

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 parity 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.

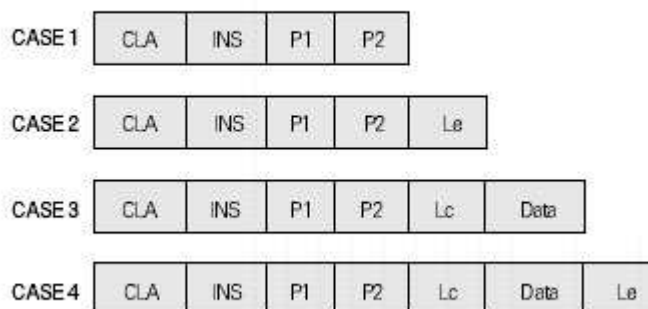


figure 1 - APDU types

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 complements 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.

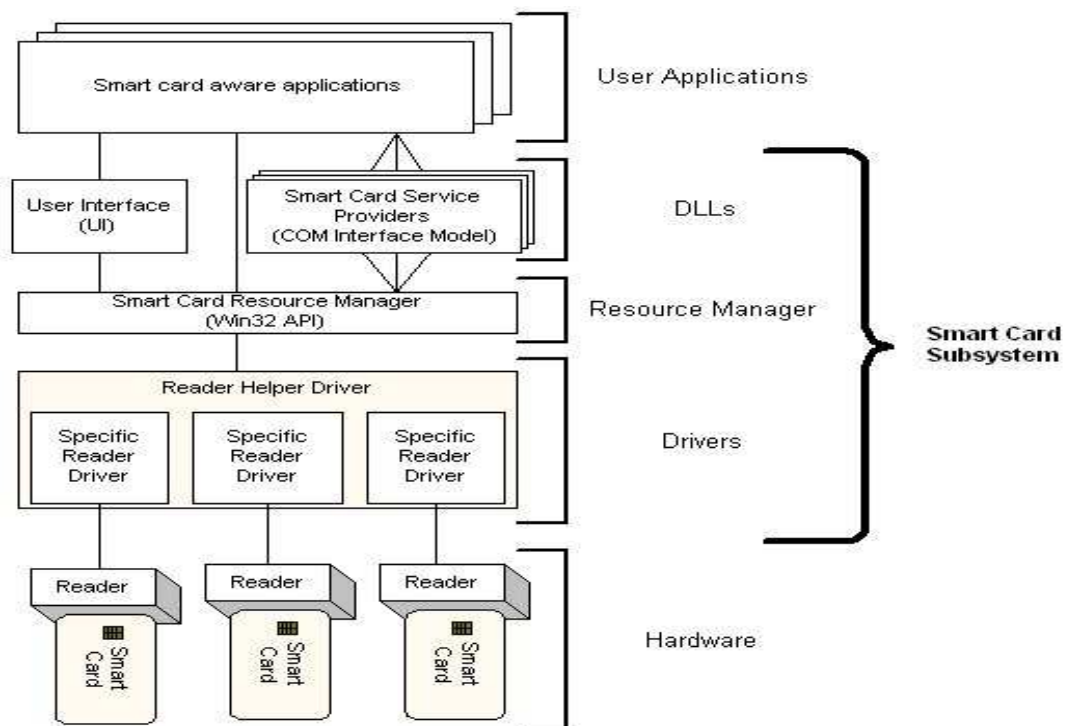


figure 2 – Windows PC/SC architecture overview

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 ExchangeAPDU(APDU pAPDU) {
    // CLEAR SOFTWARE RETURN STATUS
    pAPDU->sw = SW_NO_ERROR;

    // SEND APDU
    if ((status = TransmitAPDU(pAPDU)) == STAT_OK) {
        // CHECK FOR SOFTWARE ERROR
        if (pAPDU->sw == SW_NO_ERROR) {
            // NO SOFTWARE ERROR
        }
        else {
            // CHECK FOR 'RESPONSE DATA AVAILABLE' STATUS
            if ((pAPDU->sw & 0xFF00) == SW_BYTES_REMAINING_00) {
                if (pAPDU->lc == 0x00) {
                    // SYSTEM CALL TYPE TO OBTAIN RESPONSE OUTPUT DATA)
                    // INSUFFICIENT OUTPUT DATA LENGTH
                    BYTE realLen = LOWBYTE(pAPDU->sw);
                    pAPDU->le = realLen;

                    // NEW APDU WITH REQUIRED OUTPUT DATA LENGTH
                    status = ExchangeAPDU(pAPDU);
                }
                else {
                    // USER CALL TYPE, OBTAIN RESPONSE DATA
                    BYTE realLen = LOWBYTE(pAPDU->sw);
                    CARDAPDU apdu;

                    // PREPARE SYSTEM APDU FOR RECIEVE RESPONSE OUTPUT DATA
                    apdu.cla = 0x00;
                    apdu.ins = 0xC0;
                    apdu.p1 = 0x00;
                    apdu.p2 = 0x00;
                    apdu.lc = 0x00;
                    apdu.le = realLen;

                    if ((status = ExchangeAPDU(&apdu)) == STAT_OK) {
                        // COPY RECIEVED APDU
                        memcpy(pAPDU->DataOut, apdu.DataOut, apdu.le);

                        pAPDU->le = apdu.le;
                    }
                }
            }
            else {
                // CHECK FOR 'CORRECT DATA LENGTH' STATUS
                if ((pAPDU->sw & 0xFF00) == SW_CORRECT_LENGTH_00) {
                    pAPDU->le = LOWBYTE(pAPDU->sw);
                }
                else {
                    // SOFTWARE ERROR OCCURED
                    status = TranslateISO7816Error(pAPDU->sw);
                }
            }
        }
    }
}

```

```

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
    }
}

```

Homework

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

When the LabakAPDU Secret.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 used (e.g., DES or AES) and application does not have any pre-shared secret with smart card.

Hints:

- Copy "fake" winscard.dll library and original one renamed to 'original.dll' into LabakAPDU Secret.exe directory. After LabakAPDU Secret.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 LabakAPDU Secret.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 13.10.2008 {10 points}

References

- [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>