Smart cards – basic principles

a. *ISO norms – ISO7816-X*

ISO7816-1 specifies the physical characteristics of integrated circuit like the limits to X-rays, UV, electromagnetic field, ambient temperature etc. Additionally, properties of smart card in flexion and robustness of contacts are specified. Important mainly for card manufactures.

ISO 7816-2 defines size, location and function of the card contacts. Vcc – power supply, RST – card reset, CLK – external clock signal, GND – ground, Vpp – programming power supply for older types of EEPROM (not used now), I/O – data communication, 2 contacts reserved for future use.

ISO7816-3 specifies communication protocol between smart card and reader on the level of the electrical signals.

- Protocol T=0 – byte-oriented protocol. Older than T1, designed for maximal simplicity and minimal memory requirements. Error detection only on parit bits level. Used in GSM cards.
- Protocol T=1 – asynchronous, half-duplex, block-oriented. Support layers separation (transport layer in OSI model).

ISO7816-4 specifies:

- Content of messages, commands and responses as are transported to card and back.
- Structure and content of historical bytes send as response after RESET command (ATR).
- Methods for accessing files and data on the card and algorithms offered by card.
- Methods for the secure messaging.

b. *APDU (Application Protocol Data Unit)*

APDU is basic logical communication datagram, which allows to carry up to ~260 bytes of data and contains header with possibility to specify target application on smart card which should process given APDU.

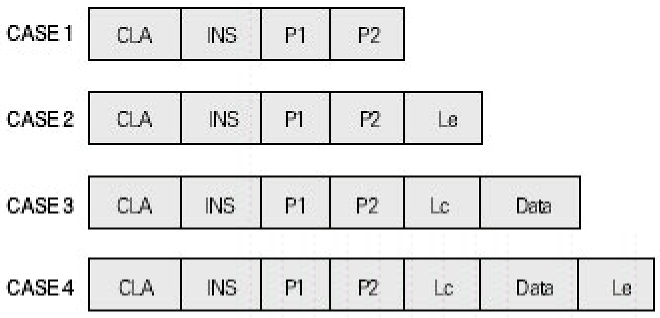

figure 1 - APDU types

CLA – instruction class, INS – instruction number, P1, P2 – optional data, Lc – length of incoming data, Le – length of the expected output data.

Communication with smart cards

a. PC/SC for Windows, PC/SC-Lite for Linux

The PC/SC Specification builds upon existing industry smart card standards - ISO 7816 and EMV - and compliments them by defining low-level device interfaces and device-independent application APIs as well as resource management, to allow multiple applications to share smart card devices attached to a system. See picture figure 2 for overview.

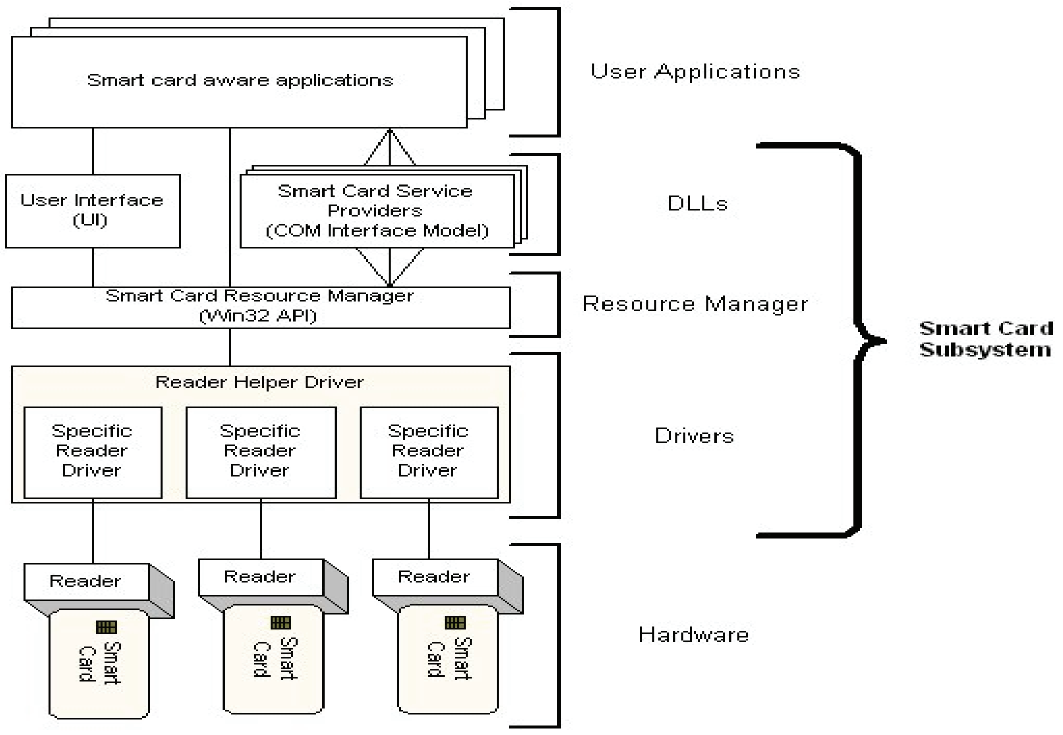

figure 2 – Windows PC/SC architecture overview

Sending and receiving APDU commands in Windows
There is the Win32 API for communicating with smart cards within Windows platform in form *SCardXXX*. Similar implementation for Linux is developed under Muscle project as PC/SC Lite API. We will work within Windows platform. Following functions will be used:

- SCardEstablishContext
- SCardListReaders
- SCardConnect
- SCardReconnect
- SCardDisconnect
- SCardReleaseContext
- SCardTransmit

```
void CardConnect(APDU apdu) {
      SCARDCONTEXT cardContext = NULL;
      SCARDHANDLE   hCard;                // OPENED SESSION HANDLE
      DWORD             scProtocol;
      char*                 readers = NULL;

      // ESTABLISHING CONTEXT OF CARD DATABASE TO BE SEARCHED WITHIN
      SCardEstablishContext(SCARD_SCOPE_USER,0,0,&cardContext);

      // LIST AVAILABLE READERS (readers NULL SEPARATED ARRAY)
      SCardListReaders(cardContext, NULL, (char *) &readers, &len));

      // … PARSE AND SELECT ONE READER INTO targetCard

      // CONNECT TO CARD WITH NAME targetCard
      SCardConnect(m_cardContext, targetCard, SCARD_SHARE_EXCLUSIVE, SCARD_PROTOCOL_T0 |
         SCARD_PROTOCOL_T1,  &hCard, &scProtocol));

      // … WORK WITH CARD

      // RESET CARD (POWER IS TURNED OFF AND THEN ON)
      // CARD IS IN DEFAULT STATE (NO SELECTED APPLETS…)
      SCardReconnect(m_hCard, SCARD_SHARE_EXCLUSIVE, SCARD_PROTOCOL_T0 |
         SCARD_PROTOCOL_T1, SCARD_UNPOWER_CARD, &m_scProtocol));

      // DISCONNECT FROM CARD AND RELEASE CONTEXT
      SCardDisconnect(m_hCard, SCARD_LEAVE_CARD);
      SCardReleaseContext(cardContext);
}
```

Function *CardConnect()* establish context, list all available readers, connect to selected smart card, reconnect and finally release connection to smart card.

High level function *ExchangeAPDU()* send APDU to smart card and handle situation when some output data are prepared on smart card and should be received. Specific APDU with CLA=0x00 and INS=0xC0 is used to obtain response data. Internally, *TransmitAPDU()* function is called to directly exchange one single apdu command.

```
  void TransmitAPDU(APDU pAPDU) {
    DWORD           outLen = pAPDU->le;
    BYTE            sendData[260];
    BYTE            responseData[260];

    // CLEAR SEND AND RESPONSE STRUCTURES
    pAPDU->le = 0;
    memset(sendData, 0, sizeof(sendData));
    memset(responseData, 0, sizeof(responseData));

    // TRANSFORM APDU STRUCTURE INTO ARRAY
    sendData[0] = pAPDU->cla;
    sendData[1] = pAPDU->ins;
    sendData[2] = pAPDU->p1;
    sendData[3] = pAPDU->p2;
    sendData[4] = pAPDU->lc;
    memcpy(sendData + 5, pAPDU->DataIn, pAPDU->lc);

     outLen = 4;
     // SEND APDU USING SCardTransmit FUNCTION ACCORDING TO TRANSMISSION PROTOCOL
     switch (m_scProtocol) {
         case SCARD_PROTOCOL_T0: SCardTransmit(m_hCard, SCARD_PCI_T0, sendData, sendData[4] + 5,
                 NULL, responseData, &outLen)); break;
         case SCARD_PROTOCOL_T1: SCardTransmit(m_hCard, CARD_PCI_T1,
                 sendData, sendData[4] + 5, NULL, responseData, &outLen)); break;
     }

     // COPY SOFTWARE STATUS
     ((BYTE*) &(pAPDU->sw))[0] = responseData[1];
     ((BYTE*) &(pAPDU->sw))[1] = responseData[0];

      // RECEIVE RESPONSE DATA, IF ANY
      if (((pAPDU->sw & 0xFF00) == SW_BYTES_REMAINING_00) ||
          ((pAPDU->sw & 0xFF00) == SW_CORRECT_LENGTH_00)) {
            // GET DATA APDU (FORM SPECIAL APDU FOR RECEIVING DATA)
            sendData[0] = 0xC0;
            sendData[1] = 0xC0;
            sendData[2] = 0x00;
            sendData[3] = 0x00;
            sendData[4] = LOWBYTE(pAPDU->sw);

            outLen = sendData[4] + 2;   // DATA OUT + STATUS

            // … SEND APDU (SEE ABOVE)

           // COPY RECEIVED DATA
           memcpy(pAPDU->DataOut, responseData, outLen - 2);
            pAPDU->le = outLen - 2;
            ((BYTE*) &(pAPDU->sw))[0] = responseData[outLen - 1];   // LAST BYTE
            ((BYTE*) &(pAPDU->sw))[1] = responseData[outLen - 2];   // PRE LAST BYTE
      }
}
```

Global Platform

Publicly available specifications [GP03] for smart card managements covering issues of smart card life cycles, installation of applets, remote card management and secure communication between smart card and user application.

a. Card Manager and Security domain

The Card Manager is the card component responsible for all card administration and card system service functions:

Command Dispatch:
- Application selection
- (Optional) Logical channel management
- Command dispatching

Card Content Management
- Content verification
- Content loading
- Content installation
- Content removal

Security Management
- Security Domain locking
- Application locking
- Card locking
- Card termination
- Application privilege usage
- Security Domain privilege usage
- Tracing and event logging

The Issuer Security Domain is card component (more independent security domains can be present) that contains keys that the Card Issuer uses in support of cryptographic operations for the Card Issuer's Applications. These Security Domains are privileged applications established on a GlobalPlatform card to represent Application Providers who require a level of key separation from the Card Issuer. Security domain is used when some card management operation is required after the card issued to card holder (e.g. upload and installation of applet, card locking, …). Keys carried by this Security domain are used to verify authenticity and integrity of request (Load File) and to provide confidentiality of transfered data. A Load File may contain one or more DAP (Data Authentication Pattern) Blocks that allow an entity other than the loading entity to verify the authenticity and the integrity of the Load File Data Block.

b. Smart card life cycles

The smart card passes various logical life cycle states between manufacture and final destruction. These life cycle states define, which operations can be performed with the card. The following card Life Cycle States shall apply:

1. OP_READY –card is ready for uploading of key diversification data, any application and issuer specific structures.
2. INITIALIZED – card is fully prepared but not yet issued to card holder.
3. SECURED – card is issued to card holder. Card management is possible only throw Security domain in secure sense (installation of signed applets etc.).
4. CARD_LOCKED – card is locked due to some security policy and no data management can be performed. Card can be locked by Security domain and later unlocked as well (switch back to SECURED state).
5. TERMINATED – card is logically "destroyed" due to card expiration or detection of the severe security thread.

The card Life Cycle States OP_READY and INITIALIZED are intended for use during the Pre-Issuance phases of the card's life. The states SECURED, CARD_LOCKED and TERMINATED are intended for use during the Post-Issuance phase of the card although it is possible to terminate the card at any point during its life.

c. GP APDU commands

Following APDU commands are defined to manage content of the smart card:

- **DELETE** – delete uniquely identifiable object (e.g. JavaCard applet)
- **STORE_DATA** – upload content of single data object
- **GET_DATA -** used to retrieve a single data object
- **SET_STATUS** – set Life Cycle status
- **GET_STATUS** – return Life Cycle status
- **INSTALL** – initiate installation, typically (JavaCard) applet
- **LOAD** – upload file from PC to smart card, e.g. JavaCard cap file
- **PUT_KEY** – update value of specified key

d. Secure Messaging – Secure channel protocol

Secure messaging stands for the process, which should lead to mutual authentication of the smart card and PC and optional creation of the authenticated encrypted tunnel between smart card and PC. Details of the process are described in Open Platform specifications [GP03].

Mutual authentication and computation of session keys consist from two phases (two APDU commands exchanged with smart card):

1. INITIALIZE UPDATE
2. EXTERNAL AUTHENTICATE

First phase (INITIALIZE UPDATE): PC application sends 8-bytes block of random data to smart card (called host random or host challenge). Smart card generates its own 8-bytes random block (card random/challenge). Using host and card challenge forms derivation data and encrypt them using card specific static encryption key (static ENC key - created during key diversification process). Result of this operation is the session key (session ENC key).

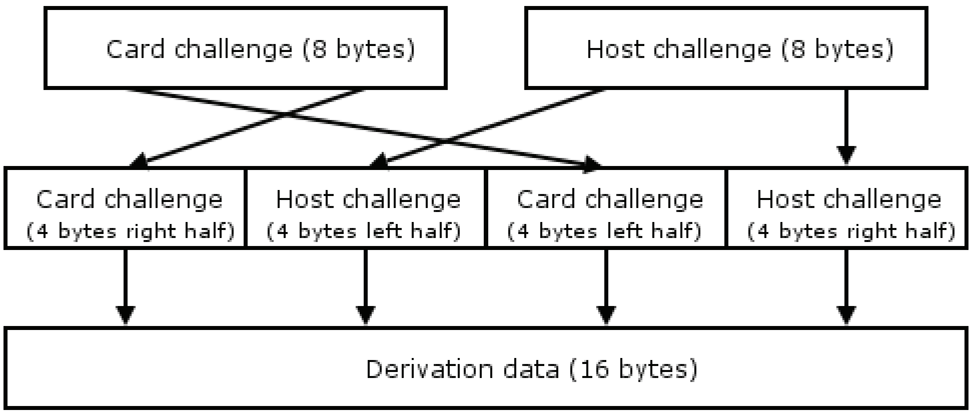
Card challenge (8 bytes)
Host challenge (8 bytes)
Card challenge (4 bytes right half)
Host challenge (4 bytes left half)
Card challenge (4 bytes left half)
Host challenge (4 bytes right half)
Derivation data (16 bytes)

figure 3 Generation of the derivation data

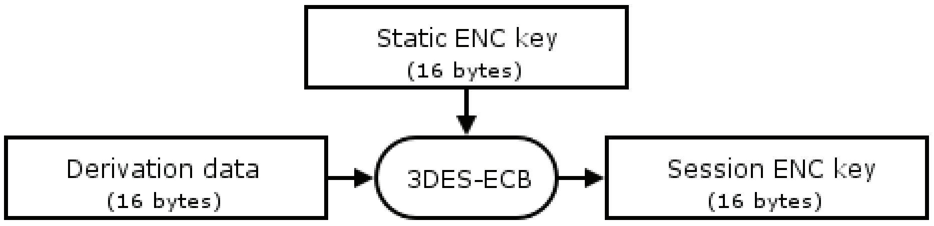
Static ENC key (16 bytes)
Derivation data (16 bytes)
3DES-ECB
Session ENC key (16 bytes)

figure 4 Generation of session ENC key

Session key is used for computation of the authentication *card cryptogram*. *Cryptogram* is computed using MAC operation. Input data are concatenated *card challenge* and *host challenge* padded by the block ('80 00 00 00 00 00 00 00'). Cryptogram is send together with card challenge back to PC.

Second phase (EXTERNAL AUTHENTICATE): PC application checks *card cryptogram* send by card and computes its own cryptogram (*host cryptogram*). Algorithm for the host cryptogram computation is again based on 3DES in CBC mode.

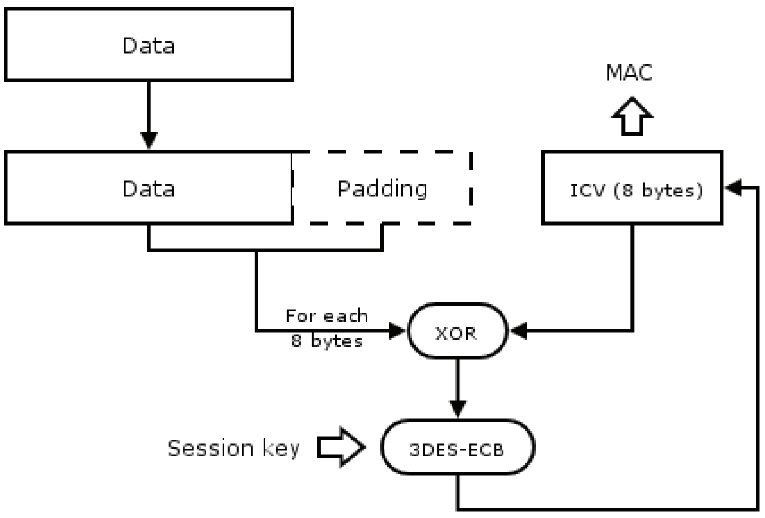

figure 5 Algorithm for MAC computation

In case of the equivalence of the received and computed *card cryptograms, host cryptogram* is send back to card together with MAC of whole APDU command (EXTERNAL AUTHENTICATE).

Algorithm for host cryptogram is same with swapped host and card challenge.

Session encryption and MAC keys are then used to create confidential authenticated tunnel between smart card and PC application for all subsequent APDU commands.

Homework

The goal of this assigment is to obtain the account number from communication exchanged between PC (LabakAPDUSecret.exe) and smart card.

When the LabakAPDUSecret.exe is executed alone, it will contact first available reader in system and try to retrieve account number from smart card in pseudo-secure way. No block cipher algorithm is use (e.g., DES or AES) and application do not have any pre-shared secert with smart card.

Hints:
- Copy "fake" winscard.dll library and original one renamed to 'original.dll' into LabakAPDUSecret.exe directory. After LabakAPDUSecret.exe execution, winscard.txt will contain the log of communication. Check this out and try to analyze it and retrieve the account number.
- You may use also other techniques like reverse-engineering of the binary code of LabakAPDUSecret.exe, LabakSecret.jar etc.

Submit:
- Whole account number and the short description of the process used to retrieve it.
- Description of the apdu commands exchanged between PC and smart card, especially: what CLA, INS, P1, P2 and LC was used in header, how long are the payload data and what is its "meaning" (what they are good for) and which sequence they pass between PC and SC.
- Deadline is 30.4.2007{10 points}

References

[GP03]	GlobalPlatform, http://www.globalplatform.org/showpage.asp?code=specifications
[JC221]	Java Card specification 2.2.1 http://java.sun.com/products/javacard/
[JPCSC]	JPC/SC API http://www.linuxnet.com/middleware/files/jpcsc-0.8.0-src.zip