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 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:
  
  \([\text{CLA}] [\text{INS}] [\text{P1}] [\text{P2}] [L_c] [C_{data}] \ldots [L_e]\)
  
  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:
  
  \([R_{data}] \ldots [\text{SW1}] [\text{SW2}]\)
  
  Data (0 ... 256 bytes) + status word (2 bytes)
  
  62 00
  
  45 53 54 90 00
Electrical Transmission Protocols (T=0/T=1)

T=0 (byte-oriented protocol):

- Simplicity and minimal memory requirements
- 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 in case the terminal sends data APDU (or APDU with incorrect $L_e$ value):
    1. Card responds with SW: 61 XX
    2. Terminal sends GET RESPONSE command: 00 C0 00 00 XX
    3. Card returns XX bytes + SW
Electrical Transmission Protocols (T=0/T=1)

T=1 (block-oriented protocol):

- APDUs are encapsulated in blocks forming TPDUs (Transport Protocol Data Units)

<table>
<thead>
<tr>
<th>IFD</th>
<th>ICC</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAD</td>
<td>PCB</td>
</tr>
<tr>
<td>NAD</td>
<td>PCB</td>
</tr>
</tbody>
</table>

- 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).
# Standard Commands

<table>
<thead>
<tr>
<th>ClaiIns P1 P2</th>
<th>Lc Send Data</th>
<th>Le Recv 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>
<td></td>
</tr>
<tr>
<td>84 16 00 00 xx MAC</td>
<td>VSDC</td>
<td>CARD BLOCK</td>
<td></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>
<td></td>
</tr>
<tr>
<td>00 82 00 xx 06 Manual</td>
<td>GEMPLUS MPCOS-EMV</td>
<td>EXTERNAL AUTHENTICATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 84 xx 08 Rnd Num</td>
<td>GEMPLUS MPCOS-EMV</td>
<td>GET CHALLENGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 88 XX xx 0A Manual</td>
<td>GEMPLUS MPCOS-EMV</td>
<td>INTERNAL AUTHENTICATE</td>
<td></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>RUN GSM ALGORITHM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A0 A2 00 xx xx Pattern xx</td>
<td>3GPP TS 11.11</td>
<td>SEEK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 A4 04 00 xx AID 00</td>
<td>GlobalPlatform</td>
<td>SELECT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 A4 00 xx xx File ID</td>
<td></td>
<td>Name 00 Manual</td>
<td>VSDC</td>
<td>SELECT</td>
</tr>
<tr>
<td>A0 A4 00 00 02 File ID</td>
<td>3GPP TS 11.11</td>
<td>SELECT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A0 B0 xx xx xx</td>
<td>3GPP TS 11.11</td>
<td>READ BINARY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 B2 xx 00</td>
<td>VSDC</td>
<td>READ RECORD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A0 B2 xx xx xx</td>
<td>3GPP TS 11.11</td>
<td>READ RECORD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 C0 1C Key Info GlobalPlatform</td>
<td>GET RESPONSE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A0 C0 00 00 xx</td>
<td>3GPP TS 11.11</td>
<td>GET RESPONSE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80 CA xx xx xx</td>
<td>VSDC</td>
<td>GET DATA</td>
<td></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>
<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>
<td></td>
</tr>
<tr>
<td>00 DA xx xx xx Data</td>
<td>VSDC</td>
<td>PUT DATA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 DC xx xx xx Data (and MAC)</td>
<td>VSDC</td>
<td>UPDATE RECORD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A0 DE 00 00 03 Data</td>
<td>3GPP TS 11.11</td>
<td>LOAD AoC(SICAP)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80 E0 xx xx xx FCI length</td>
<td>3GPP TS 11.11</td>
<td>CREATE FILE</td>
<td></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>
<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>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></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 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

- **Addressable 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.”

```
$ 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 signal on RST pin is given
Preparation: Hardware

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

- Plug the reader into the USB port
  - If using VirtualBox forward USB to guest Ubuntu

```bash
$ 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. 0z776 SmartCard Reader    --- DELL’s 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: 02 Micro Oz776 00 00
  1: Gemalto PC Twin Reader 01 00
  Reader 0: 02 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()
  ['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)
  >>> connection.getATR()
  Traceback (most recent call last):
    File "/usr/lib/python2.7/dist-packages/smartcard/pcsc/PCSCCardConnection.py", line 163, in getATR
      SCardGetErrorMessage(hresult))
  smartcard.Exceptions.CardConnectionException: Failed to get status: Card was removed.

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

There are several types of electronic ID cards:

EstEID specification in English (includes examples):
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.
### Table 2-1: EstEID card application versions’ properties

<table>
<thead>
<tr>
<th></th>
<th>v1.0 and v1.0 since 2006</th>
<th>v1.1 (DigiID)</th>
<th>v3.0 and v3.4 since 18.01.2011</th>
<th>v3.5 (since 10.2014)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation platform</td>
<td>MICARDO</td>
<td>Multos</td>
<td>Java Card</td>
<td></td>
</tr>
<tr>
<td>RSA module length</td>
<td>1024 bits</td>
<td>2048 bits</td>
<td>1024, 1280, 1536, 1984, 2048 bits</td>
<td></td>
</tr>
<tr>
<td>Hash algorithm support</td>
<td>SHA-1, SHA-224</td>
<td>SHA-1, SHA-224, SHA-256, SHA-384, SHA-518</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECC module lengths</td>
<td>Not supported</td>
<td></td>
<td>The RSA key length of the same security level</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>160 bits</td>
<td>~1024 bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>192 bits</td>
<td>~1536 bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>224 bits</td>
<td>~2048 bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>256 bits</td>
<td>~3072 bits</td>
</tr>
<tr>
<td>PKCS#1 padding support</td>
<td></td>
<td></td>
<td>v1.5</td>
<td></td>
</tr>
<tr>
<td>Protocol support</td>
<td>T=0, T=1</td>
<td>T=0</td>
<td>T=0, T=1</td>
<td>T=0, T=1</td>
</tr>
</tbody>
</table>
EstEID file system (spec page 105)

path = MF/EEEE/0033
### APDU commands (spec page 117)

<table>
<thead>
<tr>
<th>Command</th>
<th>INS</th>
<th>Brief description</th>
<th>See page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANGE REFERENCE DATA</td>
<td>‘24’</td>
<td>Changes the password for cardholder authentication</td>
<td>112</td>
</tr>
<tr>
<td>CREATE FILE</td>
<td>‘E0’</td>
<td>Creates a directory or data field</td>
<td>115</td>
</tr>
<tr>
<td>EXTERNAL AUTHENTICATE / MUTUAL AUTHENTICATE</td>
<td>‘82’</td>
<td>Authenticates the external world / external world and chip card</td>
<td>116</td>
</tr>
<tr>
<td>GENERATE PUBLIC KEY PAIR</td>
<td>‘46’</td>
<td>Generates the public and the private part of a RSA key pair.</td>
<td>118</td>
</tr>
<tr>
<td>GET CHALLENGE</td>
<td>‘84’</td>
<td>Generates and outputs a random number</td>
<td>120</td>
</tr>
<tr>
<td>GET RESPONSE</td>
<td>‘C0’</td>
<td>Reads out the response data (T=0)</td>
<td>121</td>
</tr>
<tr>
<td>INTERNAL AUTHENTICATE</td>
<td>‘88’</td>
<td>Authenticates the chip card or application</td>
<td>112</td>
</tr>
<tr>
<td>MANAGE SECURITY ENVIRONMENT</td>
<td>‘22’</td>
<td>Passes on key references, random numbers, and data to be used for key derivation</td>
<td>123</td>
</tr>
<tr>
<td>PERFORM SECURITY OPERATION</td>
<td>‘2A’</td>
<td>Various functions using symmetrical and asymmetrical keys: 125</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- COMPUTE DIGITAL SIGNATURE</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- DECIPHER</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- HASH</td>
<td>129</td>
</tr>
<tr>
<td>READ BINARY</td>
<td>‘B0’</td>
<td>Reads from a transparent data field</td>
<td>131</td>
</tr>
<tr>
<td>READ RECORD</td>
<td>‘B2’</td>
<td>Reads from a formatted data field</td>
<td>133</td>
</tr>
<tr>
<td>RESET RETRY COUNTER</td>
<td>‘2C’</td>
<td>Resets the counter of failed attempts</td>
<td>134</td>
</tr>
<tr>
<td>SELECT FILE</td>
<td>‘A4’</td>
<td>Selects a directory or data field</td>
<td>136</td>
</tr>
<tr>
<td>UPDATE BINARY</td>
<td>‘D6’</td>
<td>Modifies a transparent data field</td>
<td>137</td>
</tr>
<tr>
<td>UPDATE RECORD</td>
<td>‘DC’</td>
<td>Modifies a formatted data field</td>
<td>139</td>
</tr>
<tr>
<td>VERIFY</td>
<td>‘20’</td>
<td>Authenticates the cardholder</td>
<td>140</td>
</tr>
</tbody>
</table>
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 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,0x45,0xFE,0x65,...]:
    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, 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 reads and displays personal data file, PIN retry and key usage counters from 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
  [6]Birth date: 05.08.1986
  [7]Personal identification code: 38608050013
  [9]Expiry date: 27.08.2020
  [10]Place of birth: LĀTI / 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:

```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
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 are 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 0xFFFFFFFF (i.e., key may be used 16 million times)
  - 3 byte integer can be calculated by
    \[ \text{value} = (13th << 16) | (14th << 8) | 15th \]
    - 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=...

$ 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/EEEEE/AACE (authentication certificate)
- Select MF/EEEEE/DDCE (digital signature certificate)
- 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 0xFF 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 can be split into [MSByte, LSByte] by bitwise operations [(i>>8), i&0xFF]
  - Make sure that only the minimum number of bytes required are sent to/from the card
e-Passports

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