

Tamper resistant devices

*Foundations of Secure e-Commerce
(bmevihim219)*

Dr. Levente Buttyán

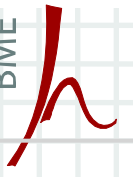
Associate Professor

BME Hálózati Rendszerek és Szolgáltatások Tanszék

Lab of Cryptography and System Security (CrySyS)

buttyan@hit.bme.hu, buttyan@crysys.hu





Outline and objective

- outline
 - introduction
 - types and applications of tamper resistant devices
 - FIPS 140
 - the IBM 4758 coprocessor
 - attacking tamper resistant devices (including API attacks)

- the objective is to understand
 - what tamper resistance means ?
 - what kind of tamper resistant devices exist ?
 - how they are attacked ?

- useful readings:
 - R. Anderson, M. Bond, J. Clulow and S. Skorobogatov, Cryptographic processors – a survey, Technical Report No. 641, University of Cambridge, Computer Laboratory, UK, 2005.
 - S. Smith, S. Weingart, Building a High-Performance, Programmable Secure Coprocessor, IBM Research Report 21102, February 1998.

- from Wikipedia:

Tamper-resistant microprocessors are used to store and process private or sensitive information, such as private keys or electronic money credit. To prevent an attacker from retrieving or modifying the information, the chips are designed so that the information is not accessible through external means and can be accessed only by the embedded software, which should contain the appropriate security measures.

Examples of tamper-resistant chips include all secure cryptoprocessors, such as the IBM 4758 and chips used in smartcards, as well as the Clipper chip.

- Smith and Weingart:

Secure coprocessors enable secure distributed applications by providing **safe havens** where an application program can execute (and accumulate state), free of observation and interference by an adversary with direct physical access to the device.

Classes of tamper resistan

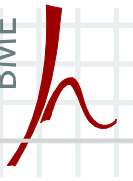


- high-end devices:
 - e.g., IBM 4758 coprocessor
 - powerful crypto engine surrounded by a tamper-sensing mesh
 - device erases its key material and renders itself inoperable if a tampering attempt is detected

- low-end devices:
 - e.g., cheap microcontrollers
 - typically capable only for symmetric key crypto
 - their read-protection mechanisms are not really designed to withstand skilled and determined attacks

- mid-range:
 - e.g., smart cards and TPM chips
 - single-chip products hardened against physical attacks

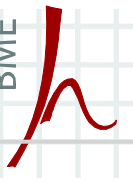




FIPS 140

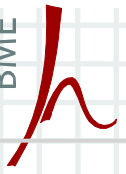
- benchmark standard that specifies the security requirements for cryptographic modules

- types of requirements considered:
 - physical security (locks, seals, coatings, covers, tamper detection and response)
 - ports and interfaces
 - operational environment (OS requirements)
 - finite state model (states and state transitions)
 - roles, services, and operator authentication
 - cryptographic module specification (algorithms, modes of operation, description of HW, SW, FW, security policy)
 - key management (random number generation, key generation and storage, key erasure)
 - EMI/EMC
 - self-tests
 - design assurances (configuration management, delivery and operation, development)



FIPS 140 security levels

- level 1
 - **no physical security** mechanisms are required in the module
 - basic requirements on cryptographic algorithms
 - example: encryption software on a PC
- level 2
 - needs **tamper evident** coating or seals
 - requires role based access control
- level 3
 - needs **tamper resistant** physical security
 - requires identity based access control
 - parameters must either be entered into or output from the module in encrypted form or be directly entered into or output from the module using split knowledge procedures
- level 4
 - highly reliable **tamper detection and response** (immediately erasing all secret data)
 - protection against a compromise due to environmental conditions or fluctuations outside of the normal operating ranges (e.g., voltage, temperature, ...)



Application areas

- ATM security
- Internet banking, electronic payment
- Automated Fare Collection (AFC)
- prepayment electricity meters
- Trusted Computing (TC)
- Public Key Infrastructures (PKI)
- military applications
- other

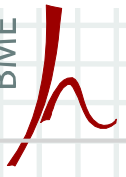
- customer PIN is derived from account number and a PIN derivation key
- the PIN derivation key needs to be protected against unauthorized disclosure (including insiders such as bank personnel)
- PIN is stored in an encrypted form on the customer's card
- verification at an ATM needs either the PIN decryption key or secure communication of the PIN from the ATM to the bank
- in both cases, keys need to be protected against unauthorized disclosure (including insiders such as ATM maintenance personnel)

Internet banking, e-payment

- some banks require their customers to use off-line tokens to produce a time-dependent password or a response to a challenge
 - example: RSA SecurID tokens



- smart cards can be used as purses to store electronic money
- balance should be protected from unauthorized modifications
- must also support protocols to transfer value between two purses (e.g., those of a user and a merchant), which need crypto keys that should be protected from disclosure (in order to prevent forging a transaction)
 - example: Mondex

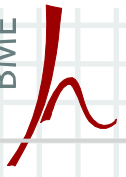


Automated Fare Collection

- electronic ticketing for public transportation
 - all transactions can be logged, collected, and analyzed
 - efficiency of the system can be increased by careful planning of schedules

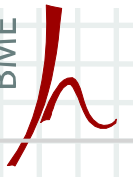
- e-tickets may store value, in which case they must be protected from manipulation

- e-tickets may need to be authenticated during the validation process
- authentication is based on a key derived from the ticket ID and a master key (key diversification)
- master key must be stored in off-line ticket validating equipment (e.g., on buses, trams, etc)
- master key needs to be protected from disclosure when equipment is stolen



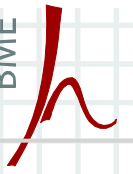
Trusted Computing

- computing platforms, such as PCs and PDAs, are envisioned to be equipped with an embedded cryptoprocessor, the Trusted Platform Module (TPM)
- the TPM (together with a microkernel) can certify both a program and the platform on which it is executing
 - viruses and other forms of malware cannot easily propagate
- the major application is DRM (Digital Rights Management):
 - in this context, the TPM enforces usage policies
 - e.g., a TC machine can assure a content vendor that it is sending a song or movie to a true copy of a media player program, rather than to a hacked copy
 - TC may also enable alternative marketing strategies, such as subscription services for listening to music
 - note that current DRM mechanisms are based on software obfuscation, and eventually get hacked



Public Key Infrastructures

- private keys of users and CAs must be protected from disclosure
- the private keys of users are typically stored on smart cards
 - the smart card can control the usage of the private key
 - the smart card can run cryptographic algorithms such that the private key does not need to leave the protected environment
- CA private keys are held in tamper resistant HSMs (Hardware Security Modules)
- the HSMs may help enforce stringent policies on key usage:
 - they can enforce dual control policies on the most valuable keys
 - they can help supervisors monitor the activities of large numbers of human operators efficiently
 - they can keep signed audit trails of activities to allow retrospective monitoring of access



Military applications

- dishonest insiders are a real threat
- modern military cipher machines use classified algorithms in tamper-resistant chips
- in addition, crypto keys are often transported in tamper resistant hardware between sites

- nuclear command and control systems
- weapons should be armed only by authorized parties and under well-defined circumstances
 - authorization is based on cryptographic codes, the verification of which needs crypto keys to be stored safely in weapons
 - example for a condition that is easy to measure yet hard to forge is the period of zero gravity experienced by an air-drop bomb on release
- tamper-resistance mechanisms are embedded in weapons in order to prevent a stolen weapon being exploded, or being dismantled to reveal an authorization code with which a second stolen weapon could be armed.

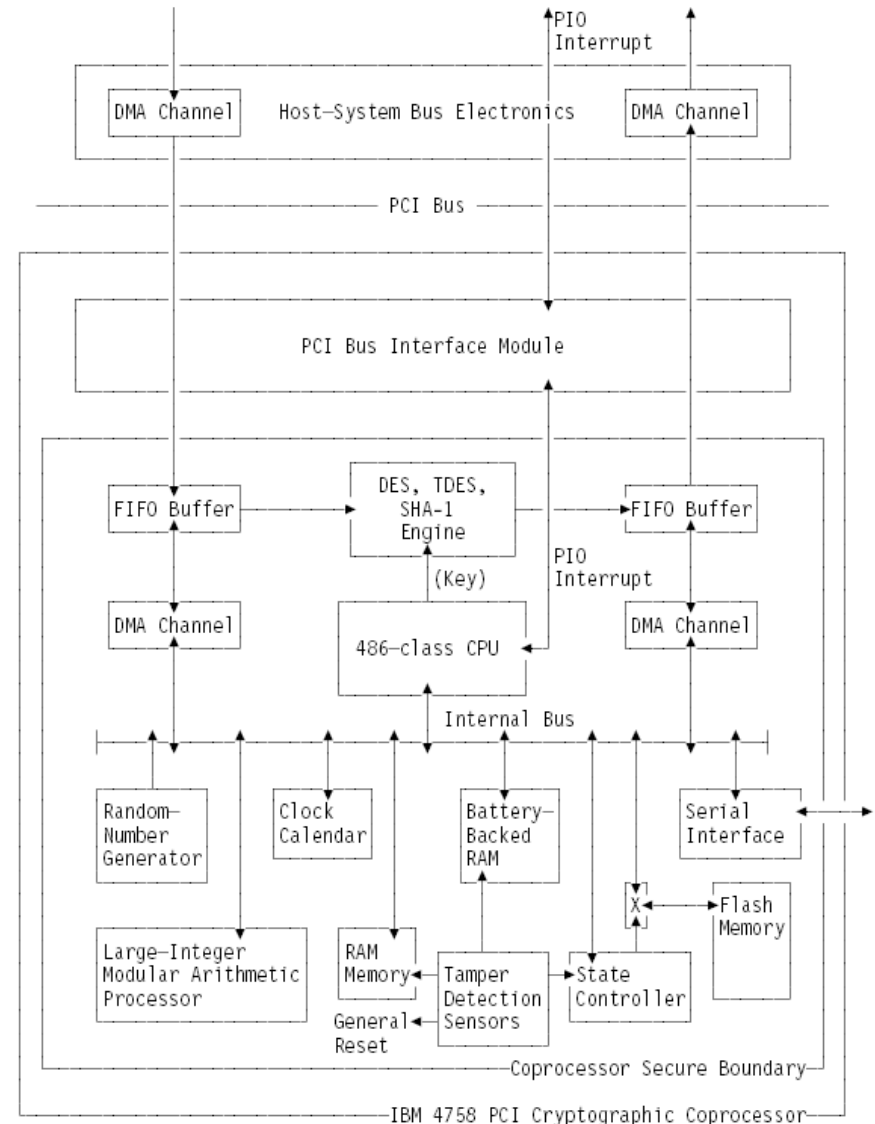
The IBM 4758 crypto coprocessor

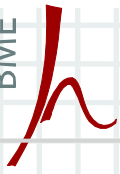
- programmable PCI board with custom hardware to support cryptography and tamper resistant packaging
- main features:
 - pipelined DES encryption engine
 - pipelined SHA-1 hash engine
 - 1024-bit and 2048-bit modular math hardware to support RSA and DSA
 - hardware noise source to seed random number generation
 - pseudo-random number generator
 - support for RSA key pair generation, encryption, and decryption
 - support for key management
 - DES based, RSA based, key diversification, PIN generation
 - secure clock-calendar
 - support for PKCS#11 and IBM Common Cryptographic Architecture (CCA)
 - battery backed RAM (BBRAM) to store secrets persistently
 - steel house with tamper detecting sensors and circuitry to erase the sensitive memory



IBM 4758 hardware

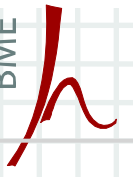
- Intel 80486 CPU (66 OR 99 MHz)
- ROM for bootstrapping code
- Flash memory to store bootstrapping code, OS, and application code
- 4 MB RAM for applications data
- 32 KB BBRAM for secrets
- hardware support for common cryptographic operations
- random number generator
- on-board clock
- tamper detection sensors (with own battery)
- state controller
 - controls access to portions of the BBRAM and Flash memory
 - functions as a hardware lock separated from the CPU
 - denies unauthorized access to secrets even if the CPU runs a malicious application





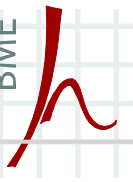
Defending against physical attacks

- the board is wrapped in a grid of conductors, which is monitored by a circuit that can detect changes in the properties of these conductors
- conductors are non-metallic and resemble the material in which they are embedded
- the grid is arranged in several layers
- entire package is enclosed in a grounded shield to reduce detectable electromagnetic emanations
- additional sensors:
 - temperature
 - humidity
 - pressure
 - ionizing radiation
 - changes in supply voltage and clock frequency
- reaction to tamper: erase BBRAM and reset the whole device
- physical security is certified at FIPS 140 level 4



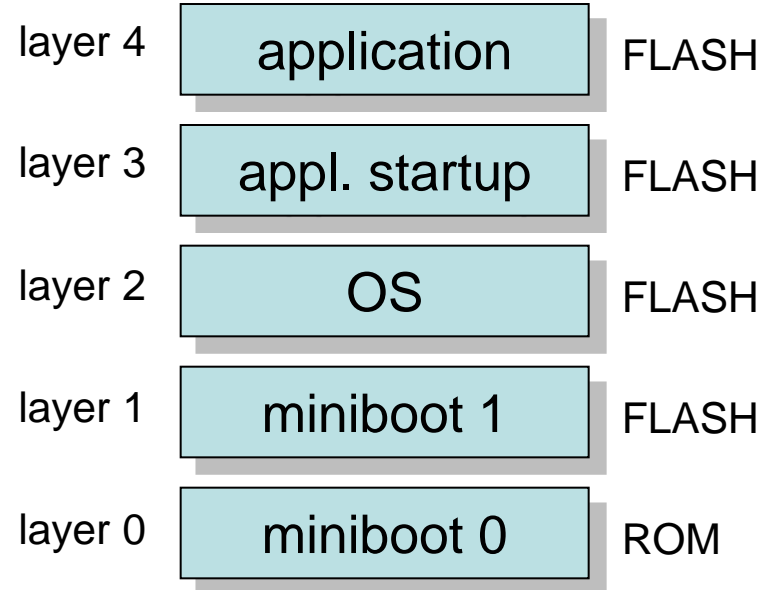
Device initialization

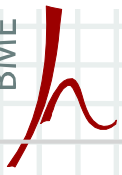
- the primary secret of the device is its RSA private key
- factory initialization
 - each device generates its own RSA key pair (using its own random number generator)
 - private key is kept in the BBRAM, public key is exported
 - external CA (manufacturer) certifies the public key by adding identifying information about the device and its software configuration and signing the certificate
 - device certificate is loaded back in the device
- regeneration of key pairs
 - the device generates a new key pair
 - signs a transition certificate with the old private key for the new public key
 - atomically deletes the old private key and activates the new one



Code layers

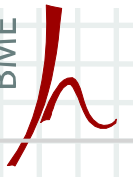
- software is organized into layers
- device is shipped with miniboot 0 and 1
- OS and applications are loaded into the Flash memory by miniboot 1
- each layer has its own page in the BBRAM, where it can store its own secrets
 - device private key is stored in page 1
- the state controller ensures that code running at layer N cannot access pages that belong to lower layers





Access to the BBRAM

| | <i>Ratchet 0</i> <i>(Miniboot 0)</i> | <i>Ratchet 1</i> <i>(Miniboot 1)</i> | <i>Ratchet 2</i> <i>(OS start-up)</i> | <i>Ratchet 3</i> <i>(Application start-up)</i> | <i>Ratchet 4</i> <i>(Application)</i> |
|-------------------------|---|---|--|---|--|
| <i>Protected Page 0</i> | <p>NO ACCESS</p> <p>READ, WRITE ALLOWED</p> | | | | |
| <i>Protected Page 1</i> | | | | | |
| <i>Protected Page 2</i> | | | | | |
| <i>Protected Page 3</i> | | | | | |

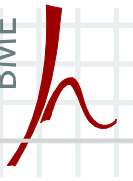


Secure bootstrapping

bootstrapping sequence:

miniboot 0 → miniboot 1 → OS startup → appl. startup → application

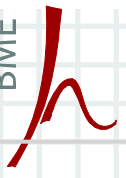
- after HW reset, the CPU starts miniboot 0 from ROM
- miniboot 0
 - runs self-test and evaluates the hardware needed to continue execution
 - checks the integrity of miniboot 1
 - advances the state controller from 0 to 1
 - and starts miniboot 1
- miniboot 1
 - runs self-test and evaluates the rest of the hardware
 - checks the integrity of OS and applications
 - advances the state controller from 1 to 2
 - and starts the OS
- OS
 - starts up
 - if needs to hide data from applications, then advances the state controller from 2 to 3 before invoking layer 3 code



Code integrity

- problem: how to ensure that a malicious application cannot change the code of the OS and the miniboots?
 - the application can remove the integrity checks and the instruction to advance the state controller
 - next time the device is booted, miniboot 1 will not check the integrity of itself and upper layer codes, and will not advance the state controller
 - then, the application can read secrets of lower layers

- solution: the state controller prevents writing access to the Flash by the OS and the applications
 - all write accesses are denied when the state is greater than 1
 - only miniboot 1 can update software in the Flash !

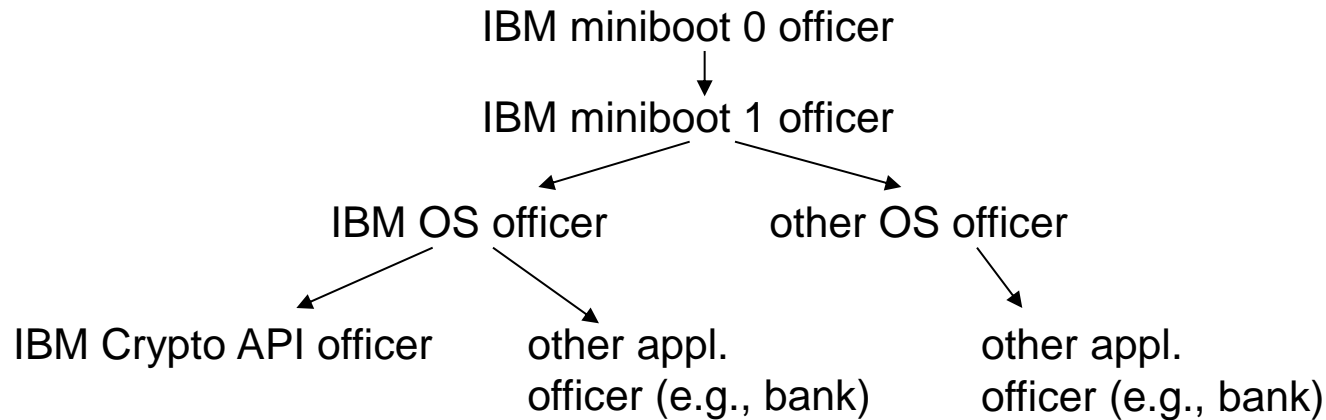


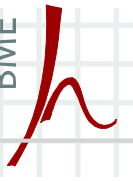
Access to the Flash memory

| | <i>Ratchet 0</i> <i>(Miniboot 0)</i> | <i>Ratchet 1</i> <i>(Miniboot 1)</i> | <i>Ratchet 2</i> <i>(OS start-up)</i> | <i>Ratchet 3</i> <i>(Application start-up)</i> | <i>Ratchet 4</i> <i>(Application)</i> |
|---|---|---|--|---|--|
| <i>Protected Segment 1</i> <i>(Miniboot 1)</i> | READ, WRITE ALLOWED | | READ ALLOWED, WRITE PROHIBITED | | |
| <i>Protected Segment 2</i> <i>(Operating System)</i> | | | | | |
| <i>Protected Segment 3</i> <i>(Application)</i> | | | | | |

Code authorities

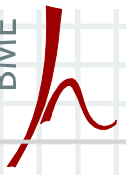
- loading of new software into a given layer is authorized by code authorities
- code authorities are organized into a tree
- each authority has a key pair
- parent certifies public key of its children
- miniboot 1 knows the public key of the miniboot 1 authority





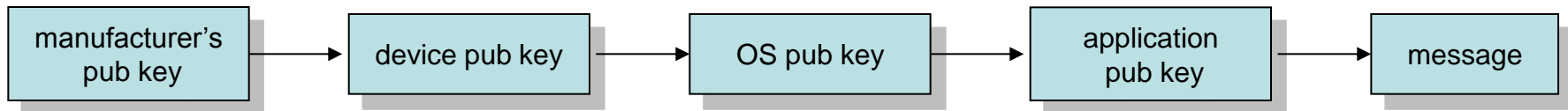
Code loading

- code to be loaded is signed by the appropriate authority
- necessary certificate chain is also attached to the signed code
- miniboot 1 can verify the chain and the signature on the code
- if everything is correct, it loads the new code into the Flash



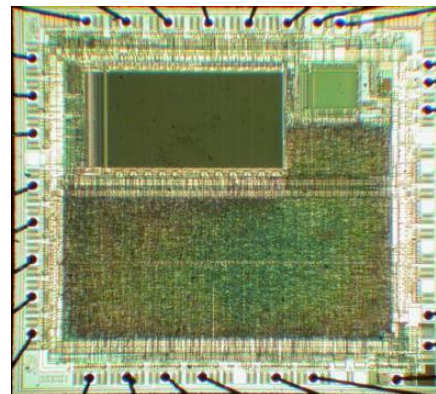
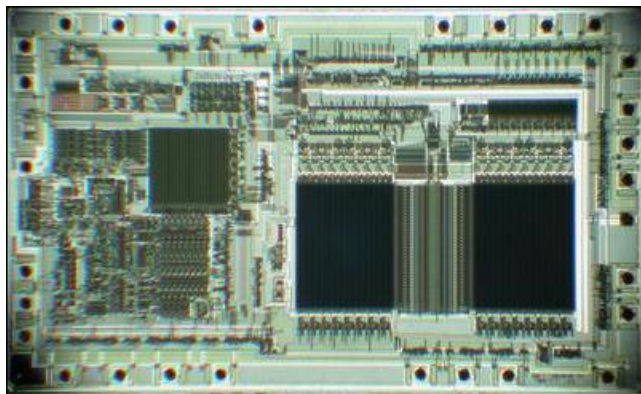
Authenticating the execution

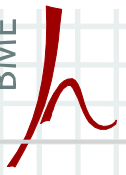
- problem:
 - how to distinguish between a message from an untampered device and a message from a clever adversary
- naïve approach:
 - device should use the device private key to sign messages
- problem with the naïve approach:
 - applications have no access to the device private key (stored in page 1 in the BBRAM)
- solution:
 - each layer N has a key pair, and its public key is signed by the private key of layer N-1
 - public key of layer 2 is signed by the device private key
 - application signs its messages with its own private key
 - signature can be verified with a chain of certificates starting from the device certificate



Protection of contemporary smart cards

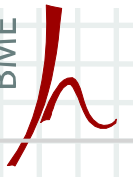
- internal voltage sensors to protect against under- and over-voltages used in power glitch attacks
- clock frequency sensors to prevent attackers slowing down the clock frequency for static analysis and also from raising it for clock-glitch attacks
- top-layer metal sensor meshes
- internal bus hardware encryption to make data analysis more difficult
- light sensors to prevent an opened chip from functioning
- password protected software access to internal memory
- standard building-block structures → randomized ASIC-like logic design (glue logic)





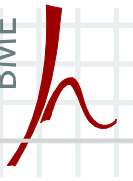
Comparison of high-end and mid-range devices

- level of security
 - HE: very high
 - MR: reasonably high
- price
 - HE: very high
 - MR: acceptable
- performance
 - HE: high
 - MR: acceptable (crypto support exists)
- flexibility and trust model
 - HE: flexible, multiple levels of trust
 - MR: simpler, but still flexible (can support multiple applications)
- robustness
 - HE: low (sensitivity to parameters, e.g., temperature)
 - MR: high
- on-board battery (and trusted clock)
 - HE: YES
 - MR: usually NO



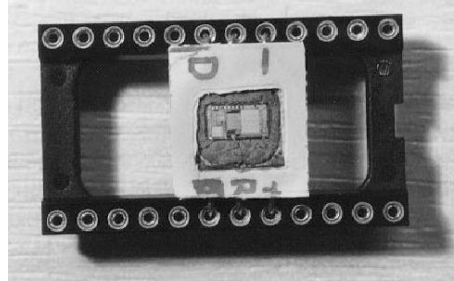
Taxonomy of attacks

- invasive attacks
 - direct electrical access to the internal components of the device
 - often permanently destroys the device
- semi-invasive attacks
 - access to the device, but without damaging the passivation layer of the chip or making electrical contact other than with the authorised interface
- non-invasive attacks
 - local: observation or manipulation of the device's operation (timing, power consumption, clock frequency)
 - remote: observation or manipulation of the device's normal input and output (API attacks)
- all of the above attacks can be passive or active

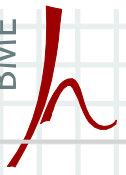


Invasive attacks

- first step: removing the chip from the plastic cover
 - nitric acid dissolves epoxy without damaging silicon

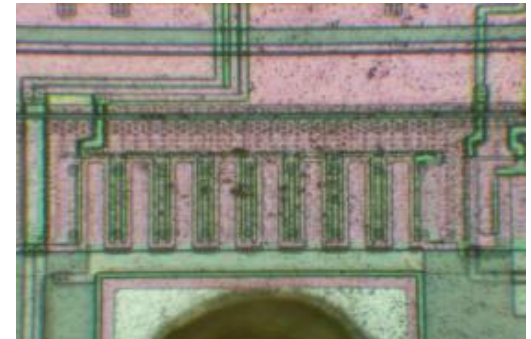
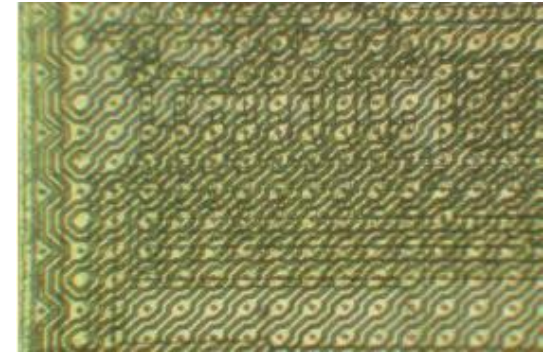


- second step: probing chip internals directly
 - second-hand semiconductor test equipment such as manual probing stations are available (e.g., renting)
 - a typical probing station consists of
 - a microscope with an objective working distance of several millimeters mounted on a low-vibration platform
 - micromanipulators to place probes (microprobing needles) on to the device
 - a laser, with which small holes can be drilled in the chip's passivation layer (holes allow electrical contact by the probes, and indeed stabilise them in position)
 - with such equipment one can probe the device's internal bus system, so that both program and data can be read out



Defending against invasive attacks

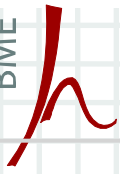
- sensor mesh implemented in the top metal layer, consisting of a serpentine pattern of sensor, ground and power lines
 - if the sensor line is broken, or shorted to ground or power, the device self-destructs
- making it more difficult to visually analyze the chip surface
 - earlier the structure of a microcontroller could be easily observed and reverse engineered under a microscope
 - although buried under the top metal layer, the second metal layer and polysilicon layer can still be seen, because each subsequent layer in the fabrication process follows the shape of the previous layer
 - today, each layer but the last one is planarised using chemical-mechanical polishing before applying the next layer
 - the only way to reveal the structure of the deeper layers is by removing the top metal layers either mechanically or chemically



PIC16F877



PIC16F877A



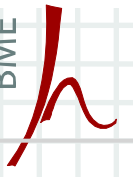
Semi-invasive attacks

- attacks that involve access to the chip surface, but which do not require penetration of the passivation layer or direct electrical contact to the chip internals
- early example: UV light to reset the protection bit on microcontrollers (memory contents could be read out)
- recently: optical probing techniques to inject faults into digital circuits
 - illuminating a target transistor causes it to conduct, thereby inducing a transient fault
 - possible to set or reset any individual bit of SRAM in a microcontroller
 - can be carried out with simple, low-cost equipment (e.g., with photographer's flash gun)
- better results can be obtained using laser probing equipment
 - in addition to setting RAM contents to desired values, it can be adapted to read out other memory technologies, such as Flash and EEPROM, and to interfere with control logic directly



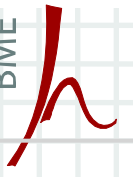
Defending semi-invasive attacks

- detection of active attacks and performing some suitable alarm function, such as erasing key material
- opaque top-layer metal grids and shields
 - the attacker may now have to go through the rear of the chip, which will typically involve ion-etching equipment to thin the device and an infra-red laser to penetrate the silicon substrate



Local non-invasive attacks

- side-channel attacks
 - careful observation of the interaction of the card with its environment during critical operations may reveal some amount of information about the sensitive data stored in the card
 - examples: timing attacks and power analysis
- unusual operating conditions may have undocumented effects
 - unusual temperatures or voltages can affect EEPROM write operations
 - power and clock glitches may affect the execution of individual instructions
 - e.g., doubling the clock frequency for a few microseconds would cause some, but not all, instructions to fail.
 - it could be possible to modify the device's control flow – for example, by stepping over the branch instruction following a failed password check

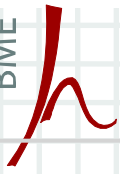


Example: Exploiting effects of a clock glitch

- a typical subroutine (writes the content of a limited memory range to the serial port):

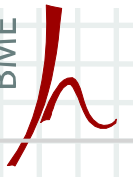
```
1 a = answer_address
2 b = answer_length
3 if (b == 0) goto 8
4 transmit(*a)
5 a = a + 1
6 b = b - 1
7 goto 3
8 ...
```

- if we can find a glitch that transforms the loop variable decrement in line 6 into something else, then the chip will dump the content of the whole memory



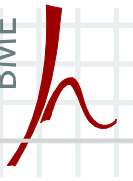
Example: Differential Power Analysis (DPA)

- measure the power consumption of a chip while it does a number of cryptographic computations (typically several hundred) with different data → power traces
- guess the value of some bit of the key
- sort the power traces into two piles, depending on whether the target bits combined with the observed input or output data would have activated some circuit (e.g., the carry function) in the processor or not
- check the guess by verifying if the two piles are statistically different



Defending local non-invasive attacks

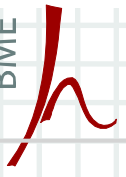
- randomization of code
 - put NOP instructions randomly in the program
- randomization of crypto algorithm implementations
 - makes it more difficult to exploit side channel information that stem from the mathematical structure of the algorithm
- randomization of all protocol messages
 - makes it difficult to collect plaintext-ciphertext pairs
- design time validation
 - tools to simulate power analysis and other emsec attacks early in the design stage, so that changes can be made before the chip is fabricated



API attacks

- a tamper resistant module is a custom computer in tamper resistant packaging
 - hardware support for cryptographic functions
 - tamper detection and reaction circuitry
 - internal battery and clock
 - ...
 - API

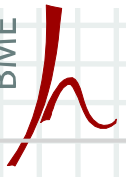
- the API of a tamper resistant module is a software layer through which the module's functions are exposed to the external world



An example API

- `key_export`
 - inputs
 - key token: $E_{MK}(K)$
 - key encryption key token: $E_{MK}(KEK)$
 - outputs
 - exported key token: $E_{KEK}(K)$

- `key_import`
 - inputs
 - external key token: $E_{KEK}(K)$
 - key encryption key token: $E_{MK}(KEK)$
 - outputs
 - imported key token: $E_{MK}(K)$

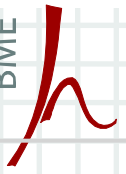


An example API

- key_part_import
 - inputs
 - key part: K'
 - key token: $E_{MK}(K)$
 - outputs
 - updated key token: $E_{MK}(K+K')$

- encrypt
 - inputs
 - key token: $E_{MK}(K)$
 - data: X
 - outputs
 - encrypted data: $E_K(X)$

- ...



API attacks

- exploit design weaknesses of the API for extracting secrets from the module or increasing the efficiency of cryptanalytical attacks
- simple examples:
 - typing attack:
 $\text{key_export}(E_{MK}(K), E_{MK}(KEK)) \rightarrow \text{returns } E_{KEK}(K)$
 $\text{decrypt}(E_{MK}(KEK), E_{KEK}(K)) \rightarrow \text{returns } K$
 - creating related keys:
 $\text{key_part_import}(K', E_{MK}(K)) \rightarrow \text{creates } K+K'$
 $\text{key_part_import}(K'+\Delta, E_{MK}(K)) \rightarrow \text{creates } K+K'+\Delta$
 - key conjuring:
 $\text{key_import}(R, R') \rightarrow \text{creates an unknown key } D_{DMK(R')}(R)$

Attacking the API of the IBM 4758

- preliminaries
 - keys are stored externally in *key tokens*
 - key tokens are encrypted with a master key or a key wrapping key (key encryption key) *modulated with the type* of the key in the token, where types are encoded in *control vectors*
 - example:
 - let K be an exportable symmetric data encryption key
 - let KEK be a key encryption key
 - K is exported under the protection of KEK in a key token $E_{KEK+CV_DEK}(K)$: “DEK” where CV_DEK is the bit string representing the control vector for data encryption keys, and “DEK” encodes the type “Data Encryption Key”
 - type indicators (e.g., “DEK”) are not protected, and hence, can be modified



Attacking the API of the IBM 4758

- use `key_part_import` to create two unknown but related key encryption keys `UKEK` and `UKEK'`:

`key_part_import (K', EMK+CV_KEK(K):"KEK")`

→ computes $UKEK = K + K'$

→ outputs $E_{MK+CV_KEK}(UKEK):"KEK"$

`key_part_import (K' + CV_KEK + CV_DEK, EMK+CV_KEK(K):"KEK")`

→ computes $UKEK' = K + K' + CV_KEK + CV_DEK$

→ outputs $E_{MK+CV_KEK}(UKEK'): "KEK"$

$UKEK' = UKEK + CV_KEK + CV_DEK$

Attacking the API of the IBM 4758

- use `key_import` to create two copies of an unknown random key URK with different types:

`key_import (R:"KEK", $E_{MK+CV_KEK}(UKEK)$:"KEK")`

→ computes $URK = D_{UKEK+CV_KEK}(R)$

→ outputs $E_{MK+CV_KEK}(URK)$:"KEK"

`key_import (R:"DEK", $E_{MK+CV_KEK}(UKEK')$:"KEK")`

→ computes $URK' = D_{UKEK'+CV_DEK}(R)$

→ outputs $E_{MK+CV_DEK}(URK')$:"DEK"

$$URK' = D_{UKEK'+CV_DEK}(R)$$

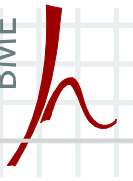
$$= D_{UKEK+CV_KEK+CV_DEK+CV_DEK}(R)$$

$$= D_{UKEK+CV_KEK}(R)$$

$$= URK$$

Attacking the API of the IBM 4758

- use key part import to create a key encryption key URK” = URK+CV_DEK:
 - key_part_import(CV_DEK, $E_{MK+CV_KEK}(URK):“KEK”$)
 - computes URK” = URK + CV_DEK
 - outputs $E_{MK+CV_KEK}(URK’):“KEK”$
- export URK:“DEK” under URK’:“KEK”:
 - key_export ($E_{MK+CV_DEK}(URK):“DEK”$, $E_{MK+CV_KEK}(URK’):“KEK”$)
 - outputs $E_{URK’+CV_DEK}(URK) = E_{URK+CV_DEK+CV_DEK}(URK) = E_{URK}(URK):“DEK”$
- decrypt $E_{URK}(URK)$ with URK:“DEK”:
 - decrypt ($E_{MK+CV_DEK}(URK):“DEK”$, $E_{URK}(URK)$) → returns URK
- export any target key K_{target} under URK:“KEK”



Financial APIs

- they support
 - PIN generation, verification, encryption
 - encrypted PIN translation between zone keys
 - ...

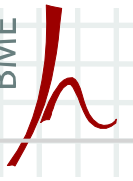
- examples:
 - PIN generation
 - encrypt the PAN with a PIN generation master key
 - decimalize result
 - select desired number of digits

 - PIN encryption
 - PIN is formatted into an 8 byte clear PIN block
 - encrypted with a PIN encryption key using DES or 3DES

 - encrypted PIN translation (and reformatting)
 - encrypted PIN is decrypted with the supplied input key
 - clear PIN is re-encrypted with the supplied output key
 - clear PIN is reformatted before re-encryption if needed

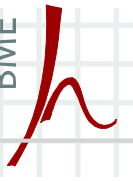
PIN generation details

- input:
 - PAN (personal account number)
 - Decimalization Table
 - PIN generation key id
- steps:
 - encrypt PAN with PIN generation key
 - decimalize result with Decimalization Table
 - select desired amount of digits → PIN
- output:
 - PIN (to printer)
- example:
 - PAN = 87654321 87654321
 - ciphertext = a0b1c2d3e4f5a6b7
 - Decimalization Table = 0123456789012345
 - decimalized ciphertext = 0011223344550617
 - four digit PIN = 0011



Remote PIN verification

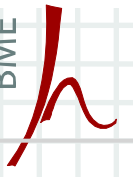
- input:
 - PAN
 - Decimalization Table
 - encrypted PIN block
 - PIN encryption key id
 - PIN generation key id
- steps:
 - decrypt encrypted PIN with PIN encryption key → PIN
 - compute PIN from PAN, PIN generation key, and Decimalization Table → PIN'
 - compare PIN to PIN'
- output:
 - accept / reject (/ PIN formatting error)



PIN offset generation

- input:
 - PAN
 - Decimalization Table
 - PIN generation key id
 - user selected PIN (UPIN)
- steps:
 - generate PIN from PAN, PIN generation key, and Dec Table → IPIN
 - digit subtraction: $UPIN - IPIN \pmod{10} \rightarrow OFFSET$
- output:
 - OFFSET (stored on the user's card)

- example:
 - PAN = 87654321 87654321
 - ciphertext = a0b1c2d3e4f5a6b7
 - Decimalization Table = 0123456789012345
 - decimalized ciphertext = 0011223344550617
 - four digit IPIN = 0011
 - user selected PIN = 1234
 - OFFSET = 1223



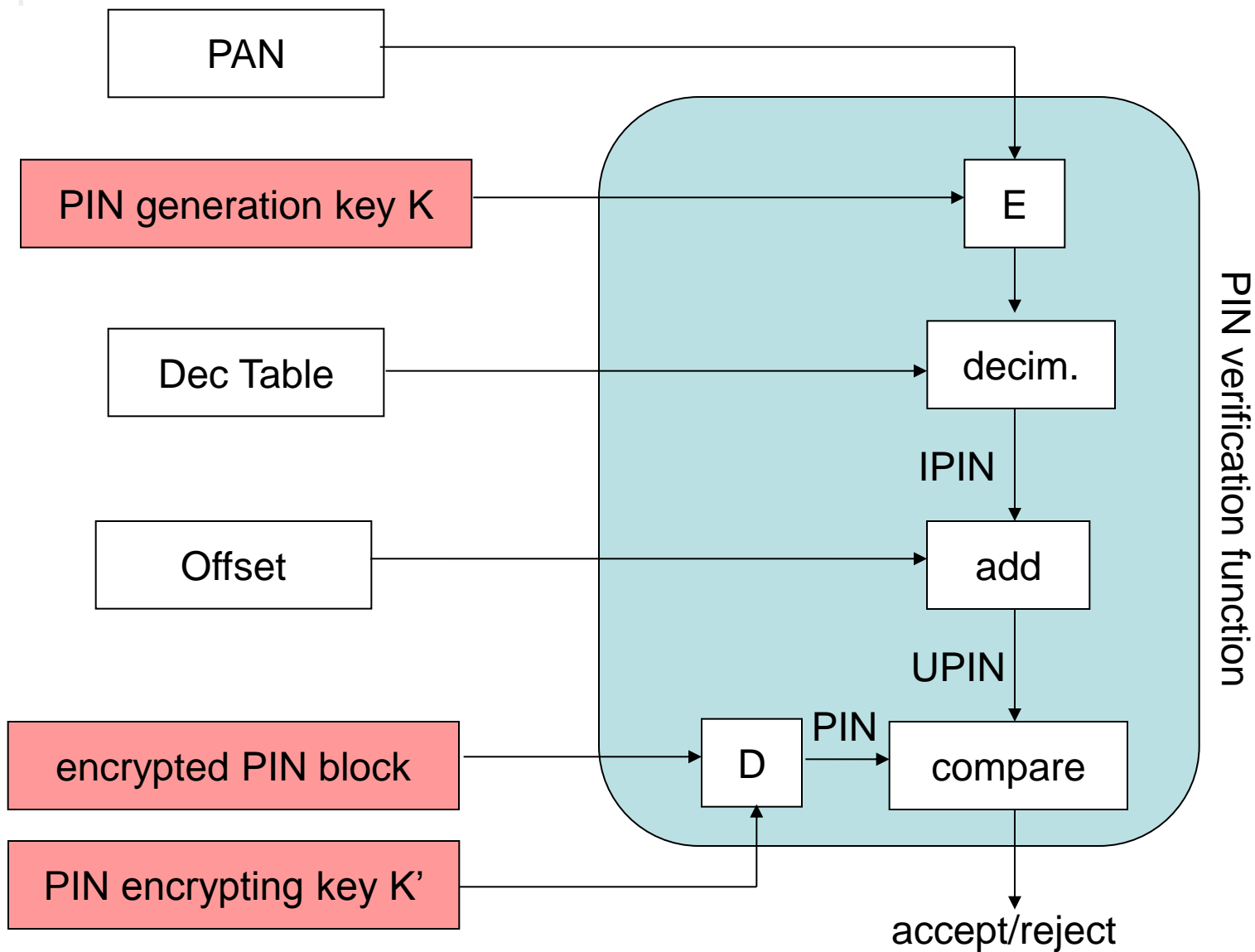
Remote PIN verification with offset

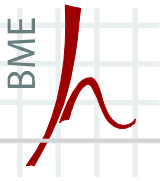
- inputs:
 - PAN
 - Dec Table
 - encrypted PIN block (EPB)
 - Offset
 - PIN encrypting key id
 - PIN generation key id

- steps:
 - decrypt EPB \rightarrow PB
 - extract PIN from PB \rightarrow PIN
 - compute IPIN from PAN, Dec Table, and PIN generation key
 - compute UPIN from IPIN and Offset \rightarrow UPIN
 - compare UPIN and PIN

- outputs:
 - accept / reject / error

Remote PIN verification with offset – illustrated



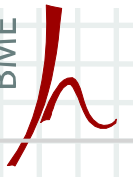


Decimalization attack

- example setup:

```
PAN =          1122334455667788
enc PAN =      E481FC5658391418
Dec Table =    0123456789012345
IPIN =         4481
UPIN =         6598
Offset =       2117
```

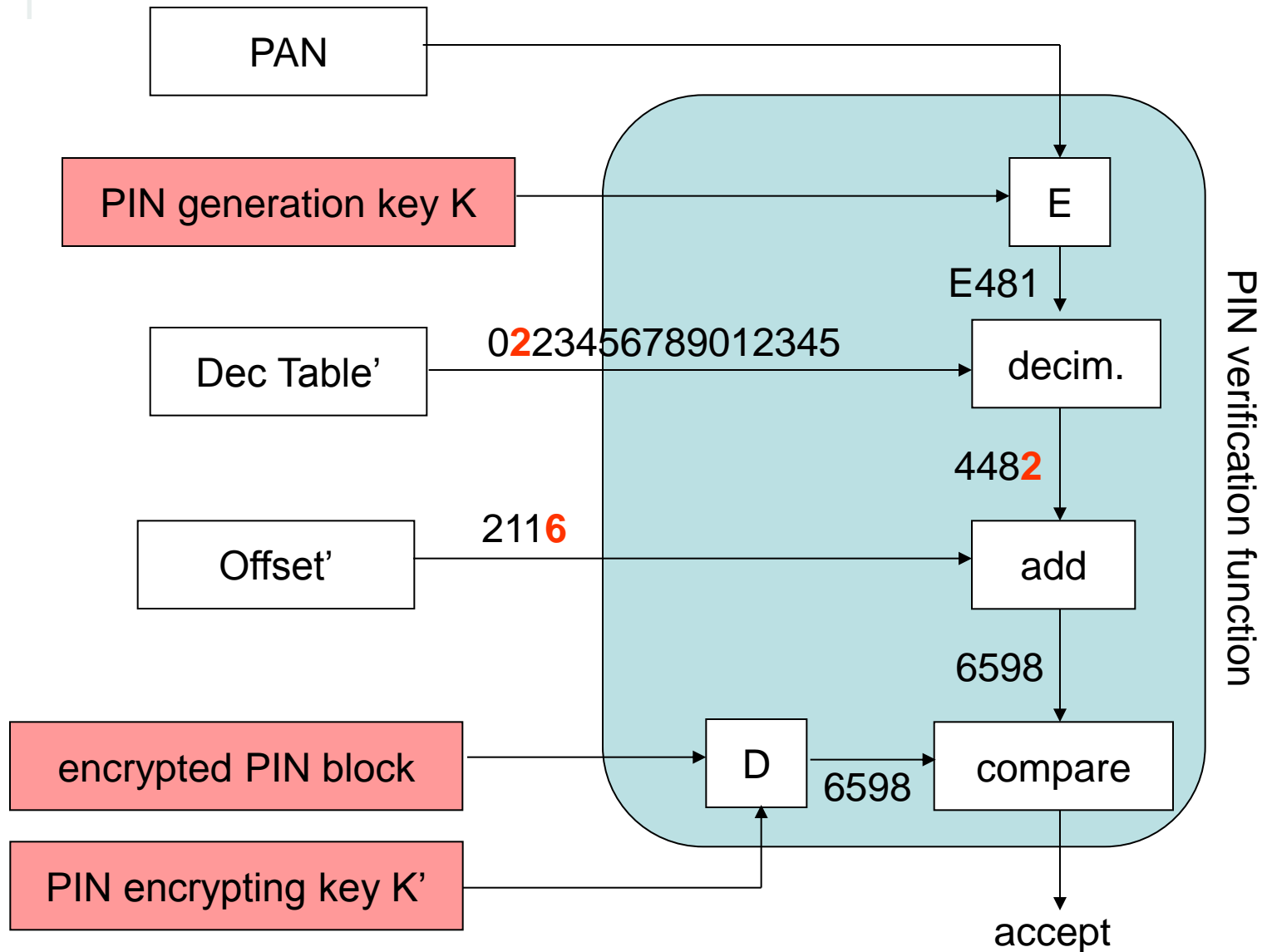
- assume the attacker can call remote PIN verification with offset and he can manipulate the input fields
- what if we change the Dec Table to **1**123456789012345 ?
 - since E481 does not contain a 0, we get the same IPIN = 4481, and Offset = 2117 will pass → the device returns “accept”



Decimalization attack

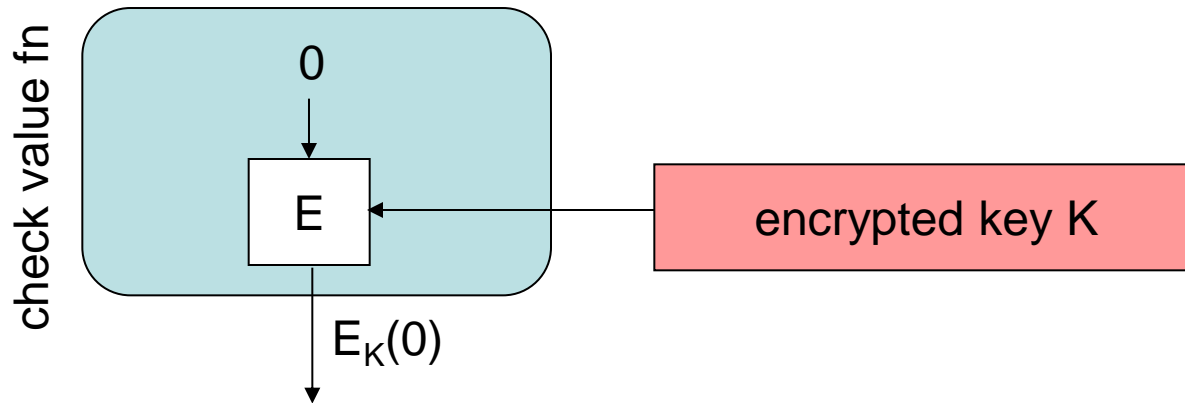
- now let's change the Dec Table to 0**2**23456789012345 !
 - the Ciphertext is decimalized into IPIN = 4482
 - Offset = 2117 does not pass ($4482 + 2117 = 6599$)
 - we know that there's a 1 in the IPIN !
 - we have to find out in which position
 - for this, we modify the Offset until it passes
 - Offset = 2116 will pass
 - we know that the last digit of IPIN is 1 !
- and so on ...
- when IPIN is obtained, we compute the UPIN as IPIN + Offset

Decimalization attack – illustrated

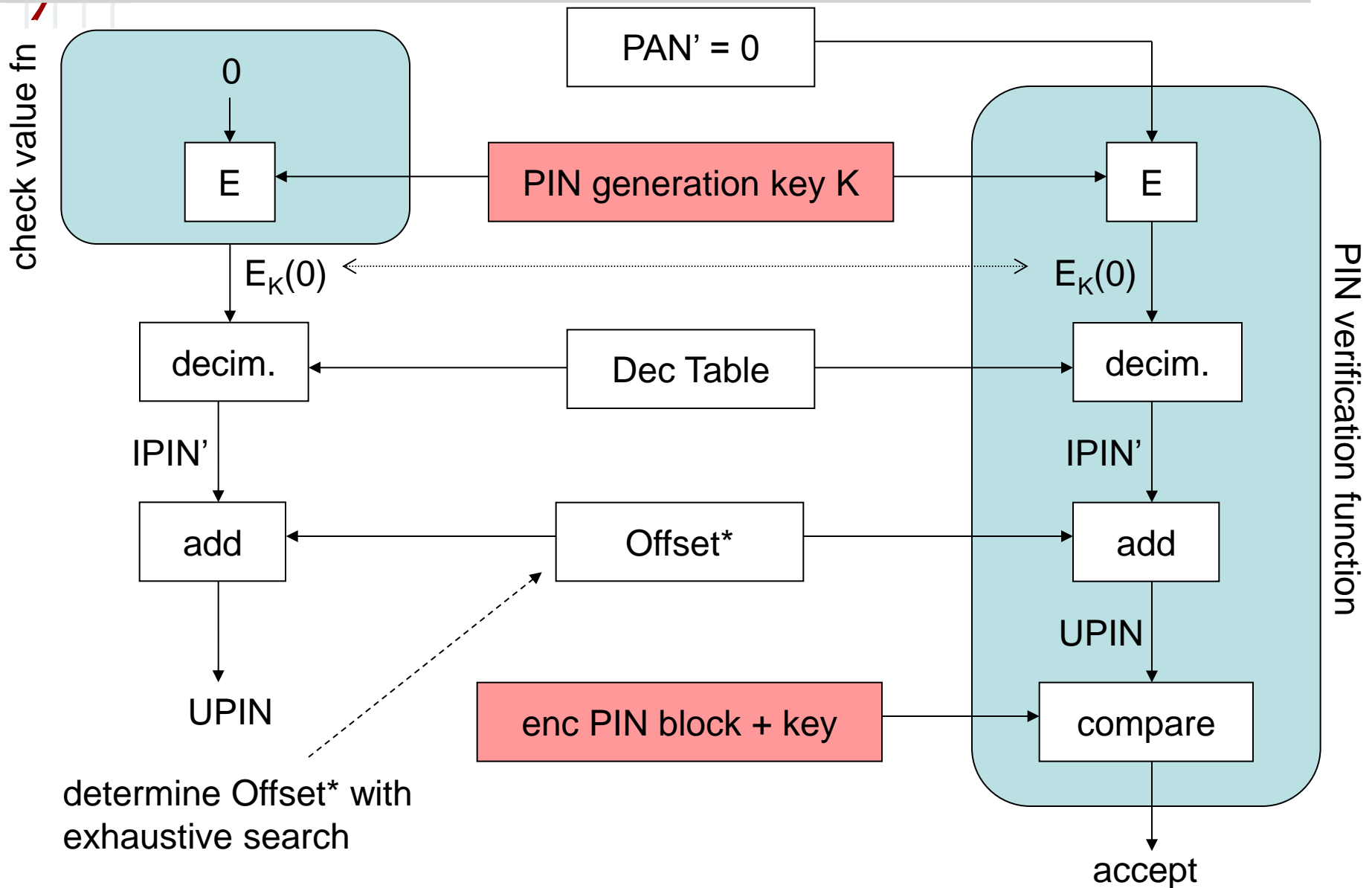


Check value function

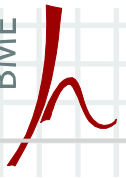
- encrypts the all 0 input with a user supplied key (token)
- used for test purposes



Check value attack



PIN verification function



Lessons learnt

- no matter how secure the device is physically if it leaks secrets due to API attacks
- most tamper resistant devices are vulnerable to some form of API attacks
- careful design and analysis of the API is indeed very important with respect to overall security



Security analysis of APIs

- API attacks can be very subtle and hard to discover by informal analysis
- the problem of API analysis seems to be very similar to that of analyzing authentication and key exchange protocols
 - the attacker interacts with the device using a well defined set of “messages”
 - the goal is to obtain some secret or bring the device in a “bad” state
- formal analysis techniques developed for key exchange protocols may be amenable to the analysis of crypto APIs