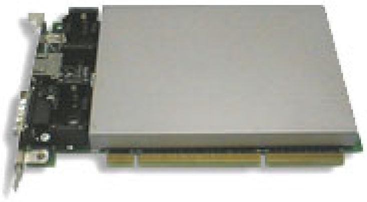
Secure Hardware and PIN Recovery Attacks

Masaryk University in Brno

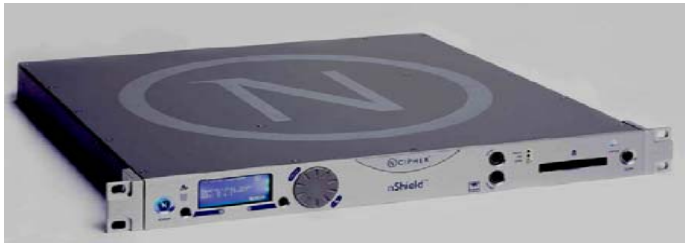
Faculty of Informatics

Jan Krhovják

Vašek Matyáš



Roadmap



- Introduction
  - The need of secure HW
  - Basic terminology
  - Architecture
- Security categories and common attacks
  - Physical security
  - Logical security
  - Environmental security
  - Operational security
- PIN recovery attacks
  - Decimalisation Table Attacks
  - ANSI X9.8 Attacks
  - Basic idea of Collision Attack

Why secure hardware

- Ensure (fast) secure communication and secure storage (of extremely critical data)

- Sensitive data (e.g. financial data, cryptographic keys) stored on hard disk or in memory are vulnerable
  - Adversary (with sufficient rights) can access them
  - Data in memory can be paged out to disk
  - Data in a hard disk can be backed up in unprotected storage device

Where secure hardware

- Critical applications have always been banking transactions
  - Primarily due to need for secure storage
  - In 70's VISA formed worldwide banking ATM network
  - Banks can't trust themselves, their employers or customers
  - This led to evolution of so-called Hardware Security Modules and financial data networks (banking machines, sales terminals, etc.)

- Certification authorities
  - Primarily due to need for accelerating crypto operations
  - Increase in the last decade for public-key cryptography support

Basic terminology

- Hardware security modules (HSM)
    - Coprocessors
    - Accelerators
    - Cryptographic smartcards
- Host devices, API
- Attacks on HSMs
    - Physical attacks
    - Side channel attacks
    - Attacks on and with API
    - We are not interested in any form of DoS attacks!
- Top-level crypto keys – always stored inside HSM
    - Other keys can be stored outside HSM encrypted by these

Architecture of cryptographic coprocessors/accelerators

- Come out from classical von Neumann architecture
  - Mechanisms of physical protection
    - Steel shielding, epoxy resin, various sensors

  - Generators of true random numbers
    - Generating cryptographic material (e.g. keys, padding values)
    - Algorithmic counter-measurements against side channel attacks
  - Special coprocessors
    - Accelerating both symmetric and asymmetric crypto

  - Non-Volatile RAM (NVRAM) => retains its content
    - Connected to a constant power source or battery
    - Storing sensitive data (e.g. master key)
  - I/O circuits

- Easy verification

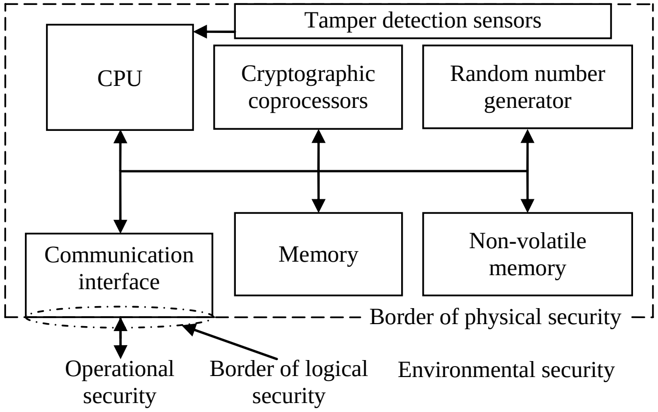
Architecture of cryptographic smartcards

- Similar building blocks as coprocessors/accelerators
  - Everything is inside a single integrated chip
    - Problems with limited silicon area => only small size of RAM
  - There is only limited power supply in mobile devices
    - New (U)SIM cards supports DES, RSA and EC cryptography
    - Their power consumption must be very small
  - Operating system is stored in ROM, applications in EEPROM
- Division according to the communication interface
  - Contact – contain contact pads =>
  - Contactless – contain an embedded antenna
  - Combined – single chip with both previous interfaces
  - Hybrid – more chips (and interfaces) on single card

- Super smartcard =>

Security categories

- Physical security
  - Technologies used to safeguard information against physical attack
  - Barrier placed around a computing system to deter unauthorized physical access to the computing system itself
    - Tamper: evidence, resistance, detection, response (more on the next slide)
- Logical security
  - The mechanisms by which operating systems and other software prevent unauthorized access to data
    - Access control, algorithms, protocols
- Environmental security
  - The protection the system itself
    - Access policies – guards, cameras …
- Operational security



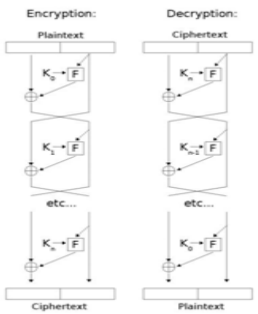
Physical security

- Tampering – the unauthorized modification of device
- Tamper evidence
  - The evidence is left when tampering occurs
  - Chemical or mechanical mechanisms
- Tamper resistance
  - Only to certain level!
  - Chemically resistant material, shielding
- Tamper detection
  - Special electronics circuits (i.e. sensors)
- Tamper response
  - Detection => destroying all sensitive information
  - Erasing/rewriting/memory destruction

Physical attacks

- Invasive attacks (passive or active)
  - Direct access to embedded components (ALU, bus, memory …)
    - Micro probing – observing, manipulating or interfering the device/chip
    - Reverse engineering – the process of analyzing an existing system to identify its components and their interrelationships
    - Memory readout techniques (e.g. freezing and probing)
      - Freezing by liquid nitrogen can increase data retention time in RAM to hours
  - They require a lot of time, knowledge and specialized equipment
- Semi-invasive attacks (only on integrated chip cards)
  - Depackaging the chip, but the passivation layer remains
    - Utilizing UV light, X-rays, laser, electromagnetic field, local heating
    - Optical fault induction – illumination of SRAM can change its content
  - They require only low-cost equipment
  - Easy reproduction of prepared attack for the same HW, FW, SW

Logical security



Feistel Cipher

- Access control
  - The assumption is existence of trusted environment
- Cryptographic algorithm
  - Mathematical functions – only keys should be secret
  - Ensuring confidentiality, integrity, authentication …
- Cryptographic protocols
  - Distributed algorithms – sets of three to ten messages
  - Their single steps are created by calling of API functions
    - API is the only one (exactly defined) communication interface between HSM and the host application
    - Economy prevails security – too many supported standards in APIs
    - API of HSM thus contains hundreds functions with many parameters

      => very big space for errors and formation of attacks

Logical attacks

- Non-invasive attacks
  - No physical damaging of device
  - Monitoring/eavesdropping
    - TEMPEST attacks
      - Electronic devices emits electromagnetic radiation
      - Reconstructing data from electromagnetic radiation
    - Side channel attacks
      - Timing analysis – measuring the time of cryptographic operations with respect to input data and algorithm implementation
      - Power analysis – measuring the fluctuations in the consumed current when the device is performing specific operations
      - Fault analysis – generating of glitches (in voltage, clock signal …)
  - Software attacks on and with API
    - No specialized equipment needed
    - They are very fast – taking only a couple of seconds

Environmental security

- The asset is the device itself (not the stored information)
  - At least interesting aspect of security from analysis perspective
  - The goal is to limit attacker's opportunity to initiate an attack by creating layers of hindrance (e.g. access policies, controls)
  - Not necessarily applicable to HSMs operating in hostile environments (they are typically highly physically secured)
    - The exception are the administrators of HSMs (i.e. security officers)
    - They have a certain amount of power over a HSMs that can be misused
    - To prevent single security officer from compromising the system, the principle of dual control policy is enforced
      - At least two security officers (e.g. from different banks) must agree to change the device configuration (e.g. installing/changing of keys)
      - At least two security officers must collude to circumvent the security
  - Administrative/procedural controls should be the part of security policy whenever is it possible

Operational security

- HSM can be operated only trough functions of API
  - With API functions can programmer interact by keyboard
  - Some devices allows the user to execute limited number of exactly defined API commands (e.g. ATMs by PINpad/keypad)
- The security risks related to proper manipulation with cash machines and their interfaces are growing
  - The user should be able to recognize the fake
    - Payment terminal, ATM, card reader =>
  - The user should know what he do with keypad
  - The user should operate cash machine alone
  - The user should be aware of latest attacks as
    - Transparent overlay of keypad, Lebanese loop =>
  - The user should safeguard his PIN

Attacks on and with API

- Examples of commonly used API
  - Public Key Cryptographic Standard (PKCS) #11
  - Common Cryptographic Architecture (CCA)
- Three major problems of cryptographic API
  - Insufficient ensuring integrity of keys
    - Problems with backward compatibility (e.g. support of DES or RC2)
    - Meet in the Middle Attack, 3DES Key Binding Attack, Conjuring Keys …
  - Insufficient checking of function parameters
    - Banking API and working with PINs => PIN recovery attacks
    - Decimalisation Table Attacks, ANSI X9.8 Attacks …
  - Insufficient enforcing of security policy
    - PKCS #11 – only set of functions, designed for one-user tokens

Example of attack on API: Conjuring Keys From Nowhere

- Unauthorized generating of keys stored outside HSM
  - Random value of encrypted key is given to HSM
    - Older HSMs used this technique to legitimate key generation
    - Today is it considered as attack
  - After decryption is the value of key also random
    - In the case of DES has with probability $1/2^8$ good parity
      - DES key is stored with odd parity – LSB in each octet is parity bit
    - In the case of two-keyed 3DES-2 has a good parity with probability $1/2^{16}$ (and this is still achievable)
  - These keys can served to form more complicated attacks
- The defense lies in carefully designed key formats

  => e.g. add before encryption checksum + timestamp
- Next part of presentation: **PIN recovery attacks**

PIN Generation and Verification

- Terminology
  - PIN, Personal Account Number (PAN)
  - Clear PIN block (CPB); Encrypted PIN block (EPB)
- Techniques of PIN generation and verification
  - IBM 3624 and IBM 3624 Offset
    - Based on validation data (e.g. account no. – PAN)
    - Validation data encrypted with *PIN derivation key*
    - The result truncated, decimalised => PIN
    - IBM 3624 Offset – decimalised result called IPIN (Intermediate PIN)
    - Customer selects PIN: Offset = PIN – IPIN (digits mod 10)
  - Verification process is the same
    - result is compared with decrypted EPB (encrypted PIN from cash-machine)

PIN Verification Function

- Simplified example of verification function and its parameters:
  1. PIN (CPB) encryption/decryption key
  2. PIN derivation key – for PIN generation process
  3. PIN-block format
  4. validation data – for PIN extraction from EPB (e.g. PAN)
  5. encrypted PIN-block
  6. verification method
  7. data array – contains decimalisation table, validation data and offset
- Clear PIN is not allowed to be a parameter of verification function!

PIN Verification – IBM 3624 Offset

- Inputs – (4-digit PIN)
  - PIN in EPB is `7216` (delivered by ATM)
  - Public offset (typically on card) – `4344`
  - Decimalisation table – `0123 4567 8901 2789`
  - Personal Account Number (PAN) is `4556 2385 7753 2239`
- Verification process
  - PAN is encrypted => `3F7C 2201 00CA 8AB3`
  - Truncated to four digits => `3F7C`
  - Decimalised according to the table => `3972`
  - Added offset `4344`, generated PIN => `7216`
  - Decrypt EPB and compare with the correct PIN

Decimalisation Table Attacks I

- Attacks utilising known PINs
  - Assume four-digit PINs and offset `0000`
  - If decim. table (DT) is `0000 0000 0000 0000`
    generated PIN is always `0000`
  - PIN generation function with *zero* DT outputs EPB with PIN `0000`
  - Let D_{orig} = `0123 4567 8901 2345` is original DT
  - D_i is a *zero* DT with "1" where D_{orig} has *i*
    e.g. D_5 = `0000 0`**`1`**`00 0000 000`**`1`**
  - The attacker calls 10x verification function with EPB of `0000` PIN and with D_0 to D_9
  - If *i* is not in PIN, the "1" will not be used and verification against `0000` will be successful

Decimalisation Table Attacks II

- Results
  - All PIN digits are discovered
  - PIN space reduced from 10^4 to 36 (worst case)

- Extended attack without known PINs
  - Assume, that we obtain customers EPB with correct PIN
  - D_i are DTs containing $i-1$ on positions, where D_{orig} has i e.g. D_5 = `0123 4`**`4`**`67 8901 234`**`4`**
  - Verification function is called with intercepted EPB and D_i
  - Position of PIN digits is discovered by using *offset* with digits incremented individually by “1”
    - Bold “**4**” changes to “5”

DT Attacks – Example

- Let PIN in EPB be `1492`, offset is `1234`
  - We want to find position of “2”
  - Verification function with D_2 results in `1491!=1492` => fails
  - Offsets `2234`, `1334`, `1244`, `1235` increment resulting generated PIN (`2491,1591,`…)
  - Eventually the verification is successful with the last offset => 2 is the last digit
- To determine four-digit PIN with different digits is needed at most 6 calls of verification function

Clear PIN Blocks

- Code Book Attacks and PIN-block formats
  - => clear PIN blocks (CPB)

p – PIN digit
r – random digit
x – arbitrary, all the same
F – 0xF digit

- ECI-2 format for 4 digits PINs
  - `ECI-2 CPB = pppprrrrrrrrrrrr`
- Visa-3 format for 4–12 digits PINs
  - `Visa-3 CPB = ppppFxxxxxxxxxxx`
- **ANSI X9.8 format for 4–12 digits PINs**
  - P_1 = `Zlppppffffffff FF`
  - P_2 = `ZZZZaaaaaaaaaaaa`
  - ANSI X9.8 CPB = P_1 xor P_2

Z – 0x0 digit
l – PIN length
f – either “p” of “F”
a – PAN digit

ANSI X9.8 Attacks I

- Attacking PAN with translation & verification functions – input parameters (key K, EPB, PAN)
  - Functions decrypt EPB & extract PIN
    `CPB xor` P_2 `= 04ppppFFFFFFFFFF` => PIN = `pppp`
  - Extraction tests PIN digits to be `0–9`!
  - If a digit of PAN is modified by `x`
    - $P_2' = P_2$ `xor 0000x00000000000`
    - `CPB xor` P_2'`=   04ppppFFFFFFFFFF xor`
      `xor   0000x00000000000`
      it means that PIN = `pppp xor 00x0`
    - If `p xor x < 10` function ends successfully, otherwise function fails

ANSI X9.8 Attacks II

- The sequence of (un)successful function calls can be used by attacker to identify `p` as a digit from set `{p, p xor 1}`
- For example if PIN digit is `8` or `9`, then this sequence will be PPFFFFFFPPPPPPPP, where P is PASS, F is FAIL and `x` is incremented from `0` to `15`

- Only last two PIN digits can be attacked
- PIN space is reduced from 10^4 to 400
- This attack can be extended to all PIN digits

ANSI X9.8 Attacks III

- Attack against PIN translation functions
  - Input/output PIN-block format can be modified
  - Consider ANSI X9.8 EPB with null PAN (wlog)
    - Attacker specifies input format as VISA-3 and output as ANSI X9.8
    - PIN is then extracted from `04ppppFFFFFFFFFF` as `04pppp`
    - `04pppp` is formatted into ANSI X9.8 CPB as `0604ppppFFFFFFFF` and encrypted
  - Attacker has EPB with six-digit PIN and can use previous attack to determine all 4 digits of original PIN

- PIN space is reduced from 10^4 to 16

ANSI X9.8 Attacks IV

- PIN can be also determined exactly
  - The attacker needs to be able to modify PAN
    - This is impossible if input format is Visa-3
    - PAN modification must be done earlier (in EPB)
  - Let's modify second digit of PAN by `x`
    - Input format is VISA-3 and output ANSI X9.8
    - PIN is decrypted from ANSI X9.8 EPB and extracted as `04pppp xor 00000x`
    - If `x = p xor F` (i.e. `x xor p = F`) then PIN is extracted as `04ppp` and formatted into ANSI X9.8
    - This can be detected by/during translation back to VISA-3 format EPB

ANSI X9.8 Attacks – Collision Attack (Basic Idea)

- Assuming well designed API (e.g. DT is fixed)
- Attack allows to partially identify last two PIN digits
  - Basic idea (simple example with one-digit PIN&PAN)

PAN	PIN	xor	EPB	PAN	PIN	xor	EPB
0	0	0	21A0	7	0	7	2F2C
0	1	1	73D2	7	1	6	345A
0	2	2	536A	7	2	5	0321
0	3	3	FA2A	7	3	4	FF3A
0	4	4	FF3A	7	4	3	FA2A
0	5	5	0321	7	5	2	536A
0	6	6	345A	7	6	1	73D2
0	7	7	2F2C	7	7	0	21A0
0	8	8	**4D0D**	7	8	F	**AC42**
0	9	9	**21CC**	7	9	E	**9A91**

  - Attacker knows for each PAN only the set of EPBs

Conclusions

- Secure hardware
  - Limited functionality – easier to verify – better security (than multipurpose hardware)
  - Dedicated circuits – faster than software implementation
- Secure hardware doesn't guarantee absolute security
  - Any secure hardware can be reengineered
  - Main reason of its usage is increased cost of attack
- Bad design and integration imply attacks
  - The security of current generation banking APIs is really bad with respect to insider attacks
  - Number of (banking) standards implemented ensures interoperability but also causes errors