware to access a smart card using PC/SC (library)
ii  pcsc-tools     Some tools to use with smart cards and PC/SC
ii  pcscd          Middleware to access a smart card using PC/SC (daemon side)

$ 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)

$ scriptor
No reader given: using Gemalto PC Twin Reader 00 00
Using T=0 protocol
Reading commands from STDIN
00 02 00 06 06
> 00 02 00 06 06
< EF B1 C6 C3 EF B1 90 00 : Normal processing.
```

Preperation: Software

- Install pyscard (we want to send APDUs using python):

```
$ sudo apt-get install python-pyscard
$ dpkg --list | grep -i pyscard
ii  python-pyscard Python wrapper above PC/SC API

$ python
>>> import smartcard
>>> smartcard.System.readers()
['02 Micro Oz776 00 00', 'Gemalto PC Twin Reader 01 00']
>>> connection = smartcard.System.readers()[1].createConnection()
>>> connection.connect()
>>> connection.getATR()
[59, 222, 24, 255, 192, 128, 177, 254, 69, 31, 3, 69, 115, 116, 69, 73, 68, 32, 118, 101, 1
>>> connection.transmit([0x0a, 0xa4, 0x00, 0x00, 0x02])
([], 110, 0)
>>> connection.getATR()
Traceback (most recent call last):
  File "/usr/lib/python2.7/dist-packages/smartcard/pcsc/PCSCCardConnection.py", line 163, i
    SCardGetErrorMessage(hresult))
smartcard.Exceptions.CardConnectionException: Failed to get status: Card was removed.
```

http://pyscard.sourceforge.net/pyscard-usersguide.html

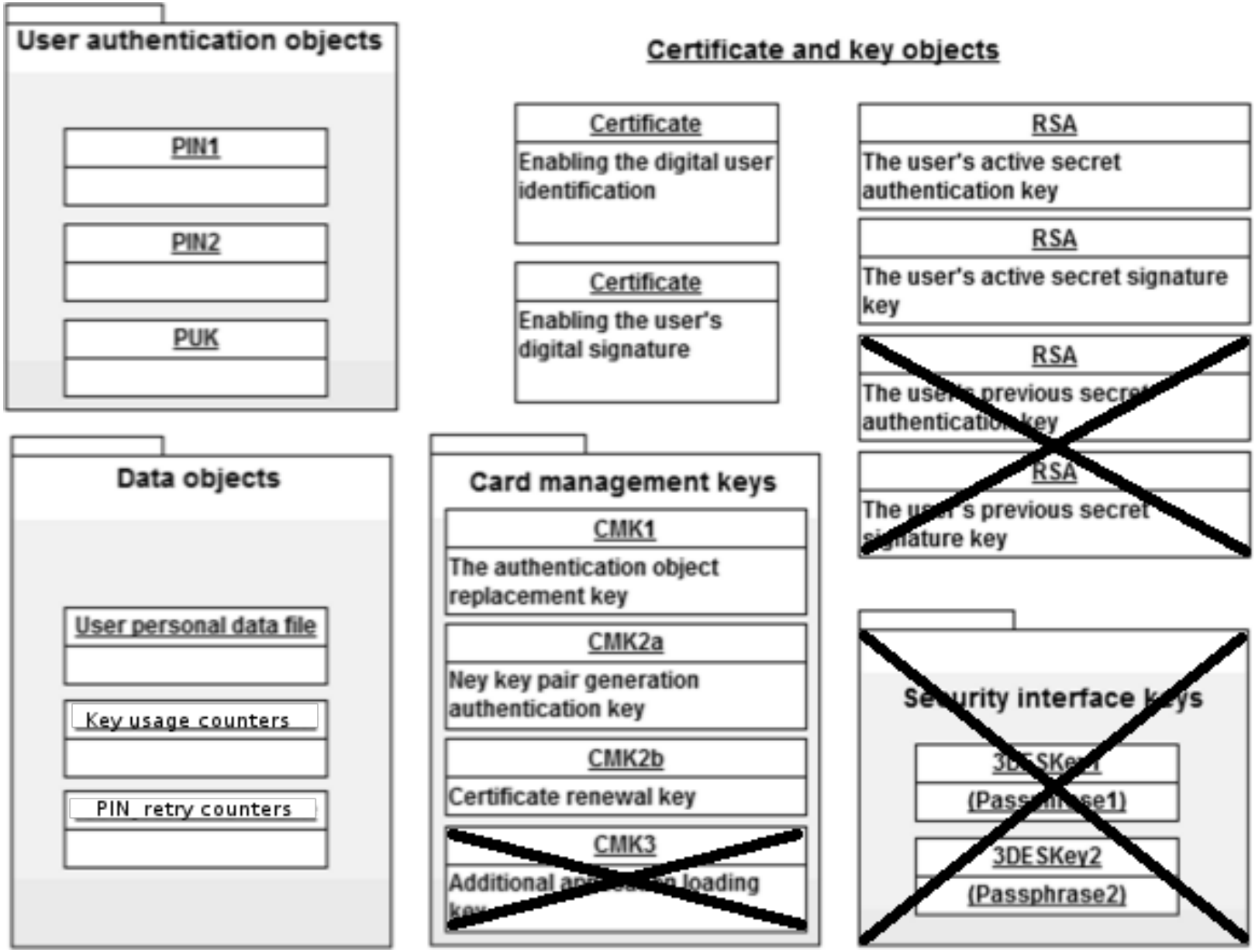
Estonian ID card

There are several types of electronic ID cards:

EstEID specification in English (includes examples):

`http://www.id.ee/public/TB-SPEC-EstEID-Chip-App-v3.4.pdf`

Objects on security chip (spec page 11)



Security chip operations (spec page 12)

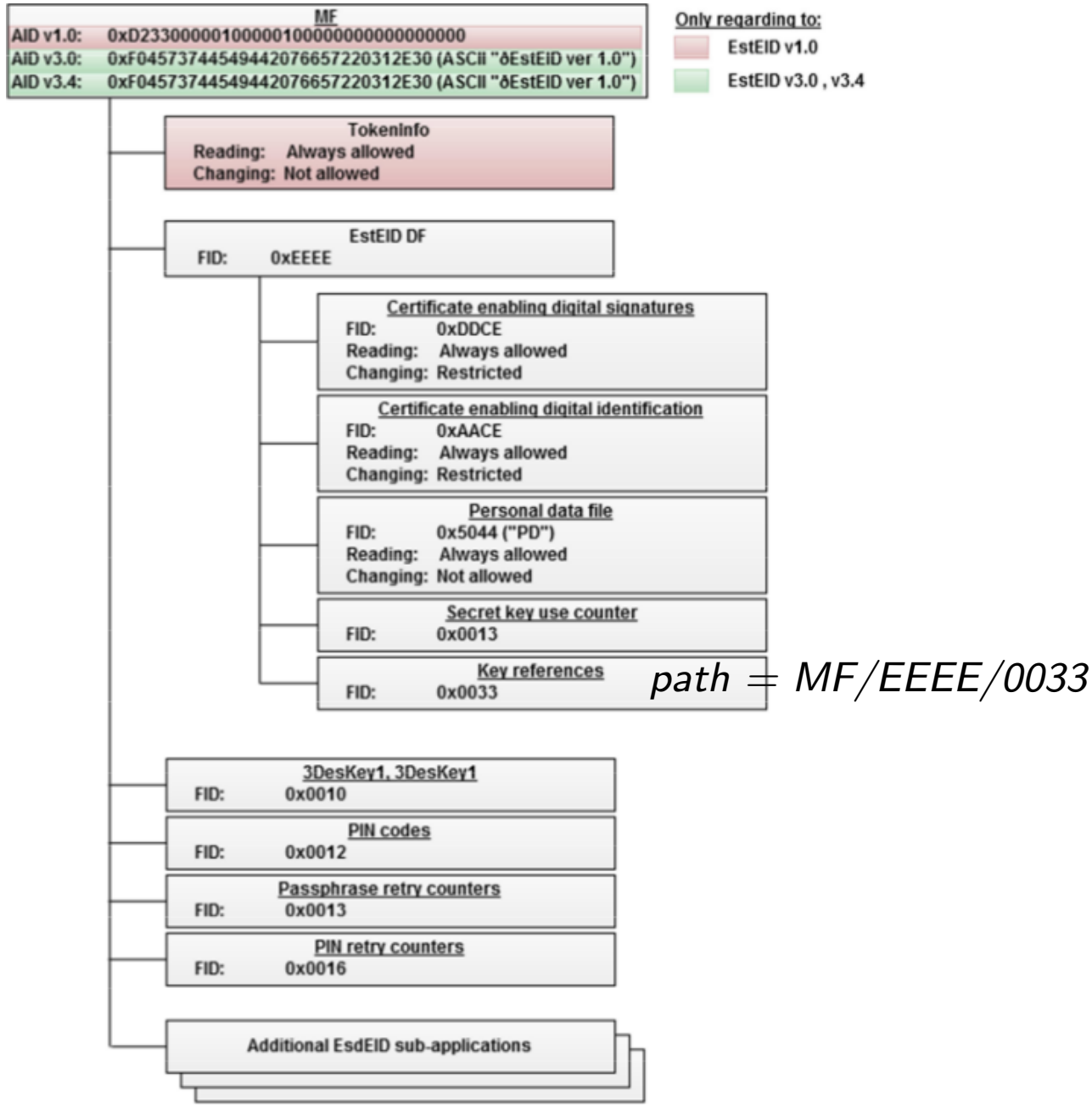
EstEID enables execution of the following operations:

1. The certificate and data reading operations
   a. Reading certificates; Certificate retrieval
   b. Reading the card user personal data file.
2. The administration of the card user authentication objects
   a. Changing the values of PIN1, PIN2 and PUK;
   b. Resetting the consecutive incorrect entries of PIN1 and PIN2;
   c. ~~Assigning values to 3DESKeys.~~
3. Card user authentication
   a. card user authentication with PIN1, PIN2 and PUK;
   b. ~~card user authentication with 3DESKey1 and 3DESKey2.~~
4. Operations with secret keys (sign/decrypt)
5. Card management operations
   a. Replacing authentication objects;
   b. Generating new key pairs;
   c. Loading certificates;
   d. ~~Loading and deleting additional applications;~~
   e. ~~Forming secure loading command series.~~

Chip versions (spec page 13)

Table 2-1 EstEID card application versions' properties				
	v1.0 and v1.0 since 2006	**v1.1 (DigiID)**	**v3.0 and v3.4 since 18.01.2011**	**v3.4**
Implementation platform	MICARDO	Multos	Java Card	
RSA module length	1024 bits		2048 bits	1024, 1280, 1536, 1984, 2048 bits
Hash algorithm support	SHA-1, SHA-224	SHA-1, SHA-224, SHA-256, SHA-384, SHA-518		
ECC module lengths	Not supported		**160** bits **192** bits **224** bits **256** bits	The RSA key length of the same security level ~1024 bits ~1536 bits ~2048 bits ~3072 bits
PKCS#1 padding support	v1.5			
Protocol support	T=0, T=1	T=0	T=0, T=1	T=0, T=1

EstEID file system (spec page 105)



path = MF/EEEE/0033

APDU commands (spec page 117)

Appendix Table 1-1 APDU commands			
Command	**INS**	**Brief description**	**See page**
CHANGE REFERENCE DATA	´24´	Changes the password for cardholder authentication	112
CREATE FILE	´E0´	Creates a directory or data field	115
EXTERNAL AUTHENTICATE /MUTUAL AUTHENTICATE	´82´	Authenticates the external world / external world and chip card	116
GENERATE PUBLIC KEY PAIR	´46´	Generates the public and the private part of a RSA key pair.	118
GET CHALLENGE	´84´	Generates and outputs a random number	120
GET RESPONSE	´C0´	Reads out the response data (T=0)	121
INTERNAL AUTHENTICATE	´88´	Authenticates the chip card or application	112
MANAGE SECURITY ENVIRONMENT	´22´	Passes on key references, random numbers, and data to be used for key derivation	123
PERFORM SECURITY OPERATION	´2A´	Various functions using symmetrical and asymmetrical keys: - COMPUTE DIGITAL SIGNATURE - DECIPHER - HASH	125 126 128 129
READ BINARY	´B0´	Reads from a transparent data field	131
READ RECORD	´B2´	Reads from a formatted data field	133
RESET RETRY COUNTER	´2C´	Resets the counter of failed attempts	134
SELECT FILE	´A4´	Selects a directory or data field	136
UPDATE BINARY	´D6´	Modifies a transparent data field	137
UPDATE RECORD	´DC´	Modifies a formatted data field	139
VERIFY	´20´	Authenticates the cardholder	140

Establishing connection

```
import sys
from smartcard.CardType import AnyCardType
from smartcard.CardRequest import CardRequest
from smartcard.CardConnection import CardConnection
from smartcard.util import toHexString, HexListToBinString

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

# using T=0 for compatibility (i.e., DigiID) 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,0x94,0x00,0xFF,0x80,0xB1,0xFA,0x45,0x1F,0x03,0x45,...]:
    print "[+] EstEID v1.0 on Micardo Public 2.1"
elif atr == [0x3B,0xDE,0x18,0xFF,0xC0,0x80,0xB1,0xFE,0x45,0x1F,0x03,...]:
    print "[+] EstEID v1.0 on Micardo Public 3.0 (2006)"
elif atr == [0x3B,0x6E,0x00,0x00,0x45,0x73,0x74,0x45,0x49,0x44,0x20,...]:
    print "[+] EstEID v1.1 on MultiOS (DigiID)"
elif 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)"
else:
    print "[-] Unknown card:", toHexString(atr)
    sys.exit()
```

Transmitting APDUs

```
from smartcard.util import toHexString, HexListToBinString
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
    # probably error condition
    else:
        print "Error: %02x %02x, sending APDU: %s" % (sw1, sw2, toHexString(apdu))
        sys.exit()
```

- 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 (i.e., `toHexString([0,50,199,255])=="00 32 C7 FF"`)
- To convert list of integers to byte string use `HexListToBinString([97,98,67])=="abC"`.

Using SELECT FILE (spec page 24 and 141)

To change pointer to Dedicated File EEEE:
`send([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 (if present)
- Data - file identifier for EF, DF or application (if present)

Task 1

Implement utility that displays personal data file, PIN retry and key usage counters on ID card.

```
$ python 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
        [5]Nationality: LVA
        [6]Birth date: 05.08.1986
        [7]Personal identification code: 38608050013
        [8]Document number: EA0043798
        [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 ELAMISÕIGUS
        [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
[+] Key usage counters:
        signature key: 0 times
        authentication key: 30 times
```

Put your output in `esteid_info.out` on your repository!

Task 1: Personal data file (spec page 24)

- Select MF/EEEE/5044
- Read all personal data file records with READ RECORD
  - Ignore the specification – read all 16 records
- Decode them to unicode using CP1252 codepage (i.e., `"somestring".decode("cp1252").encode("utf8")`)

Example for obtaining personal identification code:

```
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
record = send([0x00, 0xB2, 0x07, 0x04]) # READ RECORD 7th
print "Personal identification code:",
    HexListToBinString(record).decode("cp1252").encode("utf8")
```

Task 1: PIN retry counters (spec page 28)

- Select MF/0016
- With READ RECORD read PIN1, PIN2, PUK records (records 0x01, 0x02, 0x03 respectively)
- Record's 6th byte will contain integer value of how many retries left

Task 1: Key usage counters (spec page 33)

- Select MF/EEEE/0013
- With READ RECORD read sign and auth key records (records 0x01 and 0x03 respectively)
- Record 13th, 14th and 15th bytes joined together contain 3 byte (Big-Endian) integer counter that describes how many times key may be used
  - Initial value 0xFFFFFF (i.e., key may be used 16 million times)
  - 3 byte integer can be calculated by $value = (13th << 16) \mid (14th << 8) \mid 15th$
    - Calculation example given in EstEID spec is wrong

Task 2

Implement utility that downloads authentication and digital signature certificates stored on ID card.

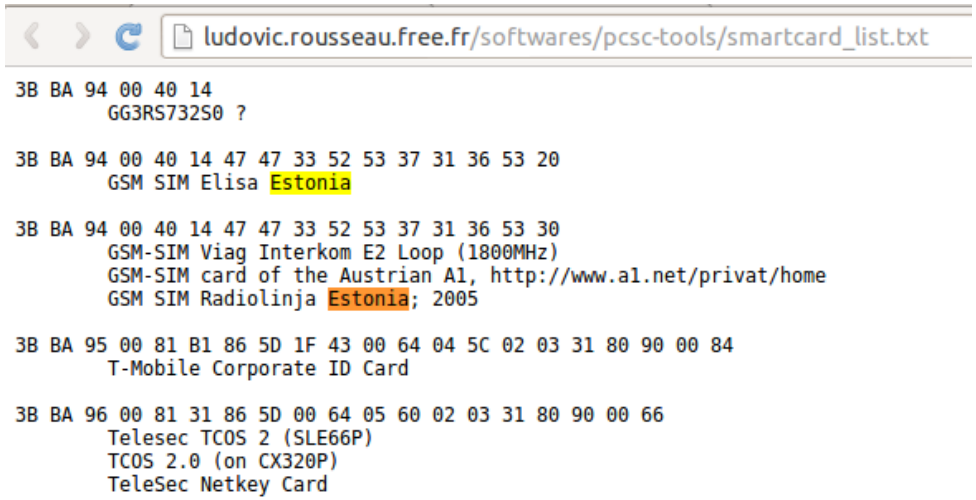
```
$ ./esteid_getcert.py --cert auth --out auth.pem
[+] Selected reader: Gemalto PC Twin Reader 00 00
[+] EstEID v3.5 (10.2014) cold (eID)
[=] Retrieving auth certificate...
[+] Certificate size: 1253 bytes
[+] Certificate stored in auth.pem
$ openssl x509 -in auth.pem -text | grep O=ESTEID
        Subject: C=EE, O=ESTEID, OU=authentication, CN=...

$ ./esteid_getcert.py --cert sign --out sign.pem
[+] Selected reader: Gemalto PC Twin Reader 00 00
[+] EstEID v3.5 (10.2014) cold (eID)
[=] Retrieving sign certificate...
[+] Certificate size: 1187 bytes
[+] Certificate stored in sign.pem
$ openssl x509 -in sign.pem -text | grep O=ESTEID
        Subject: C=EE, O=ESTEID, OU=digital signature, CN=...

$ wget https://sk.ee/upload/files/ESTEID-SK_2011.pem.crt
$ wget https://sk.ee/upload/files/ESTEID-SK_2015.pem.crt
$ openssl verify -CAfile ESTEID-SK_2011.pem.crt sign.pem
sign.pem: OK
$ openssl verify -CAfile ESTEID-SK_2015.pem.crt sign.pem
```

Put your output from these commands in `esteid_getcert.out`!

Task 2: Retrieve certificate (spec page 35)

- Select MF/EEEE/AACE (authentication certificate)
- Select MF/EEEE/DDCE (digital signature certificate)
- Certificate is stored in a DER form with garbage appended in a transparent file which is of fixed size
  - With READ BINARY (spec page 137) read first 10 bytes of certificate
  - Calculate certificate length by parsing length field of certificate ASN.1 SEQUENCE structure
- Read whole certificate (in a loop) using READ BINARY
  - On one READ BINARY only 0xFF bytes can be read
  - Offset must be specified as two byte integer specifying most significant byte in P1 and least significant byte in P2
  - Two byte integer can be split into [MSByte, LSByte] by `[i/256, i%256]` or by bit operations `[i>>8, i&0xFF]` or by `int_to_bytestring(i, 2)[0]`, `int_to_bytestring(i, 2)[1]`

ATR Collecting Party




```
$ pcsc_scan
Reader 0: Gemalto PC Twin Reader 00 00
  Card state: Card inserted,
  ATR: 3B 6C 00 00 0A 0B 0C 0A 0B 0C 0A 0B 0C 0A 0B 0C
Possibly identified card (using /home/user/.cache/smartcard_list.txt):
        NONE
Your card is not present in the database.
Please submit your unknown card at:
http://smartcard-atr.appspot.com/parse?ATR=3B6C00000A0B0C0A0B0C0A0B0C0A0B0C
```

e-Passports

- Contactless smart card chip
- Contains information printed on data page + fingerprints
- Data digitally signed by country signing CA
- Can be read only using the key encoded in MRZ
- Possibility for automated border clearance or “E-gates”