EMBEDDED SECURITY SYSTEMS 2016

Mirosław Kutyłowski

&grades: 40% lecture, 60% lab

exam, no tests during the course, exam in English unless...

short problems, skills examined not knowledge

lower bound: 40% 3, 50% 3.5, 60% 4, 70 % 4.5 80% 5.0

Objectives

presentation of architecture, limitations and functionalities of embedded systems used in security area C2 developing programming skills concerning cryptographic smart cards and FPGA

---

1. smart cards ≈6 hours
2. security printing ≈2 hours
3. telecommunication systems ≈2 hours
4. HSM, TPM, remote attestation ≈4 hours
5. FPGA ≈2
6. sensor systems ≈2 hours
7. RFID tags ≈4 hours
8. CUDA and parallel programming ≈4 hours
9. smart meters ≈2 hours

---

1. SMART CARDS

cards of no-smart solutions:

- **embossed** - credit cards: reading does not require electricity, elementary protection only
- **magnetic:** ≈1000 bits, 3 tracks, track 1: 79 6bit chars, track 2: 40 4bit chars, track 3: 107 4-bit chars, limited density (movement in reader against the head), standard data on tracks 1 2, track 3 for read-write, no physical protection, cheap readers, accidental erasure by a nearby magnet, horror as ATM cards (obsolete in EU but still in use in some countries)

  data stored in financial cards: **Track 1**, Format B:

  - start character
  - format character
  - PAN - primary account number — up to 19 characters. e.g. credit card number
  - separator character
  - name: 2 to 26 characters

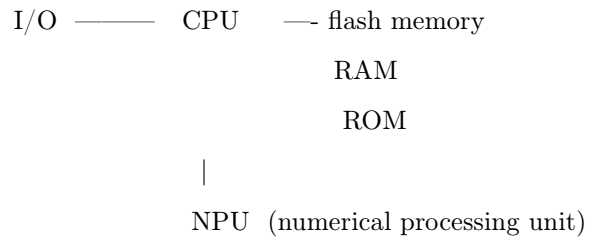
- separator character ('^')
- expirationYYMM.
- service code 3 characters
- discretionary data: may include Pin Verification Key Indicator (PVKI, 1 character), PIN Verification Value (PVV, 4 characters), Card Verification Value or Card Verification Code (CVV or CVC, 3 characters)
- end sentinel (generally '?')
- one character validity character (over other data on the track).

smart cards: classification

- memory cards (with security logic and without)

  usually memory: non-volatile EEPROM, serial communication, control logic: where you can write. Cheap. E.g. prepayed telephone cards
- processor: with coprocessor or without (as bad as: RSA in 20 minutes)
- contact or wireless

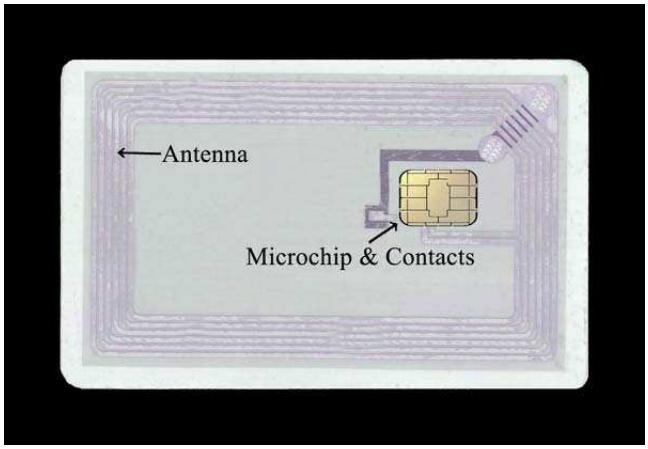
processor cards:



Contactless cards:

- energy: inductive (low!)
- small range (typicaly 10 cm)
- a reader may activate it from distance
- response with low energy, recognizable from a short distance only
- memory: kilobytes
- well sealed against corrosion
- main parts:
  - antenna (most area of the card)

- electronic part: modulation, demodulation, clock generator, voltage regulation, reset generation
- interface between RF interface and memory chip
- access logic
- application data: EEPROM, ROM



contacts:

- 8 fields, normally 6 used (2 for future applications), places for contacts strictly determined in standard
- 8 connections, 2 auxiliary and can be omitted or used e.g. for USB

  connections:

  - C1: Vcc voltage supply
  - C2: RST reset
  - C3 CLK clock
  - C4 AUX1
  - C5 GDN ground
  - C6 SPU standard or proprietary use (SWP)
  - C7 I/O
  - C8 AUX2

  |C1 C5 |
  |C2 C6|
  |C3 C7|
  |C4 C8|

- easy to destroy
- corrosion, mechanical scratches, not for intensive use

security tokens:

- type 1: USB tokens – contact interface like in USB, insert into a port after breaking out of a card
- type 2: small display (eg. 4 digits). input also possible: e.g. a card with 2 buttons (each one side of the card), battery inserted

optical:

- writing technique - like CD (linear and not circular)
- area designated field according to standards, may leave place for contact interface of a chip and magnetic strip, less place for graphical part on the card
- high volume ($\approx$6MB storage)
- redundancy via error correcting codes, therefore not easy to destroy information, particularly against burst errors
- usage: e.g. border control cards (Mexico-USA), Italian personal ID card, but market success limited

Physical properties of smart cards:

standard format: 85.6x54 mm (ID-1), other formats for SIM cards (in larger ID-1 cards with stamping),

functional components:

magnetic stripe, signature panel, embossing, imprinting of personal data by laser beam, hologram, security printing, invisible authentication feature, chip

important parameters of card body (properties defined by the standard):

- mechanical robustness (card and contacts)
- temperature resistance
- surface profile
- electrostatic discharge
- electromagnetic susceptibility
- ultraviolet radiation resistance
- X-ray radiation resistance

Material: trade-off with different properties

PVC: polivinyl chloride, credit cards, cheap, problems with low and high temperatures, injection molding impossible, lifetime 2 years, cost factor: 1

ABS: a common thermoplastic polymer, SIM cards, termally stable up to 100 C, laser engraving poor, lifetime 3 years, cost factor: 2

PC: polycarbonate, ID cards, durable, 160 C, problems with hot stamping, lifetime 5 years, cost factor: 7, low scratch resistance,

PET: health cards, mechanical: very good resistance, lifetime 3 years, cost factor: 2.5

Chip modules:

- the chip too fragile and too thick to be laminated on the surface. the chip is inserted inside
- electrical connections are the problem,
  - automatic bonding of the gold wires to the back of contacts with ultrasonic welding
  - or mechanically connected to the back of the module
- Chip-on flex modules, stages of production:
  - tape with empty modules
  - gluing the dice into modules
  - bonding the dice
  - encapsulating the dice
- lead-frame: chip produced together with contacts and the simply inserted by a robot into the card body and glued

Electrical properties:

- voltage is a problem: 3V for SIM cards (batteries for smartphones weight optimization), 5V for smart cards, higher voltage needed for EEPROM erasure. charge pumps must be applied

- max 60 mA for 5V, max ambient temperature 50 degrees**,** 350 $\boldsymbol{\mu A}$ per megaherz, power consumption too low to cause overheating, power reduction e.g. for SIM in different phases of activity (low if the phone is not transmitting and using cryptoprocesor)
- contact C6 used to be for EEPROM erasing but not needed anymore (instead used for Single Wire Protocol)
- no internal clock supply (this is a potential risk: adversary may increase the clock frequency to create faults, fault cryptanalysis)
- problems with collisions on the I/O line (too high currents would destroy interface components)
- protection against out of range voltages, electrostatic charges, precisely defined activation and deactivation sequences: first ground, then voltage, then clock, warm reset when voltage increases on the reset line

Microcontrollers:

- area: manufacturing costs and durability (bending, torsion), typically 10mm^2, square shape
- must be integrated, "standard components" are not well suited due to the size of the resulting circuit,
- native designs are proprietary, even a crime to check the layout
- semiconductor technology -> density increases -> chip area drops . But some problems: error probability, necessity to decrease voltage, ...
- extremely high reliability needed. So behind the "state-of-the-art" which is frequently instable
- memory small (e.g. 100KB), a 8-bit processor ok for less than 64KB, then extensions, usually CISC (complex instruction set computer) - instruction over a number of steps, some based on RISC (reduced instruction set computer), also 32 bit processors that needed also for interpreter based architectures (Java Card)

MEMORY

Memory types:

non volatile:

EPROM - UV erasure, not suited for smart cards,

EEPROM - electrical erasure, cell capacitors, discharged state=0, charged state=1, erase state → non-erased: changes by single bit possible, non-erased-> erased: page or sector,

size per bit: 1.14 μm, up to 100.000-1.000.000 erasures, slow write operations: 2-10ms,

it is based on tunneling efect - if there are electrons on the floating gate then they prevent flow in the substrate,

data retention time limited - currently 10 years, problems with external influence (radiation) that may result in changing cell contents

flash - a different technique for writing: hot electron injection, electron to the floating gate come from the channel and not from tunelling effect

write time - low („flash") 30μs, erase like for EEPROM (tunneling efect) ,

size 0.47, 10.000-100.000 erasures„ lower voltage (12V) than EEPROM (17V),

NOR flash: free read of individual cells, but complicated circuits,

NAND flash: dense but reading full blocks, NAND used for storage due to large block size, NOR flash may be used for storing programs

ROM - connections broken – memory via a circuit, irreversible process - one disconnected never can be reconnected, lack of connection = 0, small: 0.54μm area size

volatile:

RAM - based on transistors and flip-flop, 1.87 μm area size (12.5 bigger than ROM), erasures - unlimited, write: 70ns

---

AUXILIARY UNITs:

UART - universal asynchronous receiver transmitter, software solutions would be too slow - problem: speed of communication versus clock frequency. Divisor: how many sycles are necessary to send 1 bit. Going from 372 down

USB – USB connection has rigid timing requirements, they cannot be guaranteed by the regular chip, 12 MB/s (Full Speed), CRC and buffers on the endpoints

SWP single wire protocol - communication between SIM and NFC controller concurrently with the regular I/O, data sent with voltage modulation and returned with current modulation- full duplex,

timer - a 16 bit counter (or 16 bit), used for timeout detection, watchdog for security reasons

Error detection:

1. **XOR** checksums
   - logical XOR of all bytes
2. **CRC** cyclic redundancy clock,
   - commonly used CRC16: $x^{16}+x^{12}+x^5+1$ (ISO),
   - code: remainder from dividing data by the CRC polynom
3. **Reed Solomon** codes,
   - $x = x_1...x_k$ is the sequence to be encoded
   - $p_x(a) = \sum_{i=1}^{k} x_i a^{i-1}$ polynomial over some finite field
   - $C(x) = p_x(a_1)p_x(a_2)...p_x(a_n)$ is the code of x, where a_i is the ith power of the root of degree n.
   - $C(x) = x \cdot A$, Vandermode matrix, row 1: 1....1, row 2: $a_1, ...a_n$, row 3: a_1^2, ...,a_n^2, and so on
   - properties: distance between the codewords: $n-k+1$ (this is optimal), since two polynomials of degree k may have only $k-1$ equal values
   - it can correct half of its bits

- commonly used CRC16: $x^{16} + x^{12} + x^5 + 1$ (ISO),

RNG temperature etc. hard to implement, an implementation: LFSR + reading its state by the processor at ranom moments

pragmatic solutions: PRNG (sometimes poor),

be aware that the algorithm implemented is not original one (e.g. DSA but DSA+LFSR+...),

PRNG: the next value derived with the key from the previous values.

- Round Robin- eg 12 values in a buffer
- testing - NIST tests, good for excluding biased/faulty generators, no security guarantees
- hardware Trojans: faults in the circuitry that are not changing the layout-wires, but e.g. the number of electrons in the substrate (invisible during the audit, but may be used to "break randomness" if the manufacturer knows what are the faulty places

Clock multiplication: external clock cannot have frequency over 5MHz. Internally we can increase it a few times with a multiplication circuit. Potentially: one could adjust the speed to adjust energy usage (problems with intereference of oscillators with the GSM, UMTS communication)

MMU: memory management unit for monitoring boundaries between the application programs (strict separation). must be tailored to the operating system of the chip

JAVA accelerator: approaches 1) dedicated hardware component, high speed but takes place, 2) native instructions for java

Symmetric crypto coprocessor: 75 microseconds per DES, 150 per 3DES

asymmetric coprocessor:

RSA up to 2048, problematic key generation, probabilistic time, RSA below 2048 bits is "disallowed" now by NIST

EC 160-256 bits: create DSA, random numbers problematic

hash functions: SHA, Keccak, SHA1 in PL

memory for keys:

- masterkey, derived keys, dynamic keys (session)
- PIN: master or deriving from master key, PIN updates in different memory, problems of nonuniformity of PIN (no leading zeroes, etc), subclasses where strategy gives higher chances

---

DATA ENCODING

- saving place is more important than universality and flexibility, properties: limited flexibility, low overhead, much better than XML
- **ASN**:

  Abstract Syntax notation,

ASN.1: primitive types (boolean, integer, octet string, bitstring), constructed data types, example:

```
SC_Controller::= SEQUENCE{
  Name IA5String,
  CPUType CPUPower,
  NPU Boolean,
  EEPROMSize INTEGER,
  RAMSize INTEGER,
  ROMSize INTEGER}
CPUPower::= ENUMERATED {
  8bit (8),
  16bit (16),
  32bit (32)}
```

- encoding via **TLV structures**: (Tag, Length,Value), tags for frequently used data types are in a standard,
- **BER -Basic Encoding Rules**, Distinguished Encoding Rules – subset of BER

  Some details:

  - tag:1-2 bytes, the first byte: b8, b7 define the class: universal, application, context-specific, private class, b6: data object primitive or constructed, b5-b1: tag code, if all ones then the second byte specifies the tag code
  - Length: 1-4 bytes:
    - 1 byte: 00 to 7F: encode length 0-127
    - 2 bytes: 1st byte 81, 2nd byte encodes length 0-255
    - 3 bytes: 1st byte 82, 2nd and 3rd bytes encode length 0-65535
    - ...
- **Data Compression (useful for images):**

  only very simple methods used as compression/decompression procedures might require too much space and it does not pay off to compress. Two main methods:

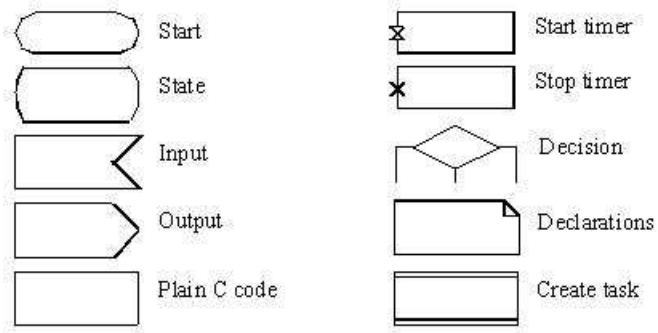
  - run-length encoding: blocks of zeroes and ones, each longer block encoded by its length after an escape symbol, short blocks encoded directly,
  - encodings like Huffman encoding: variable length encoding based on frequency of symbols. Only static methods used due to complexity of dynamic Huffman encoding:

    – take the expected probability of characters

    – for a character x try to assign a code of length $\log_2 \Pr(x)$

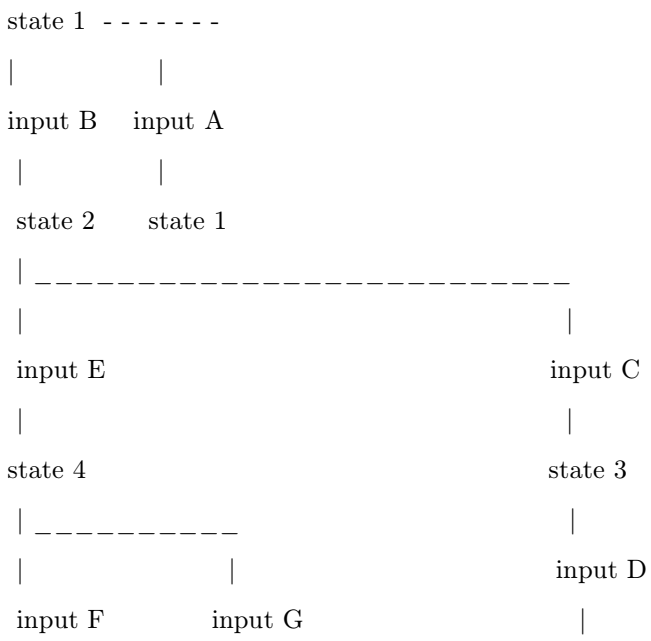
STATE MACHINES

- after activation and reset always **in the same initial state**
- in case of any trouble - **reset**
- state machine: transitions describing the events
- standard SDL notation:



and circle as a label (instead of C code from picture simple conditions)

- design rule: write an automaton, avoid cycles (except for reset) with a few short paths
- the input consists of (standard) instructions. Their execution automatically delivers response to the reader. the set of instructions is available in the card profile
- there are methods to design automata with compound states
- example:



| | | state 3

state 1 state 5

|

input H

|

state 5

A, C: operation SELECT

B: VERIFY

D: READ BINARY

E: GET CHALLENGE

F: all instuctions but EXTERNAL AUTHENTICATE

G: EXTERNAL AUTHENTICATE

H: SELECT/UPDATE BINARY

FILE MANAGEMENT

general:

- previously direct physical addressing
- still no man-machine interfaces, hexadecimal adressing, etc.
- the file itself contains all information about itself in a **header**, the rest is the **body**
- a header is not changed frequently (problems with EEPROM, flash) , so it is placed in a separate page
- the files are located in the memory according to security requirements, files are shared sometimes via shared headers
- **separation of applications via separation of directories**

lifecycle:

1. CREATE (might be with initial data or not)
2. ACTIVATE/DEACTIVATE – activation necessary before use (only a limited number of files might be active at a time!)
3. TERMINATE - permanently blocked, its memory is permanently inaccessible (secure option)
4. or: DELETE - memory recovered (insecure)

Types:

MF- Master file, main directory, implicitly selected after activation

DF - Dedicated Files, directories, hierarchy, always not deep (1-2 levels)

ADF - application dedicated files - not below MF

EF - elementary file, data necessary for application

Internal EF - hidden files used by operating system. They cannot be selected by applications.

File names

logical names due to transferability of programs

- File Identifier (FID), 2 bytes, some FIDs are reserved
- MF: FID is 3F00 (reserved value for MF)
- DF: FID, DF name (1 to 16 bytes, the first 5 for the name of DF, normally bytes 5-16 are AID - application ID: 5 bytes RID (registered identifier: A-international registrations, D-national registration, country code, application provider number), the rest is PIX (proprietary application identifier extensions)
- EF: FID (2 bytes), some reserved/predefined (e.g. GSM, student identity cards,...), SFI (short FID) - to be used in instructions values from 0 to 30, while 0 is the current EF

File selection

one opened for I/O

SELECT (explicitely) or implicit (by READ ..., UPDATE, ...)

File structures

- **transparent**: a single sequence of characters, no internal structure, reading file contents by specifying offset, instructions: READ BINARY, WRITE BINARY, UPDATE BINARY
- **linear fixed:** equal length records, reading by specyfing record number, instructions: READ RECORD, WRITE RECORD, UPDATE RECORD
- **linear variable:** frequently not supported, should be avoided, the same instructions as for linear fixed, but execution more complicated
- **cyclic:** a fixed number of records of a fixed size, access to the first, the last, the previous or the next record (in relation to the current record)

Access conditions:

- defined at file creation time and almost always unchanged, access conditions stored in the file header
- **state oriented conditions** (the current state is compared with the file access information) or **command oriented** (which commands have to be executed before accessing the file – risky in multiapplication cards)
- global security state (card, state of MF) and local security state (state of a file)
- some commands influence directly local security status: e.b LOCK, and REHABILITATE

STANDARD OPERATIONS

- each card implements some set, **but unnecessary commands are removed for saving space**
- **different collections of operations**: general, payment cards, telecommunication cards, over 20 standards, the most important are Global Platform Specification, Common Electronic Purse Specification (CEPS)
- arguments, return values

File commands:

CREATE FILE

DEACTIVATE FILE or INVALIDATE: locks file

ACTIVATE FILE or REHABILITATE: unlocks file

TERMINATE EF irreversible version of DEACTIVATE

TERMINATE DF

DELETE FILE – actually the same as TERMINATE, however it depends on the operating system whether the file is really deleted or only marked as deleted

TERMINATE CARD USAGE

SELECT - selects a file, or an upper DF

STATUS - shows selected file and it properties, otherwise no action

READ BINARY for transparent files, arguments': (number of bytes to be read, offset) return: data, return code

WRITE BINARY ... this is not a write instruction but the logical AND

UPDATE BINARY - this is the write operation in the standard meaning

ERASE BINARY (not necessarily physical erase)

READ RECORD, WRITE RECORD (like WRITE BINARY), UPDATE RECORD, **APPEND RECORD** (adding a new record at the end)

PUT DATA (structure of data objects, objects), GET DATA (number of data objects, tag) - direct access to TLV objects

SEEK (length of data, pattern, offset, mode (forward from the start, backwards from the start, forward from the next record, backwards from the previous record) returns record number

SEARCH - another standard, file can be checked

SEARCH BINARY- the same for transparent files

INCREASE, DECREASE – for cyclic files, operations on the counter (changing the current position)

EXECUTE – starts executable EF (some operating systems enable this)

PIN commands:

VERIFY PIN, VERIFY CHV – PIN verification, (CHV = Card holder Verification) , switch: global PIN or application specific (each application may have its own PIN)

CHANGE CHV – reset PIN

RESET RETRY COUNTER – with PUK, counter like in ATM cards enable only a limited number of PIN trials

UNBLOCK CHV (using PUK)

ENABLE VERIFICATION REQUIREMENT – e.g. GSM for PIN with SIM cards

DISABLE VERIFICATION REQUIREMENT – switch off the PIN verification

Security operations:

GET CHALLENGE - returns a random number

INTERNAL AUTHENTICATE - authentication of a card against a terminal

- the terinal sends a random number r and the key number (the key is shared with the card)
- the card returns: $X_{\mathrm{ICC}} := \mathrm{Enc}_{\mathrm{key}}(r)$
- the terminal tests the result

EXTERNAL AUTHENTICATE - authenticaton of the terminal via a key shared with the card

- the card sends a random number r to the terminal (via GET CHALLENGE)
- $X_{\mathrm{IFD}} = \mathrm{Enc}_K(r)$ sent to card, the card checks whether the result is correct (the key K is shared with the terminal)

MUTUAL AUTHENTICATE - based on a shared key

- GET DATA (data about chip number)
- GET CHALLENGE – $\mathrm{RND}_{\mathrm{ICC}}$ transfered to the terminal
- authentication:
  - the terminal generates $\mathrm{RND}_{\mathrm{IFD}}$, computes $X_{\mathrm{IFD}} = \mathrm{Enc}_K(\mathrm{RND}_{\mathrm{IFD}}, \mathrm{RND}_{\mathrm{ICC}}, \mathrm{chip\,number})$
  - the card decrypts X_{IFD}, checks if $\mathrm{RND}_{\mathrm{ICC}}$ is present there. If no then abort.
  - the card sends $X_{\mathrm{ICC}} = \mathrm{Enc}_K(\mathrm{RND}_{\mathrm{ICC}}, \mathrm{RND}_{\mathrm{IFD}}, \mathrm{chip\,number})$
  - the terminal decrypts and checks the plaintext $\mathrm{RND}_{\mathrm{ICC}}, \mathrm{RND}_{\mathrm{IFD}}, \mathrm{chip\,number}$

GENERAL AUTHENTICATE (for the use e.g. with PACE)

PERFORM SECURITY OPERATION

option COMPUTE CRYPTOGRAPHIC CHECKSUM - computes a cryptographic MAC of a file

option VERIFY CRYPTOGRAPHIC CHECKSUM - checks cryptographic MAC of a file

option ENCIPHER – encrypts the data, the algorithm and mode used are determined first by MANAGE SECURITY ENVIRONMENT command

option DECIPHER - returns decrypted data

option HASH - returns hash of data, a switch available for performing only the last part of hash computation (for efficiency reasons)

option COMPUTE DIGITAL SIGNATURE

option VERIFY DIGITAL SIGNATURE

option VERIFY CERTIFICATE

option GENERATE ASYMMETRIC KEY PAIR

option MANAGE SECURITY ENVIRONMENT – setting active key, padding, parameters for key generation, ...

Global Platform:

LOAD – load data

INSTALL – specifies among others the size of volatile and nonvolatile memory reserved

Hardware test

TEST RAM

TEST NVM – testing non volatile memory, given area rewriten many times with a pattern

COMPARE NVM – reading NVM and checking if the given pattern really retained there

DELETE NVM – clears a given area of NVM

Electronic Purse commands (IEP Intersector Electronic Purse)

INITIALIZE IEP for Load– options:

load amount, currency code , PPSAM (SAM=Secure Access Module) descriptor, random number, user defined data

response: provider id (PPIEP), IEP identifier, cryptoalgo used, expiry date, purse balance, IEP transaction number, key information, signature, return code

CREDIT IEP

load: information on key to be used, signature,

response: signature, response code

INITIALIZE IEP for Purchase – the card sends data to the terminal: purse provider identifier, IEP identifier, crypto algorithm used, expiry data, purse balance, currency code, authentication mode, IEP transaction number, key information, signature

DEBIT IEP – charging the balance, parameters: PSAM identifier, PSAM transaction number amount to be debited, currency code, key info, signature, the card returns a signature of the transaction

Credit card commands

GET PROCESSING OPTIONS

GENERATE APPLICATION CRYPTOGRAM

command: desired application cryptogram, transaction related data

response: cryptogram information data, application transaction counter, application cryptogram, return code

Processing times examples:

(below data may differ for different implementations, especially for cryptographic operations)

READ BINARY 100 bytes: processing time 2ms, transfer 146 ms

UPDATE BINARY 100 bytes with erasing: processing 35ms, transfer 162 ms

EXTERNAL AUTHENTICATE: 235ms processing, 270ms data transfer

INTERNAL AUTHENTICATE: 135ms processing, 201ms data transfer

MUTUAL AUTHENTICATE: 135ms processing time, 163 ms data transfer

VERIFY PIN: 27ms processing time, 56 ms transfer time

DEBIT IEP: 235 ms processing time, 270 ms transfer

CREDIT IEP: 175 ms processing time, 222ms transfer

corollary: processing and transmission times are substantial!

Communication

sequence of steps:

1. power up on card
2. card sends ATR (Answer to Reset) , the smart card again in a sleep mode
3. the smart cards obtains APDU and changes to the active mode
4. the card responds

steps 3 and 4 executed in a cycle

ATR "answer to reset", sent on I/O line, max 33 bytes, usually a few bytes, transmission "divisor rate" the same for all cards, start must occur in a time window (400-40000 clock cycles - e.g. up to 8.14 ms for 4.9153MHz frequency of the signal), if not then tries 2 more times

TS - initial character, German: „00111011", French: „00111111" , start with 1, measures the time of elementary time unit: the time between two first falling edges of TS and divides by 3

T0 format character - which interface characters transmitted afterwards in ATR (a few options possible - see below)

interface characters: TA1, TB1, TC1, TA2 ... interface characters defining basic parameters of transmission, like guard time, divisor, etc.

- TA1: $\text{initial etu} = \frac{372}{f} \cdot \text{sec}$, where f is the frequency used

  $\text{work etu} = \frac{F}{D} \cdot \frac{1}{f}$ sec, where F is the rate conversion factor, D= bit rate adjustment factor (used due to variations of conditions during operation), the standard describes an encoding for a few combinations of values

- TAi for $i > 1$ define supply voltage and parameters of the clock

T1, ... Historical characters – chaos, many informations on smart cards, OS, ..

TCK check character (error detection code):

- protocol T=0: not sent as there is bytewise error detection
- protocol T=1: XOR checksum for all bytes starting from T0 up to the TCK

PPS (Protocol Parameter Selection)

Some cards allow changing parameters of the transmissions – important for SIM cards, USIM cards.

- there is a fixed collection of possibilities
- some of the configurations are not specified ("for future use")

APDU application protocol data unit

APDU is the only way of communication with the card.

- fixed format, quite compressed
- initiated by command APDU - sent from the terminal to the card, the answer is response APDU

command APDU

- header:
  - CLA (class byte, e.g. GSM denote as 'A0') it may denote credit card, electronic purse, private use, ...
  - INS instruction byte
  - P1, P2: two parameters, (no more possible, the meaning defined by the standard)
- body:
  - Lc field (length of data transmitted)
  - data field
  - Le field (length of expected response)

response APDU

- data comes first (length is defined by Le from Command APDU)
- afterwards the status bytes SW1 and SW2. The following options for the status: process completed (normal, warning), process aborted (execution error, checking error)

Secure messaging

the goal: securing the communication with the card: authentication, or even confidentiality

this is a problem even for contact interface as I/O contact can be traced

- **authentic mode procedure:**
  - the CCS (cryptographic checksum) computed, e.g. with AES,
  - the input is the original APDU+some padding (necessary to have a full number of blocks),
  - the out is the old APDU in plaintext encoded as TLV structure plus a TLV encoded CCS
  - Warning: there is no confidentiality!
- **combined mode procedure:**
  - CCS computed similarly as for authentic mode
  - then the resulting APDU encrypted: but only the TLV objects,
  - the result is APDu with CLA, INS, P1,P2 unencrypted, then TLV encoded cyphertext in the data field
  - Problem: the command and parameters are in plaintext! Information leaked
  - A solution: using a command ENVELOPE which says the card to decode the ciphertext and find there the real command to execute
  - the response APDU also contains encrypted data

send sequence counter

also a security mechanism: as APDU might be undelivered e.g. by jamming the radio channel

- started with a random value during a communication session
- then modified at each round
- two methods of encoding the counter:
  - as a separate data field
  - the counter is XORed with some portion of APDU before computing CCS – advantage: if the recipient knows the expected value of the counter, then he can easily recompute it . Main advantage: no extra communication

Logical channels

application may run in parallel on the card, no interleaving between request-response, an APDU may specify the logical channel - to which application the communication belongs

---

Data transmission with contacts

Physical layer

communication only digital, 0V as reference level, the other level is +5V, or +3V, or +1.8V