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

---

1. SMART CARDS

cards of no-smart solutions:

- **embossed** - credit cards: reading does not require electricity, elementary protection only
- **magnetic:** $\approx$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

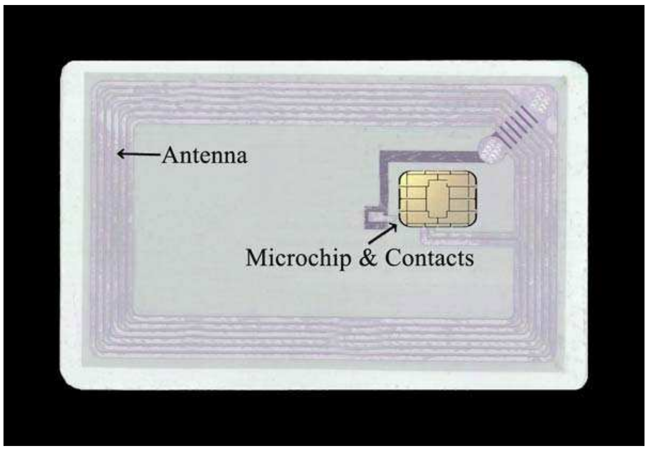
processor cards:

```
I/O ——— CPU  —- flash memory
                  RAM
                   ROM
           |
          NPU (numerical processing unit)
```

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 $\rightarrow$ 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: $a_1, \ldots a_n$, 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):**

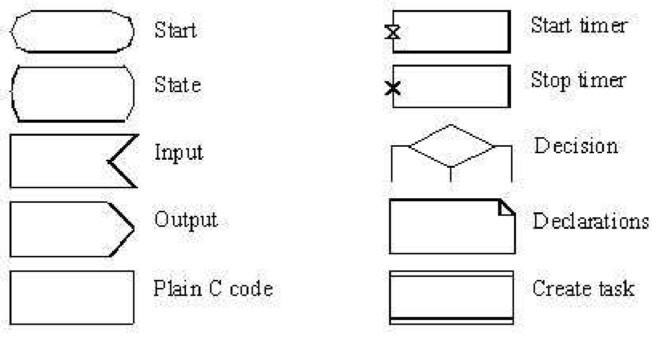
  only very simple methods used as compression/decompression procedures might require too much space and it does not pay off to compress. Two main methods:

  - 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 1  - - - - - - -
|               |
input B    input A
 |              |
 state 2      state 1
 | _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                                    |
input E                              input C
|                                    |
state 4                              state 3
| __________                          |
|            |                        input D
input F      input G                   |
|            |                        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 $X_{\mathrm{IFD}} = \mathrm{Enc}_K(\mathrm{RND}_{\mathrm{IFD}}, \mathrm{RND}_{\mathrm{ICC}}, \text{chip number})$
  - the card decrypts X_{IFD}, checks if $\mathrm{RND}_{\mathrm{ICC}}$ is present there. If no then abort.
  - the card sends $X_{\mathrm{ICC}} = \mathrm{Enc}_K(\mathrm{RND}_{\mathrm{ICC}}, \mathrm{RND}_{\mathrm{IFD}}, \text{chip number})$
  - the terminal decrypts and checks the plaintext $\mathrm{RND}_{\mathrm{ICC}}, \mathrm{RND}_{\mathrm{IFD}}, \text{chip number}$

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” (“hl+ hhlhhhll”), French: „00111111” (“ hl + hhlllll”), 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 - defines 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
- TC_1 defines extra guard time (standard value is 2etu)
- TC_2 defines maximum waiting time between characters

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.

- changes on demand by the terminal
- there is a fixed collection of possibilities
- some of the configurations are not specified (“for future use”)

APDU - acronym for 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 denoted 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 output 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

conventions: 0V represents 0 (direct convention), or 1 (inverse convention)

based on RS232 (0V used instead of negative voltage)

I/O line is in high level always when the data is not sent

serial communication

asynchronous: so the sender adds the start bit at the beginning of each transmission (low voltage), 8 data bits, followed by the parity bit, and stop bits (used as guard time)

transmission speed: frequency/divider, divider says how many periods of the carrier frquency needed to encode a bit. Typically divider is 372 and 512, 5MHZ max frequency, 32 minimum usable divider so at most 156 250 bit/s

Memory cards

simplest protocols, variety,

prepaid telephone cards

ISO protocols (15 versions described), mosty used T=0, T=1

T=0

- France, early standard, simple, minimal memory usage
- used for GSM worldwide
- byte oriented
- command structure: CLA, INS,P1,P2,P3, data field, P3 specifies the length of data field
    - after receiving the header the card send ACK in a form of Procedure Byte
    - then the data transmitted

- one byte - twelve bits due to start bit, parity bit and 2 etu for guard time

error handling:

- retransmission of byte immediately if an error detected (not after a block of bytes), error detected by wrong parity of a byte
- reporting error I/O line down after byte transmission, in halfway of etu (so in a "wrong place"), during the guard time
- this mechanism technically problematic sometimes, since in some cases below 1 etu nothing is detectable

SM hard - many versions (with overhead)

interpretation of an APDU by a relatively simple state machine

T=1 protocol

- asynchronous
- half-duplex (both directions communication but not at the same time)
- block oriented (a block is the minimal unit to be transmitted)
- follows layer concept of communication protocols (data for higher layers sent transparently by data link layer)

Block:

- contains application data and control data of the protocol
- structure of a block:
  - prologue field: 1 byte NAD(node address- destination and source), 1 byte PCB (protocol control block - e.g. sending sequence number(mod 2)), 1 byte LEN (length)
  - information field: APDU
  - epilogue field: EDC . It contains either a CRC error detection code from ISO or just LCR - XOR of all bytes (faster, easier but less usable as error detection)
- send/receive counter: binary
- parameters for transmission to avoid deadlocks:
  - → CWT -character waiting time, in order to avoid deadlock, (time interval between the leading edges of characters), CWT = $2^{\mathrm{CWI}} + 11$ etu, CWI parameter is taken from ATR, default CWI=13
  - → BWT block waiting time: between leading edge of XOR in the epilogue field and leading edge of NAD in the response of the card

    $\mathrm{BWT} = 2^{\mathrm{BWI}} \cdot 960 \cdot \frac{372}{f}\,\mathrm{sec} + 11\,\mathrm{etu}$

for the default BWI=4 it means ≈1.6 sec

→ BGT - block guard time: time between communications in opposite directions

- waiting times can be changed during protocol execution
- block chaining: if block are larger than the buffer of the sender or receiver, then the data field is splitted and a chain of blocks is used. a flag “more” is used to indicate a chain
- error handling:
  - sophisticated mechanism
    - error ⇒ the sender receives an R block indicating an error ⇒ retransmission of the last block
    - if retransmission ok, then resynchronisation with an S block, card acknowledges and the counters (on both sides) reset to 0
    - if a block cannot be sent then RESET and the whole session terminated, data transmitted so far is lost

Other protocols with potentially high importance:

USB (Universal Serial Bus)

- T=0, T=1 are no for slow data transmission,
- target: 12Mbit/s USB communication
- the main problem is time synchronization, USB assumes stability of frequency on both sides, but the smartcard has frequency from the reader – this is unstable. Extra circuit on the smart card required
- problem also with the code: USB requires a few KB of code for interface on card
- USB is encoding bits via difference of electrical levels on a pair of signal lines (more reliable than the encoding on a line), C4 and C8 used for USB (AUX1 and AUX2 conatacts, normally unused)
- encoding method NRZI (nonreturn to zero) encoding: a 0 encoded by polarity reversal, for 1 polarity is unchanged,
- clock for the bus retreived from the polarity reversals, what to do if there is a long block of 1’s? after six ones obligatory bit stuffer - a zero.
- logical connection: 4bit “endpoints”. EP0 for control, other endpoints are unidirectional
- transfer modes:
  - control transfers: initiated on EP0, acknowledged on both directions,
  - interrupt transfers for small amounts of data – not really an interrupt, since the time periods for interrupt transfers are fixed

  - bulk transfers: non-time critical for large amounts of data
  - isochronous transfers: for time critical transfers
- least significant bit sent first (little endian), data sent in frames at fixed time intervals (about 1ms), a frame starts with a SOF (start of frame) packet containing a packet identifier, frame number and CRC code

SWP (Single Wire Protocol)

- NFC controller has to find a way to communicate with the SIM card while the card is communicating with the telephone with another channel
- C6 contact used as a single line (previous usage as external programming voltage for EEPROM not needed anymore)
- full duplex
- complicated encoding at the electrical level (card is always the slave):
  - S1 (master-to-slave): high voltage at 75% time – a 1, high voltage at 25% of time - a 0
  - S2 (slave to master): influences the voltage at the high voltage period of S1: 0 by reducing the voltage, 1 by keeping it high

---

Contactless transmission

flexible batteries - expensive, so energy must come together with communication

INDUCTIVE COUPLING:

range depends on power consumption inside: no computation on card, then range about 1m, with writing on the card- range 10cm to enable enough energy, 1m – few tens of microwats, writing already 100 microwats, processor already 10miliwat, legal restrictions on energy sent by readers

wavelength 2400m or 22m depending on frequency: much smaller than the distance between the card and reader: physical transformer model can be used

voltage: rectified on the chip, proportional to the signal frequency, depends on the number of turns in the coil and the coil area, strong current needed on coils

picture page 286

transmission back: (small) fluctuations of current on the side of smart card, in order to be able to separate from the strong signal form the reader: a different frequency for modulation is used, the reader applies a filter to separate them

picture page 287

CAPACITIVE COUPLING:

by putting very close (on surface), coupling area both on terminal and on the card, two fields on card, two on reader, too low energy to run microprocessor directly so inductive coupling used anyway for providing energy

Collision avoidance:

collisions between cards: possible as a single radio channel.

Methods of separation:

- space division multiple access
- time division multiple access - TDMA
- frequency division multiple access - FDMA
- code division multiple access - CDMA – codes on the same frequency. Decoding: searching how the transmission received could be composed of individual transmissions (main idea of UMTS)

CLOSE COUPLING CICC

up to 1cm – slot or surface operation, must tolerate electrostatic discharge,

power transmission: via inductive coupling, power frequency 4.9 MHZ,

two pairs H1,H1 and H3, H4 have phase shift of 90 degrees,

two pairs to make it resistant to card position

data transmission:

- terminal to card (PSK phase shift keying): 4 alternating magnetic fields, all 4 fields shifted by 90^o, to change a bit all four fields shift simultaneously by 90^o, at the beginning a 1 is always sent picture page 294
- data transmission from the card to the terminal: modulation at 307.2kHz, change of bit by shift of phase by 180^o of the subcarrier. picture page 294

Capacitive data transmission

- timing constraints: how much time when power off, when power rises, for the first transmission to the card, for the first transmission from the card

PROXIMITY COUPLING ISO 14443

- normal operation up to 10cm, large antennas for signal detection from bigger distances
- inductive coupling, magnetic field strength defined to be in some interval
- transmission frequency about $f_c = 13$ Mhz
- many different applications
- communication interfaces: type A, type B (terminals periodically perform both just to be able to talk with both types of cards)

type A:

- initialization: bitrate $f_c/128$, later a different bitrate chosen: f_c/k where $k = 128, 64{,}32, 16$

so the bitrate is 106kbits/s or 212 or 424 or 847

communication from terminal to card:

- amplitude modulation - with signal going to zero (short interrupts)
- small interrupts (3 μs) $\Rightarrow$ no energy transmitted to the card but the card survives operating
- encoding: modified Miller encoding:
  - $\rightarrow$ transmitting a 1: pause after half bit interval
  - $\rightarrow$ transmitting a 0: no pause
  - $\rightarrow$ transmitting more zeroes: a pause between two consecutive zeroes, the first 0 has a pause before
  - $\rightarrow$ start: a pause
  - $\rightarrow$ example: | 0 | 1 | 0 | 0 | 1 |

    □■■■■■□■■■■■□■■■■■□■
  - $\rightarrow$ end of message logical zero 0 followed by one bit period with no pause

from card to terminal

- load modulation with a subcarrier
- transmitting a 1: carrier modulate by subcarrier in the first half of the bit interval
- transmitting a 0: carrier modulate by subcarrier in the second half of the bit interval
- start of a message: carrier modulated by the subcarrier in the first half of the bit interval
- end of the message: no modulation by a subcarrier

fig. 10.21 page 302

type B:

- no interruptions of energy supply

terminal to card:

- subcarrier modulation
- modulation index 10% (slight changes of the amplitude)
- encoding: NRZ (non-return-zero), simply a higher amplitude means a 1 and the lower one means a 0.

card to terminal:

- modulation with BSPK (binary phase shift keying)- shift of $180^{\mathbf{0}}$
- no change of the character = no pahse shift, if there is a change then a phase shift,
- initially: subcarrier with no phase shifting for some period of time
- the first character is understood to be a logical 1

figure page 304

Anticollision mechanism

two cards communicating with the reader during initialization phase

type A

- Manchester encoding used to ensure collision detection:
  - subcarrier modulated in the half of the interval
  - ”first high- then low” or “first low-then high” to encode two bit values
  - if the signals are synchronous than overlapping produces a “high-high” and an evidence of collision
- all cards are transmitting synchronously their IDs
- after Reset the card in the Idle state and can only answer the Type A Request (REQA) and Type A Wake-up (WUPA),
- then the cards send synchronously Answer to Request type A (ATQA)
- the terminal receives the responses and knows that at least one card is present
- message format ATQA: the first part for the reader the second part for the cards,

  in the first part bytes of address specified, in the second part all matching cards respond.
- The length of the parts vary, but the total length is the always the same. The first part has at least 16 bits and no more than 55 bits.
- ANTYCOLLISION or SELECT: the message specifies UID (card ID - either random for the session or a fixed one) and the number of bits relevant. Only the cards that match the UID (on a given portion), reply
- step by step until only one card responds

type B

- predefined time slots
- dynamic slotted Aloha, number of slots in the REQB
- all cards Idle until REQB, then choose a random slot to send its data

SIM Cards and telecommunication systems

Problems to be solved:

- how to authenticate the subscribers?
- how to protect the communication confidentiality?
- how to know the position of the subscriber?
- distributing security functions?
- how to work in roaming with partially trusted partners?
- interoperability

GSM

- suscriber ID: IMEI (international mobile equipment identity) - ID of a phone, IMSI- international mobile subscriber information - ID of a user (never transmitted in clear), TMSI - temporary mobile subscriber identity – temporary and related to local area
- SIM card (Subscriber Identity module):
  - dedicated commands
  - storage for some user data
  - user management
  - security functions: authentication, encryption
- key management:

  a secret key Ki on the smartcard, shared with the provider, a copy available in AuC

  possible on-the-fly derivation Ki=f(IMSI), where f is a private function of the provider
- authentication:
  1. BSS (or MSC) generates RAND at random and sends to the mobile station
  2. mobile station: SRES:= g(RAND,Ki), where g is a provider specific function (e.g. might be A3)
  3. SRES sent to BSS, where the result is checked
- encryption:
  1. RAND reused
  2. mobile station: Kc:=h(RAND,Ki), where h is a provider specific function (e.g. might be A8)
  3. BSS: Kc:=h(RAND,Ki)

4. encryption with the session key Kc and a stream cipher. The key for a frame:

   $r(\text{TDMA frame number}, \text{Kc})$ where r is a provider specific function (e.g. A5),

   the encrypted frame is the

   frame plaintext XOR frame key

5. the communication: down and uplink interleaved, short frame

- problems with distributing the user secret keys:
  - instead of providing the key Ki of the user, AuC distributes on demand the triples (RAND, SRES, Kc)
  - the local provider from the Visited Network does not have to implement the provider specific algorithms, or even to know them ⇒ easy roaming and cooperation with different vendor systems
- attack (IMSI catcher device used):
  - the mobile user gets authenticated, but the network is not authenticated,
  - use a fake base station, starting authentication, accepting whatever SRES comes and switching off the encryption
  - connecting to other subscriber via a different link (via a prepaid card, not showing subscriber number)

- attack: only the links between the mobile phoine and the base station are secured, no end-to-end encryption or authentication. E.g. point-to-point microwave links used in the access network, where eavsdropping is easy
- protecting identity: TMSI used instead of IMSI after first authentication of a user in a service area
- local authentication: not all time the triples from the AuC used, instead the sessions are linked in the Visited Network
- cloning a SIM card is a problem (the provider may create a copy of the card e.g. for eavsdropping)

UMTS

- USIM card used instead of SIM
- CDMA coding - more sophisticated and in principle better. In practice leads to problems, and LTE is simplified a lot (a nd makes it more efficient)
- reuse of concepts: limited trust in the Visited Network, identity protection
- correcting security problems:
  - fake base station
  - encryption in the access network

    - no cleartext transmissions

- authentication tuples: (RAND, XRES, CK, IK,AUTN), XRES=expected response, CK=cipher key, IK=integrity key

- XRES=f2(RAND,Ki), CK=f3(RAND,Ki), IK=f4(RAND,Ki), AUTN=SQN$\otimes$ AK||AMF||MAC, AK=f5(RAND, Ki), MAC=f1(RAND,AMF,Ki), AMF - system parameters

- AKA (Authentication and Key Agreement protocol):

    1. service network to mobile user: RAND(i), AUTN(i) (taken from the list)

    2. mobile user computes verifies AUTN(i) (SQN must be in a correct range), recomputes XRES(i) via definition, replies with XRES(i)

    3. use CK(i) for encryption and IK(i) for integrity

- problematic synchronisation with SQN

Graphical security means:

- Guilloche patterns - fine lines on the surface under the outer transparent foil, in case of any manipulation the pattern destoryed. Technique used on bank notes

- colored signature field - printed paper strip glued to the surface

- microtext - look like simple lines but something printed – used on bank notes, defence against photocopying, readable only under a loupe

- ultraviolet ink

- barcodes (one and two dimensional), two dimensional PDF 417 can encode up to 1000 bytes, error correction codes so that up to 25% of the surface can be damaged (dirty)

- hologram - few companies in the world, cheap, holograms are embossed holograms, holograms reflected in diffuse daylight (some holograms require laser light), permanently bonded to the surface microstructure,

- kinegram - as holograms, show different image from different angles.

- MLI: (multiple laser image) - small lenses, some are blackend by the laser. Looks like a hologram but can contain personalized information (holograms are always the same)

- embossing - like in credit cards (the characters are pressed with a considerable force). Rather old style...

- laser engraving (surface or inside, uder the coat) – equipment fairly expensive, used to personalize cards. However, it is slow (major slow down for production of ID cards). However - a professional forger can make corrections ...

- scratch field – nice for card delivery. The character printed under the coat are not readable even with ultraviolet, infrared light etc

- thermochrome (TC) display: not a real display, but can be reprinted with a special reader. heating a point to 120^oC makes a black dot. Heating the whole strip makes it almost transparent again
- MM (modulated feature) – hidden MM box, invisible, contains control digits for the contents of the magnetic strip. Used by POS and ATM terminals. Control digits computed with MM algorithm

---

TPM

- Trusted Platform Module, small integrated circuits on motherboard of PC, laptop (but also specifications for mobile, embedded, ...), 28 pin, may be integrated with other chips
- passive hardware module (responds to requests and nothing more), complicated command set, TCG Software Stack to ease using TPM,
- specification by Trusted Computing Group of Trusted Platform Alliance (industry initiative)
- on major products: Dell, Lenoovo, HP, Toshiba, ...
- not allowed in Russia (no TPM), own obligatory specification in China (partially secret)
- ”root of trust”
- specification TPM 1.2 and TPM 2.0
- lack of commertial success
- inexpensive chip, hard to use, limited usability
- controversies: a tool to monopolize the software market
- origin: sophisticated malware attacks, defend insecure OS by TPM as antivirus is only partially effective

goals:

- secure storage
- keeping track of booting process
- attestation of the state of the operating system
- not really converting into secure operating systems
- applications started only in a certain state of OS
- flexible to work with different systems

Crypto (TPM 1.2):

- RSA (2048 and 1024)
- SHA-1 (no hash longer than 160 bits!)
- HMAC (keyed-Hash Message Authentication Code)
- RNG
- no symmetric crypto due to US law at that time (restrictions to export and therefore the whole idea of standardization fails for symmetric algorithms)
- operations: asymmetric encryption, signatures, HMAC,

Storage:

- PCR (platform configuration registers), storing SHA-1 hash values (160 bits)
- each PCR has an index and is devoted to different data: 0-7 measure core system components like BIOS, 8-15 are OS defined, 16-23 for late launch
- operations on PCR:
  - reset at boot time
  - otherwise only extend operation Extend(PCR,value)=Hash(PCR||value)
- NVRAM - small nonvolatile random access memory
- counters (only increasing until reset when back to 0)
- storing keys:
  - the main key permanently in TPM and never allowed to leave it ("stored internal to the TPM") (SRK - Storage Root Key)
  - other keys encrypted and exported outside. In case of need imported to TPM, decrypted with keys that are in TPM
  - exported data contain the key value itself and a "proof" - a value known to TPM only, and authorization information. The exported ciphertext is called a "blob".
  - importing: load command and a blob, TPM decrypts with SRK, checks the proof, checks the supplied password
  - keys might be for encryption or digital signature

Measured Boot

- booting process: a sequence of stages where the next code uploaded and executed
- before executing a code its hash "extends" the PCR value (which PCR depends on the stage)
- the first piece of code should be completelty write-protected

- result: a set of PCR values that cannot be manipulated. A fingerprint of what has been booted. may be used to check against a know value – in case of manipulation the state of PCRs is different than expected
- this is not the same as Secure Boot where the signature over the code is checked at each stage

Late launch:

the same mechanism as for booting for security critical issues

not controlled by operating system, not breakable

Sealed Storage

- files encrypted with key exported from TPM. Decryption only via TPM
- decryption only when the machine is in a proper state (judgement based on the contents of PCRs)

Network Attestation

- sending data about the machine state
- steps:
  - random nonce sent from a server to TPM
  - client software determines identity key to be used
  - TPM hashes PCR values together with nonce and signs with identity key
- nonce is for freshness

Overview of TPM 2.0

Cryptography:

- RSA of TPM 1.2 became weak
- SHA-1 of TPM-1 became weak, but replacing it with SHA-2 or SHA-3 impossible due to hash length (additional 12 bytes)
- TPM 1.2. stores RSA keys via factorization: but only one factor - which saves space, but is slow (storing both factors would be impossible due to the size). TPM 2.0 uses symmetric encryption and there is no problem
- TPM 2.0 lists a variety of algorithms: SHA-1 and RSA for backward compatibility but also SHA-256, AES, EC, (also Chinese algorithms?)
- encryption of a key should be by a key at least as strong as the encrypted key. Contorversies what is stornger... Mixing algorithms not allowed.
- if a key cannot be duplicated and moved to another TPM then it must be encrypted with exactly the same algorithm

Authentication

- TPM 1.2 very limited: only by passwords and PCR values
- TPM 1.2: password inserted into the command sent to TPM, problem with two users for the same machine!
- Enhanced Authorization (EA) of TPM 2.0:
    - basic mechanism: software interacting with TPM must show knowledge of a password
    - policies: multiply authorization methods and Boolean logic to combine them
    - TPM stores a hash of a Boolean logic and values for authentication methods
    - authentication methods:
        - passwords
        - PCR values
        - TPM counter
        - commands (object use may be limited to certain commands)
        - digital signature of a user (form a smart card)

Compilible Specifications

- specification well structured (1.introduction, 2. list of variables, structures, constants, 3. list of commands with brief description in English, 4. subroutines for part 3 in pseudocode)
- misunderstandings because of pseudocode
- Microsoft tool to extract compalible reference TPM – but not available for external users

specification well structured (1.introduction, 2. list of variables, structures, constants, 3. list of commands with brief description in English, 4. subroutines for part 3 in pseudocode)

Activation

- TPM 2.0 on by default
- firmware has full access to TPM just like the owner

TPM 1.2 details

secure storage

- initialization of TPM: invoking TPM_TakeOwnership creates SRK (Storage Root Key), at the same time also secret tmpProof - to identify the blobs produced by the TPM
- tree of keys

- creating a key: TPM_CreateWrapKey , arguments: parent key, new encrypted authdata, authorization HMAC based on the parent authdata, key exported
- loading a key: necessary to use the key, TPM_LoadKey2, arguments: key blob, returns handle to the key to be used in following commands
- after key loaded one can use TPM_Seal and TPM_Unseal , requires authorisation from authdata

authentication sessions

OIAP – object independet authentication protocol,

nonces of the user process: “odd”

nonces of the TPM: “even” nonces

nonces sent with each message, authentication HMAC takes into consideration the most recent odd and even nonce (linking the messsages)

for commands that can mniupulate many objects

OSAP – object specific ...,

OSAP secret: based on even and odd nonce and authdata for the object

HMAC computed with OSAP secret

ADIP encryption

AuthData Insertion Protocol

OSAP session can invoke it

authdata is encrypted with XOR with one-time key computed as SHA-1(s, n_e) where s is the OSAP secret and n_e is the current even nonce

Platform Authentication

- each TPM has a unique pair of *endorsement keys* (EK) set at manufacture time and (usually) certified by manufacturer, used for encryption only
- may be changed by the owner, but regarded as the identity long time key, (tpmProof is secret, so cannot be used as identity)
- AIK (*application identity key*) can also be created. They are used to sign application-specific data and PCR state via TPM_Quote,
- AIK has to be authenticated! Either from a certificate from privacy CA or via Direct Anonymous Attestation

Privacy CA

- signs public part of AIK
- user software:
  - creates AIK via: TPM_MakeIdentity,

- sends public key together with certificate on EK to CA
- CA checks the certificate of EK
- CA signs a certificate for AIK
- CA encrypts certificate with a new session key -> blob B1
- CA encrypts the session key and the publik AIK with EK, creating blob B2
- CA sends blobs to software
- software sends B2 to TPM
- TPM checks that AIK is in the TPM and releases the decryption key
- software decrypts B1

Key migration

migratable keys

- TPM-CreateWrapKey with parameter, authdata specified for allowing migration
- TPM_CreateMigrationBlob with input: wrapped key, its parent key handle, migration type, authorisation digest (parent key usage authorisation, migration authorization)
- type REWRAP: rewraps with migration public key (key of another TPM)
- type MIGRATE: unwraps and XORs with a random password r, returns the blob and r

certifiable migratable keys

- created with TPM_CMK-CreateKey and then certified with TPM_CertifyKey2
- certificate specifies Migration Selection Authority (or more) (authorities restricting destinations)
- Pre-approved Destinations (PAD)
- creating a CMK:
  - TPM_CMK_ApproveMA: source TPM creates a ticket to approve some MSA and PDAs
  - TPM_CMK_CreateKey: takes ticket from TPM_CMK_ApproveMA
  - TPM_AuthorizeMigrationKey: owner of TPM approves migration of the key
- exporting CMK: (to PDA or MSA)

  TPM_CMK_CreateBlob on

    - CMK wrapped key
    - migrationKeyAuth

  - MSA list
  - sigTicket form MSA nad restrict Ticket (public keys belonging to MSA, destination parent key, and key-to-be-migrated)

Delegation

the purpose: delegating some priviledge of the owner to another party and may be withdraw later

possible to create authdata value for the resource with list of admitted commands

Blobs:

AuthData– controls ability to read, write, aund use objects stored in the TPM and execute TPM commands, 160 bit secret, knowledge of it by computing HMAC

key blobs - Blob(PK,SK) = PUBBlob(PK,SK) || Enc(PKparent, PrivBlob(PK, SK))

PUBBlob(PK,SK) = PK || PCR values

PrivBlob(PK, SK) = SK|| Hash(PubBlob(PK,SK)) ||tpmProof

Problems:

– huge documentation

- complex administration

- when starting then operation resembles malware operation (rebooting, strange communicates)

- typical PKI problems

- privacy issues (when used for attestation)

– low effect

- malicious use not controlable by OS

REMOTE ATTESTATION

goal:

- attestation that a target machine has certain (desirable) properties
- make decision whether further interaction with this machine (e.g. whether to transmit sensitive data)
- it should check, whether the target machine is safe if the target is safe

Problems to be solved:

attacks:

- inserting unauthorized code to the servers on a large scale
- the attack is remote
- attacked computers are standard general purpose machines,

- network involves interaction between independent entities (retailers, distributors, financial organizations,...)

Goal: protocol parties can check remotely that the computer of the partner is in some sense in a decent state (e.g. a bank and a customer: the bank has to check if the customer's PC is protected)

Attestation architecture:

desired properties:

- measurements: measure diverse aspect of target, choice non-trivial
    - a bad strategy: hashes of memory, one cannot derive too much about machine behavior,
    - problems with data compression
- domain separation:
    - the attester should not interfere too much with the operation on the target
    - the state of the measurement tool must be inaccessable to the target machine
    - the result: *corrupted* or *uncorrupted* - additional data is itself a problem
    - separation implementation strategies:
        - a copy of the target on a Virtual Machine
        - hardware separation: coprocessor used for attestation only, visibility by hardware constraints and not by configuration of a hypervisor
- self-protecting trust base: not too big (then audit/certification of the trust base too complex)
- delegate attestation to proxies:
    - if some data must not reach the attester, then use a proxy between the target and the attester
    - specialized proxies
- attestation management: flexibility is necessary, queries: constraints subject to policies, communication, Attestation Manager as a local component to govern these issues

Example Architecture

Components:

- trust base: TPM/CPU, hypervisor
- S-Guest: supervisory virtual platform: contains virtual TPM, M&A (measurements and attestation)

- U-guest: user platform with "normal" operating system: contains virtual TPM, M&A

PROTOCOLS

IBM Protocol:

verifier: sends a nounce N, 160 bits to attestation service

attestation service: request N to TPM

TPM to attestation service: ML (measurement list), $\mathrm{Sign}_{\mathrm{AIK}}(\mathrm{PCR}, N)$

attestation service to verifier: $\mathrm{Sig}_{\mathrm{AIK}}(\mathrm{PCR}, N), \mathrm{ML}, \mathrm{AIK}_{\mathrm{cert}}$

verifier: check signatures and certificates

CAVES

- Attester, Client, Server, Verifier, EPCA (Enterprise Privacy CA)
- steps:
  1. Client to Server: request R, attester A, key K encrypted by Server's key (K is a session key for Client-Server)
  2. Server to Verifier: S (Server), R, A, N_S (server nonce) encrypted with Verifier's public key,
  3. Verifier gets from EPCA a signed record containing certificate, A, I(identity key of Attester), E (name of EPCA)
  4. Verifier to Server: encrypted: N_V (verifier's nonce), J (query), V, N_S, M (PCR mask)
  5. Server to Client: N_V, J, V, M encrypted with K.
  6. Client to Attester: (encryption with a long period shared symmetric key) S, N_V, J, V, M, R.
  7. Attester to Client: (encryption as above)
     - K' (extra key), N_V, B (blob)
     - B is a ciphertext encrypted with the public key of the verifier of:
       - K', S, J_O (mesurement), M, P (PCR vector)
       - signature created with I for $\mathrm{Hash}(\mathrm{Hash}(A, V, R, N_V, J, J_O), M, P)$
  8. Client to Server: B
  9. Server to Verifier: B
  10. Verifier to Server: encrypted with K_S: verification result, N_S, K'
  11. Server to Client: data encrypted with K'

Some properties:

- client and server cannot read the attestation result and cannot manipulate it without detection

---

Direct Anonymous Attestation (DDA)

goal: remote testing a device on the other side without learning its identity

parties:

- host (operating system and application software)
- TPM
- the issuer (similar as Privacy CA and checks EK certificate)
- verifier

protocol:

- DDA-join: TPM chooses a secret f, the issuer creates a blind signature of f (credential cre), encrypts it with the Endorsement Key and sends it to the TPM
- DDA-sign: signature of AIK using (f, cre)

Building blocks

Schnorr authentication modulo an RSA number

- RSA number n of length l_n
- user's public key $y = g^x \bmod n$, x is a private key of length l
- authentication: the prover chooses r of length $l_n + l_c + l_{\text{phi}}$ and $t = g^r \bmod n$, he sends t to the verifier
- the verifier chooses c at random
- the prover replies with $s = r - x \cdot c$ computed **as integers**, for large l_{phi} this does not leak much information about x
- the verifier checks that $t = y^c \cdot g^s$
- security argument based on Strong RSA assumption:

  on a random RSA number n and $u < n$ it is infeasible to find v and $e > 1$ such that $v^e = u \bmod n$.

  It follows that for random g, h it is hard to compute $w < n$ and integers a, b, c such that $w^c = g^a h^b \bmod n$ where c does not divide a and b

A version proving that x is in a given interval

- as above but r of length $l_x + l_c + l_{\text{phi}}$
- .. and a check that s is of the length at most $l_x + l_c + l_{\text{phi}}$ (including negative numbers)
- similar protocols may be built to show that the secret is in a given interval

Camenisch-Lysyanskaya Signature

- blind signature: a signer does not see the message to be signed,
- the resulting siganture is randomizable
- keys: $n = p \cdot q$ RSA number, public key: randomly chosen squares $R_0, R_1, ..., R_{L-1}, S, Z < n$, secret key: p
- message space: L strings of l_m bit numbers
- creating a signature:
    1. choose a prime number e of length $l_e > l_m + 2$
    2. choose a random number v of length $l_v = l_n + l_m + l_r$, where l_r is a security parameter
    3. compute $A := \left(\frac{Z}{R_0^{m_0} ... R_{L-1}^{m_{L-1}} S^v}\right)^{1/e} \bmod n$

       where $1/e$ means $1/e \bmod (p-1)(q-1)$
    4. output (A, e, v)
- verification: check that $Z = A^e R_0^{m_0} ... R_{L-1}^{m_{L-1}} S^v \bmod n$ and that the range of each element is as described
- provable security: based on the strong RSA assumption

Camenisch-Lysyanskaya Signature on a (partially) secret message

- as CL but now the receiver computes $U = R_0^{m_0} ... R_i^{m_i} S^{v'}$ for v' chosen at random, and sends U to the signer
- the receiver proves (noninteractive zero konwledge proof) that he actually knows some $m_0, ..., m_i, v'$ that yield U
- the signer computes $A := \left(\frac{Z}{U \cdot R_{i+1}^{m_{i+1}} ... R_{L-1}^{m_{L-1}} S^{v''}}\right)^{1/e} \bmod n$
- the signer proves that A has this form
- the signature is (A, e, v), where $v = v' + v''$

Proving posession of a signature

- proving possesion of signature (A, e, v) of m, i.e. satisfying $Z = R_0^m S^v A^e \bmod n$

- given a signature (A, v, e) choose r at random and compute $A' := A \cdot S^{-r}$, $v' := v + e \cdot r$, the new signature is (A', e, v')
- A' is uniformly distributed, so it does not leak information on the signature
- proving possession of ϵ, v', μ such that $Z = R_0^{\mu} A'^{\epsilon} S^{v'}$ and that they are in a proper interval

Direct Anonymous Attestation - an overview

Properties:

unforgeability. DAA-Sign can be sucessful only if TPM before has executed DDA-Join

anonymity. information from DDA-Sign do not enable to link with DDA-Join

rogue tagging. given DDA related secret information of a rogue TPM, the verifier can check if DDA-Sign has been executed by this party

System parameters:

- lengths of diverse parameters,

Key generation (performed by the Issuer)

- RSA number $n = p \cdot q$ $(p = 2p' + 1,\ q = 2q' + 1)$
- S random generator of quadratic residues mod n (squares modulo n)
- random $x_0, x_1, x_z < p'q'$
- $Z = S^{x_z}$, $R_0 = S^{x_0}$, $R_1 = S^{x_1}$
- a proof that Z, R_0, R_1 have been computed correctly
- primes ρ, Γ, with $\Gamma = r\rho + 1$, where ρ is not a factor of r
- group $\mathbb{Z}_\Gamma^*$, and an element $\gamma \in \mathbb{Z}_\Gamma^*$ of order ρ
- a random identifier ζ_I

DDA-Join

- TPM chooses a random $f \in \mathbb{Z}_\Gamma^*$, $f = f_0 || f_1$ (concatenation)
- TPM computes its pseudonym $N_I = \zeta_I^f \bmod \Gamma$ and sends it to the Issuer
- the issuer checks all rogue (f_0', f_1') whether $N_I \neq \zeta_I^{f_0' || f_1'} \bmod \Gamma$ (if yes, then abort)
- blind signature of f by the Issuer:
    - TPM chooses v' at random, $U = R_0^{f_0} R_1^{f_1} S^{v'}$, sends U to the issuer
    - TPM proves that U has been created correctly
    - TPM authenticates U to the issuer with the endorsement key

- Issuer chooses random $v'' = \hat{v} + 2^{\cdots}$ (a leading one inserted)

  $$A = \left(\frac{Z}{U \cdot S^{v''}}\right)^{1/e}$$

  sends A, v'', e to the host of TPM

- Issuer proves to the host that A, v'' have been correctly created

- the host stores A, v'', e and sends v'' to the TPM

- TPM sets $v = v' + v''$ and stores (f_0, f_1, v)

- so after executing the protocol host+TPM obtain a "certificate" from the Issuer, but neither the host not the TPM knows it completely

DDA-Sign

- the host sends ζ, a random power of γ, to the TPM

- TPM computes $N_V = \zeta^{f_0 || f_1}$ and sends it to the host

- TPM and the host create a "signature of knowledge" that they know together DDA-certificate and that the same numbers f_0, f_1 have been used:

  i. the host picks w at random at some range and computes $A' = A \cdot S^{-w} \bmod n$

  ii. TPM picks r_v, r_0, r_1 in some predefined range and randomizes:

  $\hat{U} = R_0^{r_{f_0}} R_1^{r_{f_1}} S^{r_v} \bmod n$

  $\hat{r_f} = r_{f_0} || r_{f_1} \bmod \rho$

  $\widehat{N_V} = \zeta^{\hat{r_f}} \bmod \Gamma$

  and sends $\hat{U}, \widehat{N_V}$ to the host

  iii. the host picks r_e, r_w at random (in an appropriate range) and computes

  $\widehat{A'} = \hat{U}^{-1} \cdot A'^{r_e} \cdot S^{r_w}$

  iv. the host receives a nonce n_V from the verifier and computes a hash c_h of

  ... (parameters), $\zeta, A', N_V, \widehat{A'}, \widehat{N_V}, n_V$

  v. TPM computes $c = \text{Hash}(\text{Hash}(c_h, n_t), b, m)$ for a nonce n_t and $b = ?$

  and sends c, n_t to the host

  vi. TPM computes $s_v = r_v + c \cdot v$, $s_{\phi_0} = r_{f_0} - c \cdot f_0$, $s_{\phi_1} = r_{f_1} - c \cdot f_1$ and sends them to the host

  vii. the host computes over integers

  $s_{\hat{\epsilon}} = r_e + c \cdot (e - 2^{l_e - 1})$, $s_{\hat{v}} = s_v + r_w + c \cdot (e \cdot w)$

  viii. the signature $\zeta, A', N_V, c, n_t, s_{\hat{v}}, s_{\phi_0}, s_{\phi_1}, s_{\hat{\epsilon}}$

- Verification:

   i. reconstruct $\widehat{N_V}$ as $N_V^c \cdot \zeta^{s_{\phi_0}+2^{l_f}}$

   ii. reconstruct $\widehat{A'}$ as $\left(Z \cdot A'^{-2^{l_e-1}}\right)^{-c} \cdot R_0^{s_{\phi_0}} \cdot R_1^{\phi_{s_1}} \cdot S^{s_{\hat{v}}}$

   iii. recompute c

   iv. change the range of all elements

   v. check whether N_V is on the rouge list

FPGA

to be expanded here

RFID security

Distance bounding

many applications require that the RFID device communicates with a terminal in a close proximity. Normally, a device can talk only with a terminal that is physically close. But there are attacks:

- delay short, standards leave plenty of time - long timeouts (e.g. 5s)
- NFC devices can be used (emulate cards and readers)
- practical attacks on biometric passport, studentID, Mifare classic, world cup tickets 2006

Attack types

- distance fraud: a sole dishonest prover convinces the verifer that he is at a shorter distance than he really is
    - e.g. with a strong signal from remote place
- mafia attack: the prover is honest, but an attacker tries to modify the distance that the verifer establishes by interfering with their communication.
    - a relay between a genuine card and a genuine terminal
    - the relay consists of two device: one close to the terminal and one close to card

    - only relaying messages. If fast then the card believes it is close to the terminal
    - communicating with a sales terminal on one side and with a remote clients card on the other side - enabling payment that is supposed to work only from a small distance
- terrorist attack: the dishonest prover colludes with another attacker that is closer to the verifer, to convince the verifier that the distance to the prover is short.
- distance hijacking: dishonest prover convinces the verifer that it is at short distance exploiting the presence of an honest prover

Distance bounding protocols

takes response time or signal strength

A basic protocol:

1. for $i=1$ to n (rapid bit exchange)
    - verifier: start clock, a random C_i sent to the prover
    - prover: send a random R_i to the verifier
    - verifier: stop clock
2. prover: signs $C_1, R_1, ..., C_n, R_n$
3. verifier: checks the signature and checks the maximal time distance between sending C_i and receiving R_i (should be within some security bounds)

Attack (problem): a rogue prover can send R_i before getting C_i

Solution (Chaum-Brands):

1. verifier chooses at random n bits α_i
2. prover chooses at random n bits m_i
3. prover creates a commitment for the bits m and sends it to the verifier
4. rapid bit exchange: for $i=1$ to n
    - verifier: sends α_i to the prover
    - prover: $\beta_i = m_i \operatorname{xor} \alpha_i$ and sends β_i to the verifier
5. prover: signs $\alpha_1\beta_1\alpha_2\beta_2....\alpha_n\beta_n$, sends the signature and the opening to the commitment
6. verifier: checks the signature and checks the maximal time distance between sending α_i and receiving β_i (should be within some security bounds), check the commitment

Hijicking attack:

- intercept the signature sent in step 5 (with jamming of the signal to the verifier)
- send own signature over $\alpha_1\beta_1\alpha_2\beta_2....\alpha_n\beta_n$ and the opening to the commitment

Version based on public key verification (Fiat-Shamir protocol)

(no signatures but proof of knowledge of square root X of Y)

1. prover generates at random n numbers R_i and sends $S_i = R_i^2 \bmod N$ to the verifier
2. prover chooses at random n bits β_i,
3. prover creates a commitment for the bits $\beta_1 \beta_2 .... \beta_n$ and sends the commitment to the verifier
4. verifier chooses at random n bits α_i,
5. rapid bit exchange: for $i = 1$ to n
    - verifier: sends α_i to the prover
    - prover: immediately responds with β_i to the verifier
6. prover: presents an opening to the commitments for $\beta_1 \beta_2 .... \beta_n$, and sends the responses $X^{\alpha_i \mathrm{xor}\, \beta_i} R_i$ (which is either $X \cdot R_i$ or R_i depending whether $\alpha_i = \beta_i$)
7. verifier: checks the response (by squring it and comparing with $S_i Y$ or S_i) and checks the maximal time distance between sending α_i and receiving β_i (should be within some security bounds), check the commitment

Problem of noise and transmission errors

- many transmissions lost, so the protocol might be instable
- asymmetric cryptography
- mafia attack suceeds with pbb 2^{-n}

Hacke-Kuhn

(kind of standard protocol in this area)

- the prover is weak, the verifier is strong (terminal)
- the prover and the verifier share a secret key K
- steps:
    1. the verifier sends a nonce N_V
    2. the prover sends a nonce N_P
    3. the prover calculates $R = h(K, N_V, N_P)$, and splits R into two halves: R_1, R_2,
    4. verifier chooses $C_1, ..., C_n$ at random
    5. rapid bit exchange: for $i = 1$ to n:
        a) the prover sends C_i

b) the verifier responds with the ith bit of R_{C_i}

6. the verifier checks the bits received

problem: chance of $\left(\frac{3}{4}\right)^n$ to succeed if the distance is bigger but the adversary knows K: the answers sent in advance and for about $n/2$ positions the bits of R_0, R_1 are the same, only for $\frac{1}{2}n$ bits we need to guess the answer in advance

options:

void slots: the third pseudorandom sequence that determines if in a given time slot to send a message or not

(asking for a response in a wrong time will be detected)

Swiss-knife:

shared secret x, the reader has a database of the tags and their keys, the protocol achieves mutual authentication and distance bounding

1. the reader chooses a random nonce N_A and transmits it to the tag
2. the tag chooses a random N_B, computes a temporary key $a = f_x(N_B, \text{system param})$ using its permanent secret key x
3. the tag splits his permanent secret key x in two shares by computing $Z^0 = a$, $Z^1 = a \operatorname{xor} x$
4. the tag transmits N_B to the reader
5. rapid bit exchange: for $i = 1$ to n perform (the response time measured each time)
    i. the reader chooses a bit c_i and sends to the tag
    ii. the tag responds with $r_i = Z^i_{c_i}$
6. final phase (it is slow, time not measured):
    i. the tag computes $t_B = f_x(c_1, ..., c_n, \text{ID}, N_A, N_B)$
    ii. the tag transmits t_B and $c_1, ..., c_n$ it received during the rapid bit exchange phase.
    iii. the reader searches its database for (ID, x) such that $t_B = f_x(c_1, ..., c_n, \text{ID}, N_A, N_B)$; it recomputes Z^0, Z^1.
    iv. the reader checks the responses for:
        - the number of errors for c_i in response of the tag
        - the number of bad responses r_i
        - the number of too slow responses

        the total numer should be small. There is a threshold above which the reader rejects
    v. the reader responds with $t_A = f_x(N_B)$

vi. the tag checks t_A

RFID privacy

purpose: provide a system where a (passive) eavesdropper cannot trace the tags and overwrite them.

EPCGen2 standard

hardware:

- UHF 860-960 MHz
- CRC (cyclic redundancy code), fixed based on polynomial of degree 16,
- a (poor) PRNG (pbb of single 16bit strings close to 2^{-16}, repetition 10.000 tags of the same pseudorandom sequences $< 0.1\%$, predicting the next 16bits based on previous ones $< 0.025\%$
- no hash function, half-duplex communication
- basic functions: select tag, inventory, access (reading/writing)

inventory:

1. query to tag - number q in the range [1..15]
2. each tag chooses a q bit random number t
3. the tag waits time t with QUERYrep
4. the tag chooses a random number from RN16
5. reader ack with one RN16
6. the chosen tag responds with EPC data

tag memory: 32 bit kill password, 32 bit write password, EPC id (32-96 bits), CRC bits, TID memory - parameters for readers, user memory

privacy:

- session unlinkability (two sessions, hard to say if belong to the same tag if the first tag updated its state in the meantime),
- forward privacy (past interactions are safe if state of the tag revealed after refreshing),
- backward privacy (future interactions after refreshing safe are safe)

EPC global network architecture

→ ONS: Object Naming Service: return address of the EPC issuer

→ EPCDS: EPC Discovery Service: returns addresses of the parties holding events (time, location, ...) on one EPC

$\rightarrow$ EPCIS: EPC Information Service - database run by RFID management with events data

essential problem: access to EPCIS data not under strict control

Main security issues:

i. tag/reader mutual authentication

ii. key distribution

iii. path authentication

iv. clone detection

Anti-cloning solutions

OSK Internal Hash Chain

- two hash functions: G and H
- each tag shares a secret with a backend database, entries ID_i, k_i^1
- secret on the tag updated at each authentication, key stored k_i^j
- execution:
    i. current password shown $G\big(k_i^j\big)$, backend database searches for a matching entry

    ii. secret update: $k_i^{j+1} := H\big(k_i^j\big)$

Dimitriou

i. the reader chooses r_1 at random and sends to the tag

ii. the tag generates r_2 at random, transmits $H\big(k_i^j\big), r_2, H_{k_i^j}(r_2, r_1)$

iii. the reader computes $k_i^{j+1} := G\big(k_i^j\big)$, and sends $H_{k_i^{j+1}}(r_2, r_1)$,

iv. the tag verifies $H_{k_i^{j+1}}(r_2, r_1)$, if ok, then updates $k_i^{j+1} := G\big(k_i^j\big)$

against cloning and tracing! But the backend server might be ahead of the tag

Flyweigth protocol

- system holding data loosely synchronized with tags
- for each tag entry in a database
- each tag shares with database a synchronized state of RNG, they share at least one number at all times
- tag and server get mutually authenticated by sending 3 messages (or at most 5 if something goes wrong)

- tag stores: (RN1 ;RN2 ; $\mathrm{ID}_{\mathrm{tag}}$; g_{tag}; K^r;cnt) where g_{tag} is the current state, K^r the key for refreshing, and a bit flag cnt
- database stores: ($\mathrm{RN1}^{\mathrm{cur}}$;$\mathrm{RN1}^{\mathrm{next}}$;RN2 ;RN3 ;RN4 ;RN5 ; $\mathrm{ID}_{\mathrm{tag}}$; g_{tag} ;K^r ; cnt')
- RN1 is either $\mathrm{RN1}^{\mathrm{cur}}$ or $\mathrm{RN1}^{\mathrm{next}}$
- update procedure on the server: $\mathrm{cnt}' \leftarrow 0$, $\mathrm{RN1}^{\mathrm{cur}} \leftarrow \mathrm{RN1}^{\mathrm{next}}$, draw the next numbers from the RNG: RN2 ;RN3 ;RN4 ;RN5;$\mathrm{RN1}^{\mathrm{next}}$
- alarm is a flag for detection of replay attacks on the side of the tag, it takes the value of the previous cnt
- alarm' is analogos on the side of the server
- cnt (cnt' for the server) is a flag that indicates whether the update has finished successfully

1. Reader $\rightarrow$ Tag : Query

   Tag : set alarm $\leftarrow$ cnt, cnt $\leftarrow 1$. Broadcast RN_1.

2. Tag $\rightarrow$ Reader $\rightarrow$ Server: RN_1

   Server :

   i. Check if RN1 is in Database.

   ii. If $\mathrm{RN}_1 = \mathrm{RN}_1^{\mathrm{cur}}$ for an item in Database,

   then $\mathrm{alarm}' \leftarrow \mathrm{cnt}'$, $\mathrm{cnt}' \leftarrow 1$, and return RN_2

   else if $\mathrm{RN}_1 = \mathrm{RN}_1^{\mathrm{next}}$ for an item in Database, then set $\mathrm{alarm}' \leftarrow 0$,

   update, and return RN_2

   else abort

   iii. Server $\rightarrow$ Reader $\rightarrow$ Tag: RN_2

   Tag :

   i. Check RN_2, if wrong then abort

   ii. If RN_2 is correct then draw five successive numbers from g_{tag} , assign them to the variables RN3, RN4, RN5 (volatile), RN1 , RN2, and set cnt $\leftarrow 0$

   iii. If alarm $= 0$ then return $\mathrm{RN} = \mathrm{RN}_3$

   iv. Else return $\mathrm{RN} = \mathrm{RN}_4$

   iv. Tag $\rightarrow$ Reader $\rightarrow$ Server: RN

   Server :

   a) If $\mathrm{RN} = \mathrm{RN}_3$ and $\mathrm{alarm}' = 0$, then update and

   accept the tag as the authorized Tag

   b) Elseif $\mathrm{RN} = \mathrm{RN}_4$, then return RN_3 , store RN_5 and update.

   c) else abort

v. Server → Reader → Tag: RN_3

   Tag :

   a) Check RN_3

   b) If RN_3 is correct and alarm = 1 then return RN_5

   c) else abort

vi. Tag → Reader → Server: RN_5

   Server :

   a) Check RN_5

   b) If RN_5 is correct then update and accept the tag as the authorized Tag

   c) else abort

Lightweight devices

Practical limitations

traget devices – electronic Product codes etc:

- price $0.05 to $0.10
- Application-specific Integrated Circuits (ASICs)
- printed, designed with tools Hardware Description Languages
- usually passively powered
- like Electronic Product Codes (EPCs)

area:

- a lot of depends on interconnections, however ...
- Measured in Gate Equivalents (GEs): 1 GE = area of a two-input NAND gate.
- higher area - negative and positive influence on the speed
- higher area $\Rightarrow$ higher cost
- 2000 GEs an upper limit for really simple devices

- examples: AES 3400 GE. More lightweight encryption: below 1000GEs

Nonvolatile non-permanent memory:

- EEPROM is expensive
- limit 2K

Power

- passive devices, limit 10 μW
- big differences depending on particular solutions
- temperature constraints, electromagnetic constraints (e.g healthcare devices)

Clock

- limit 100 kHz
- $\Rightarrow$ number of clock cycles per session 150.000

Communication

- animal ID: 30-300 kHz, bandwidth $<$10kBit/s, distance 0.1-0.5 m
- contactless payment: 300kHz -3Mhz, bandwidth$<$50kbit/s
- access control: 3-30Mhz, bandwidth$<$100kbit/s
- range counting: 300Mhz-3GHz, bandwidth$<$200kbit/s
- vehicle identification, toll gates: 3GHz-30GHz, bandwidth$<$200kbit/s, distance 10 m

RNG

- even if randomness is cheap, still we need circuits to handle it (testing entropy ...)
- no verifiable data
- at most 128 bits per authentication session is a good estimate

Conventional protocol (implementable)

- a key K shared by the reader and the tag
- challenge-response: reader sends a random nonce a, the tag returns $\mathrm{Enc}_K(a)$, the reader decrypts and compares with a
- implementaion: less than 1000GEs, 2.5μW power, 500 clock cycles,
- communication 128 bits
- memory 80 bits
- no RNG needed

A warning: LPN "lightweight protocols":

- secrets: x, y
- round:
  1. prover to verifier: b
  2. verifier to prover: challenge a
  3. prover to verifier: $z = a \cdot x + b \cdot y +$ error-vector
  4. verifier: Checks the Hamming weight of $z - a \cdot x + b \cdot y$
- based on LPN problem: Learning Parity with Noise
- key storage: 2K (ok)
- random bits: over 200K to insure many rounds (disaster)
- communication complexity: over 200K, practical bound at about 30K

Lightweight encryption example: Trivium

- stream cipher, 80 bits key, 80 bits initialization vector, 288 bits of the internal state
- phases: initialization, state update+output of bits
- generation pseudorandom bits:

```
for $i = 1$ to $T$ do
$t_1 := s_{66} + s_{93}$
$t_2 := s_{162} + s_{177}$
$t_3 := s_{243} + s_{288}$
$z_i := t_1 + t_2 + t_3$
$t_1 := t_1 + s_{91} \cdot s_{92} + s_{171}$
$t_2 := t_2 + s_{175} \cdot s_{176} + s_{264}$
$t_3 := t_3 + s_{286} \cdot s_{287} + s_{69}$
$(s_1, s_2, ...., s_{93}) := (t_3, s_1, ..., s_{92})$
$(s_{94}, s_{95}, ...., s_{177}) := (t_1, s_{94}, ..., s_{176})$
$(s_{178}, s_{179}, ...., s_{288}) := (t_1, s_{178}, ..., s_{287})$
```

- initialization:
  - 4 full cycles (4*288 rounds) without giving the output
  - initial state: $(s_1, ..., s_{93}) := (K_3, ..., K_{80}, 0, ..., 0)$, $(s_{94}, ..., s_{177}) := (\mathrm{IV}_1, ..., \mathrm{IV}_{80}, 0, ..., 0)$, $(s_{178}, ..., s_{288}) := (0, ..., 0, 1, 1, 1)$

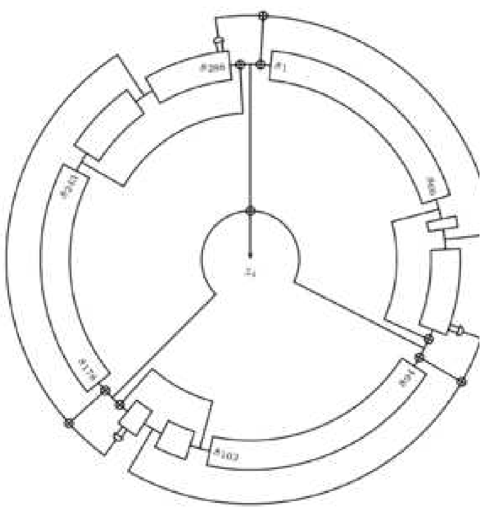


Figure 1.

Side channel analysis

conventional cryptanalysis: input-output data, black-box model (internal values not known, even the code)

side channel cyptanalysis: other data indirectly leaked from the device used as well

side channel information:

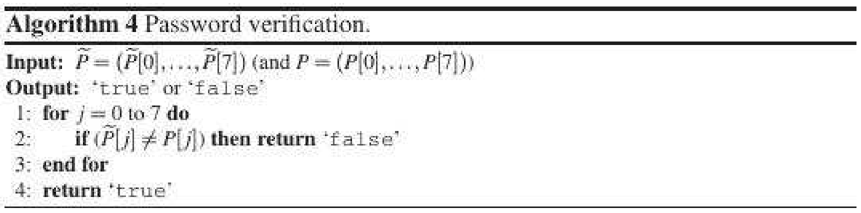
- execution time (e.g. the if-branches may have different execution time)
- electromagnetic radiation (electromagnetic field created by current in a wire, depends on direction)
- power consumption (e.g. consumtion depends on instructions and changes of contents)
- heat
- light, noise, ...
- frequency change and errors (some operations may not finish on time)
- ...

coverage by industrial standards and requirements:

- frequently mentioned
- proprietary countermeasures, patents
- standards such as FIPS are behind the attacks state-of-the-art

TIMING ATTACKS

example: password verification



Algorithm 4 Password verification.

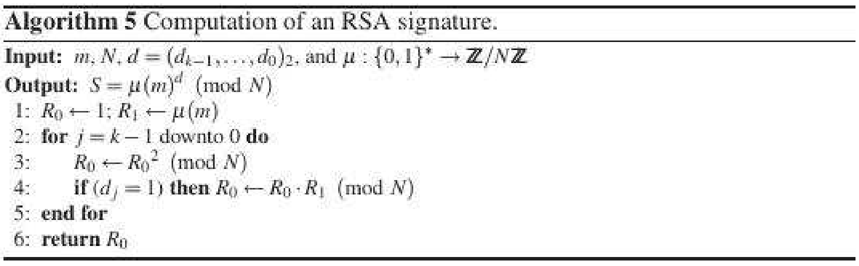
```
Input: $\widetilde{P} = (\widetilde{P}[0], \dots, \widetilde{P}[7])$ (and $P = (P[0], \dots, P[7])$)
Output: 'true' or 'false'
1: for $j = 0$ to 7 do
2:     if ($\widetilde{P}[j] \neq P[j]$) then return 'false'
3: end for
4: return 'true'
```

– propose passwords of the type (b,0,0,...0)

– for the right b the error message comes slightly later

– then proceed with the correctly guessed bytes and guess the next ones

example: RSA



Algorithm 5 Computation of an RSA signature.

```
Input: $m, N, d = (d_{k-1}, \dots, d_0)_2$, and $\mu : \{0,1\}^* \to \mathbb{Z}/N\mathbb{Z}$
Output: $S = \mu(m)^d \pmod{N}$
1: $R_0 \leftarrow 1$; $R_1 \leftarrow \mu(m)$
2: for $j = k-1$ downto 0 do
3:     $R_0 \leftarrow R_0^2 \pmod{N}$
4:     if ($d_j = 1$) then $R_0 \leftarrow R_0 \cdot R_1 \pmod{N}$
5: end for
6: return $R_0$
```

– time depends on the number of multiplications

– the multiplication executed e.g. by Montgomery method that creates the output modulo $2N$ and not N. Subtraction necessary in some cases

– when $d_{k-1}, \dots, d_{k-n+1}$ already known we try to derive d_{k-n}

– guess d_{k-n}

– $L_0 = \{m$: computing $R_0 := R_0 \cdot R_1$ does not require subtraction$\}$

$L_1 = \{m$: computing $R_0 := R_0 \cdot R_1$ requires subtraction$\}$,

– $\tau_i =$ average time for L_i ,

– if $\tau_0 \approx \tau_1$ then the guess was wrong, but we learn d_{k-n}

SIMPLE POWER ANALYSIS

– measuring power consumption: probe+resistor+oscilloscope on power supply or ground

– easy for smart cards etc with external power supply

- models of power consumption:
    - Hamming distance model: consumption= number of changes the bits stored in the memory
    - Hamming weight model: consumption= Hamming weigth of the values stored after the operation

Applications:

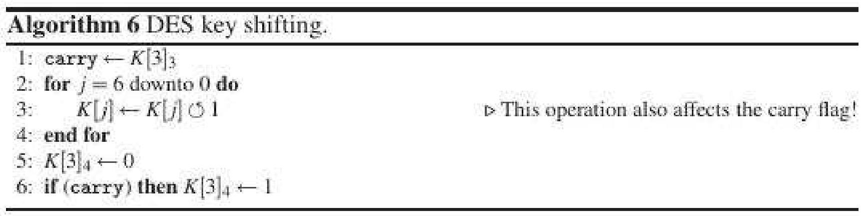
- power traces of operations, reverse engineering the code executed
- trace signatures for operation arguments

example: RSA

- computation starts with squaring a 1 (low consumption)
- multiplication has higher consumption then squaring, so visible if low+high or only high consumption. So the bit of the key derived

example: DES key schedule

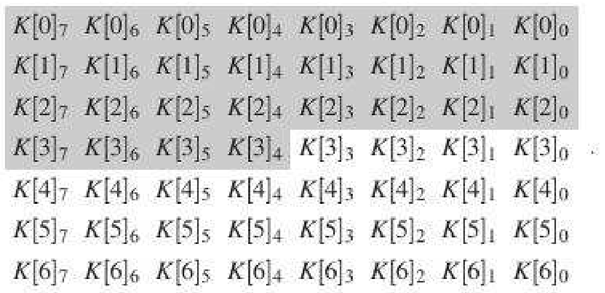
- $K_t = C_t || D_t$, where $C_t := C_{t-1} \ll b$, $D_t := D_{t-1} \ll b$, where b depends on the round



Algorithm 6 DES key shifting.

```
1: carry ← $K[3]_3$
2: for $j = 6$ downto 0 do
3:     $K[j] ← K[j] \circlearrowleft 1$          ▷ This operation also affects the carry flag!
4: end for
5: $K[3]_4 ← 0$
6: if (carry) then $K[3]_4 ← 1$
```

-



$K[0]_7$	$K[0]_6$	$K[0]_5$	$K[0]_4$	$K[0]_3$	$K[0]_2$	$K[0]_1$	$K[0]_0$
$K[1]_7$	$K[1]_6$	$K[1]_5$	$K[1]_4$	$K[1]_3$	$K[1]_2$	$K[1]_1$	$K[1]_0$
$K[2]_7$	$K[2]_6$	$K[2]_5$	$K[2]_4$	$K[2]_3$	$K[2]_2$	$K[2]_1$	$K[2]_0$
$K[3]_7$	$K[3]_6$	$K[3]_5$	$K[3]_4$	$K[3]_3$	$K[3]_2$	$K[3]_1$	$K[3]_0$
$K[4]_7$	$K[4]_6$	$K[4]_5$	$K[4]_4$	$K[4]_3$	$K[4]_2$	$K[4]_1$	$K[4]_0$
$K[5]_7$	$K[5]_6$	$K[5]_5$	$K[5]_4$	$K[5]_3$	$K[5]_2$	$K[5]_1$	$K[5]_0$
$K[6]_7$	$K[6]_6$	$K[6]_5$	$K[6]_4$	$K[6]_3$	$K[6]_2$	$K[6]_1$	$K[6]_0$

-
- step 6 is executed or not!

DIFFERENTIAL POWER ANALYSIS

- statistical difference for power consumption depending on whether at a given moment 0 or 1 processed
- building a statistical model
- testing

Example: AES

- message m and key K are 4x4 matrices
- AddRoundKey and then 10 rounds of: SubBytes, ShiftRows, MixRows, AddRoundKey
- guess the round key $k_{u,v}$ of the first round and check power consumption of Subbytes
- classes corresponding to result of XOR with the key – consumption by SubBytes at (u,v)
- a lot of noise from other operations

More involved statistical tests: checking correlations, ...

FAULT ATTACKS

example RSA with Chinese Remainder Theorem

- instead of $m^d \bmod N$ for $N = p \cdot q$:
  - compute $c_1 = m^d \bmod p$ and $c_2 = m^d \bmod q$
  - reconstruct m^d as $c_1 \cdot q \cdot (q^{-1} \bmod p) + c_2 \cdot (p \cdot (p^{-1} \bmod q) \bmod N$
- error in computation of c_1: the result differs from the correct one by a multiply of p

COUNTERMEASURES

- reducing the side channel signal
- artificial noise of the side channel signal: wait states, dummy opearations, compiling from equivalent subblocks but different characteristics
- unstable clock
- masking: e.g. instead of computing $\mathrm{Hash}(m)^d \bmod N$ compute

  $(\mathrm{Hash}(m) + r_1 \cdot N)^{d + r_2 \cdot \phi(N)} \bmod r_3 N) \bmod N$
- avoid key dependant branching
- checking the output before outputting

SUBVERSION MODEL

- assume the adversary may manipulate software on the embedded device, no way to check it:
  - the problems might be already on the hardware level,
  - restricted access to memory (due to security issues)
  - no testing circuit (due to security issues)
  - logic of the circuit may cheat about the memory contents

  - protection against invasive methods
  - ... but a tiny change may subvert the software into a malicious one

- narrow kleptographic channel in order to leak a secret key K
  - requires some randomness in the protocol and a secret P known to the adversary and the software
  - during the ith protocol execution, choose randomness r (visible to outsiders)
  - compute $(...,b,a) := \mathrm{Hash}(r,P)$
  - if the ith bit of K is b, then proceed, else choose randomness again

- countermeasures:
  - no randomness for protocol execution (hard, the designers love to use randomness as it simplifies the design very much. Major rethinking needed)
  - different key material for each device

    EXAMPLE: smart meters case in Spain:
    - communication between a smart meter and a power grid protected with a secret key
    - symmetric crypto used due to price issues
    - the same key used for all smart meters
    - it suffices to break into one device
    - software updates in smart meters secured with this key
    - how to do it properly:
      - → system key K
      - → a device with the serial number i holds a key $\mathrm{Hash}(i, K)$, the infrastructure recomputes the key when needed
      - → a hierarchy of keys possible
    - open specification necessary (see BSI Richtlinie)
  - key evolution:
    - devices A and B change their shared key after each successful session
    - the change: $K := F(K,i)$ where K is the key and i is the parameter depending on the session (e.g. the session key), F is a "one-way function"
    - the adversary has to observe most/all interactions to keep track on the key changes. If not, then the shared key becomes invisible again
    - key evolution for RSA: $e := e \cdot \delta$ (as integers, public key), $d := d/\delta \bmod \phi(n)$ (private decryption key)

- key pool: random key predistribution: a large pool K, a device holds a subset of cardinality k.
    - projection space with $|K|$ elements, each point holds a key
    - each device holds keys corresponding to points from a line
    - each two lines intersect in exactly one point
    - compromising t devices makes at most t points compromised from a line of a different user

- the problem of key generation:
    - must be randomized in order to avoid key guessing. So the general recommendation does not work
    - the generation process may be focused on a subspace of P such that:
        - for an observer it is infeasible to detect that we are using the keys form a subspace (example: exponents x such that $h^x \bmod p$ is odd while h is secret)
        - the subspace may enable an attack
        - example: attack against RSA
    - countermeasures:
        - "cliptography":
            - standard procedure: secret key generated as $k := E(r, \text{params})$, where r is the random parameter
            - cliptographic: r chosen at random, then $r' = \text{Hash}(r)$, and $k := E(r'$, params)
            - intuition: key derivation may try to get a key from a subset of "weak keys" or "keys with a trapdoor". Difficult to hit such a subset after hashing
            - does not work against bounded size of probability space for r (brute force still works)
        - "co-generation": the user himself can modify the key:
            - for DL based systems: instead of x chosen at random by device and public key g^x
                - → the device chooses x_1 and declares the public key $y_1 = g^{x_1}$
                - → the user chooses x_2, sends x_2 to the device, the public key should be $y = y_1^{x_2}$. Private key $x = x_1 x_2$
            - example: for Pseudonymous Signature from BSI specification:
                - → the key is x_1, x_2 such that $x = x_1 + x_2 \cdot z \bmod q$ for some system secrets x and z.

- → the device has two pairs preinstalled: (x_1', x_2') and (x_1'', x_2'') and takes their linear combination

  $\alpha \cdot (x_1', x_2') + \beta \cdot (x_1'', x_2'')$ for $\alpha + \beta = 1 \bmod q$
- → perfectly secure, since a subspace of dimension 1 in a linear space of dimension 1!
- → verifiable that the device follows the procedure

– multiple devices:

- → protocol executed by more than one device on the side of the user, devices from different providers
- → example: electornic signature from Schnorr scheme:
  1. the 1st device computes $r_1 := g^{k_1}$ for a random k_1
  2. the 2nd device computes $r_2 := g^{k_2}$ for a random k_2
  3. exchange r_1 and r_2,
  4. compute $r := r_1 \cdot r_2$
  5. $e := \mathrm{hash}(r, M)$ for a message M to be signed
  6. the 1st device computes $s_1 := k_1 - x_1 \cdot e$
  7. the 2nd device computes $s_2 := k_2 - x_2 \cdot e$
  8. compose the signature (r, s) where $s := s_1 + s_2$

– hardware separation:

- → case of CAM:
  - → compute $Y := g^y$ for a random y, derive DH key $K := Y_A^y$, then compute and send $w := y/x$, the recipient can check that $X^w = Y$
  - → choose y at random, $Y := X^y$, derive DH key $K := (Y_A^y)^x$, send $w = y$, the recipient can check that $X^w = Y$

  advantage: the second option may implement x in a unit that only answers by raising to power x

model of encapsulated subunits ("Forbidden City")

POWER GRID

- threat: large scale autonomous systems providing stabilization of the power supply network
- difficult issue of balancing the power consumption and production
- local safety measures may create a blackout on a global scale
- poor design of security, black-box solutions, proprietary
- major threat (not only for cyber war)
- recommendation: self-stabilization, local systems that survive