Hardware Security

Lecture 13

Protocols and Secure Devices

Alice wants to talk to Bob but she's got just a key shared with C

Messages

- (1) A->C: timeA, B
  //please, can you generate a key for A-B
- (2) C->A: E_{Kac}(timeA, Kab, B), E_{Kbc}(timeC, Kab, A)
  // OK, here's the key copy for you, and this is for Bob
- (3) A->B: E_{Kbc}(timeC, Kab, A), E_{Kab}(timeA, B)
  // hi Bob, I got a key copy from C, here it is
- (4) B->A: E_{Kab}(timeB, A)
  // thanks, I send back my time encrypted with our key

Protocols and Secure Devices

Messages

- (1) A->C: timeA, B
- (2) C->A: E_{Kac}(timeA, Kab, B), E_{Kbc}(timeC, Kab, A)
- (3) A->B: E_{Kbc}(timeC, Kab, A), E_{Kab}(timeA, B)
- (4) B->A: E_{Kab}(timeB, A)

WE ASSUME THAT

- C has all keys ***stored*** securely
- C – is trustworthy to ***create*** good keys for A-B
- C – ***deletes*** its copy of key Kab as soon as message (2) is created
- there are shared keys A-C, B-C
- …

Protocols and Secure Devices

- Condition
  - There are shared keys between A-C and B-C
- Realisation
  1. we have to specify how to: generate keys, how to transfer keys, update of keys
  2. define mechanisms for the requirements
  3. implement mechanisms, define operational environment, define best practices, …
  4. work out an audit and verify security of realisation

Documentation may take *thousands* of pages

Protocols and Secure Devices

We would love to say instead:
"if device X is used, key scheme is secure"

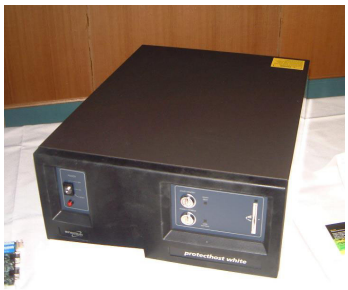
Introducing Secure Hardware

- Secure Hardware comes in many shapes and sizes, and has a wide variety of uses.
- Types of secure hardware
  - **HSM** : Hardware Security Module (aka TRSM tamper-resistant security module), the biggest, most powerful and the most secure
  - **Secure Microcontroller** : Tamper-resistant microcontroller chip used as a component in secure devices such as Point-of-Sale terminals
  - **Smartcard** : Low-end, highly portable chip-based secure hardware, typically carried in wallets of end user.
  - **Other** : Variety of other sorts, based around smartcard technology within a new form-factor e.g. USB token, visual display only (RSA SecurID),

Hardware Security Modules

Cv source: Mike Bond

Hardware Security Modules (II)



Daniel Cvrček, 2005 source: Mike Bond

Hardware Security Modules (III)

Smaller form factors…

source: Mike Bond

Tamper-Resistance

- Example: The IBM 4758
  - Potted in urethane resin
  - Protective tamper-sensing membrane, chemically identical to potting compound
  - Detectors for temperature & X-Rays
  - “Tempest” shielding for RF emission
  - Low pass filters on power supply rails

Moving on swiftly: talking too much about countermeasures precludes discussion of attacks they resist…

source: Mike Bond

Who Needs Secure Hardware ?

- Those who need to enforce access policies to sensitive information
  Example: Granting signing permission at a Certification Authority
- Those who need to protect mission critical sensitive data
  Example: Protecting PIN generation keys at banks
- Those who need to protect data in hostile environments
  Example: Protecting Token Vending Machines (Electricity, National Lottery etc…)
- Those with high crypto throughput requirements
  Example: SSL acceleration for webservers

source: Mike Bond

Applications

- ***Banking Security*** – Generating, Transporting and Verifying PINs in bank ATM and POS networks
- ***Electronics Payments Systems*** – Authentication and storage tokens for retail and online payments systems (e.g. "Chip and PIN")
- ***Prepayment Token Systems*** – For regulating the issue of electronic credit tokens, for prepayment electricity, mobile phones, electronic stamps for postage
- ***Trusted Computing*** – For Digital Rights Management (DRM), Secure Boot, Attestation,
- ***Secure Storage*** – Keeping the keys safe in SSL enabled webservers and at Certification Authorities

source: Mike Bond

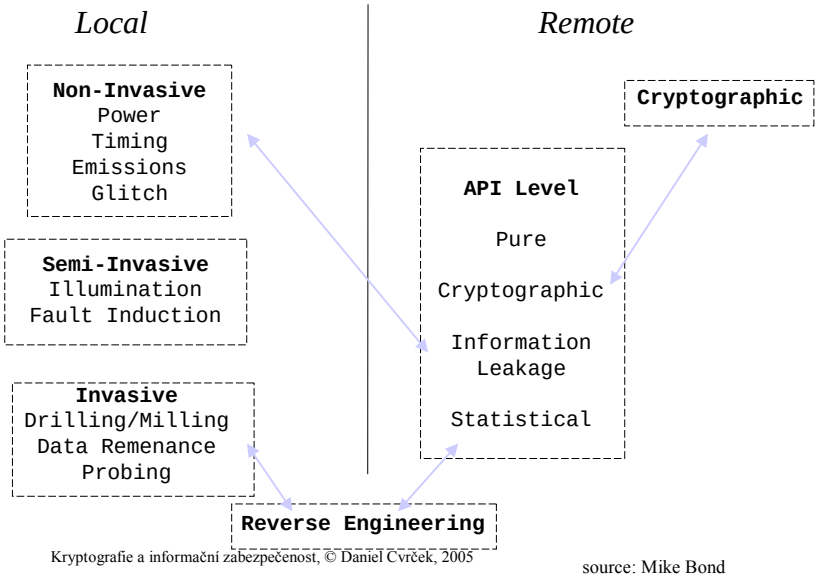
Applications (II)

- ***Pay TV*** – Authorisation and decryption tokens for broadcast video streams in Satellite and Cable TV systems. Controls your access to various channels.
- ***Authorisation Tokens*** – One time password generators (e.g. RSA SecurID series), and other authentication tokens for logon to other systems or authorisation of specific transactions
- ***High Value Trading Systems*** – For regulating electronic stocks and shares, protecting algorithms, or ensuring fair policy. E.g. electronic control of "Bills of Lading" for large ships.
- ***Compulsion Resistance*** – For secure storage of keys and processing in censorship-resistant and anonymity systems (e.g. mix networks)
- ***Military Security*** – Mitigating losses in event of battlefield capture of equipment, access control and authorisation for nuclear ordinance.

source: Mike Bond

Methods of Hacking Secure Hardware

Taxonomy of Attack Methods



source: Mike Bond

Local Non-Invasive Attacks

- These attacks require physical proximity to a device, and may require logical access to send commands. They exploit the official communications channels with the device (e.g. smartcard serial connection), and unintended side-channels in order to extract secrets (usually key material)
- Non-invasive attacks are mainly passive: they observe only legal communication with the device, and record information leaked
- We'll focus on this area in particular later…

source: Mike Bond

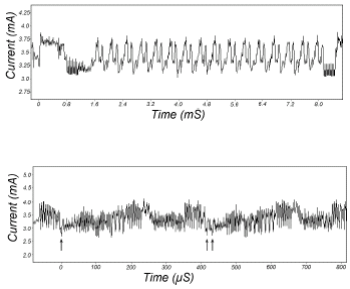
L.N-I : Timing Attacks

- Timing Attacks work by measuring the time taken to perform particular sensitive operations on the secure device (operation is usually a cryptographic one, with secret key)
- Known about since early UNIX days, when partial matches with passwords could be detected by measuring how long the logon program ran before rejecting the password. Each letter of the password could be searched independently
- Nowadays: timing attacks on loops for performing modular exponentiation in crypto algs, much much more (and still the old attacks refuse to die: weak key matching attacks in DES security modules)

source: Mike Bond

L.N-I : Power Analysis

- Power Analysis attacks exploit the changes in power consumption of microprocessors, dependent on the particular instructions being executed, or data being manipulated



Data dependent power consumption leaks data; differential analysis of traces between runs where some difference has been introduced is even more powerful. Mathematical modelling of signal and of noise further improves results.

source: Mike Bond

L.N-I : Emissions Attacks

- Emissions attacks exploit unwanted electromagnetic signals from processors executing their instructions. Carefully designed antenna can exploit all sorts of effects.
- Small coils mere millimetres in diameter positioned above surface of chips can receive emissions local to specific sub-components of the processor
- Carefully crafted antenna can pick up emissions from Hardware Security Modules high performance crypto chips at distances of tens of metres (allegedly, see IBM research). Inspired by "Tempest" attacks reconstructing images displayed on monitors from hundreds of metres away

source: Mike Bond

L.N-I : Fault Induction (Glitch) Attacks

- Fault Induction Attacks work by interfering with a microprocessor's execution flow.
- Temporary glitches in a clock stream, or pulses in the power supply to a smartcard can, for instance, cause repeated execution of an instruction, failure to branch, or failure to update a register with a particular result
- Effects:
  - a loop that reads out a buffer of data can be made to continue indefinitely, reading out the whole memory
  - Cryptographic algorithm security is very sensitive to security: they can be damaged, modified to be insecure, thus leaking the key material through the encrypted data

source: Mike Bond

Local Semi-Invasive Attacks

- Semi-Invasive Attacks may require partial depackaging of an integrated circuit, but achieve their results without full invasive tampering – avoiding expensive microprobing and anti-tamper mechanisms triggered by full depackaging.
- Example: Ultraviolet light shone selectively at secure microcontroller memory could reset the 'protection bit', allowing a full memory dump
- Example: Optical fault induction can be done on microcontrollers, to flip bits in SRAM selectively by firing a camera flash gun focussed through a microscope

source: Mike Bond

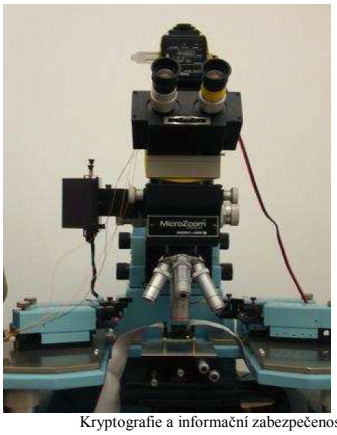
Local Invasive Attacks

- Fully invasive attacks aim to depackage an integrated circuit completely, and land microprobes to observe data on bus lines, and extract secrets.
- Active invasive attacks may use machines such as Focussed Ion Beam (FIB) workstations to selectively break or connect tracks to change the operation of a microprocessor
- Invasive attacks may use supplementary technology to resist anti-tamper mechanisms, for instance cooling of circuitry or exposure to high radiation levels to create data remanence effects

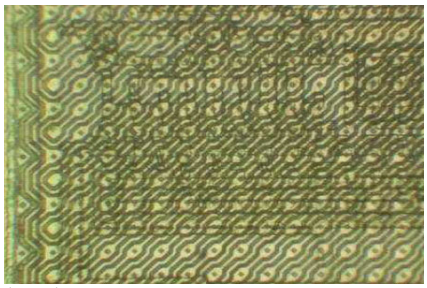
source: Mike Bond

L.I : Micro Probing

- Low cost probing workstation



Sensor mesh on ST16 smartcard, designed to resist probing



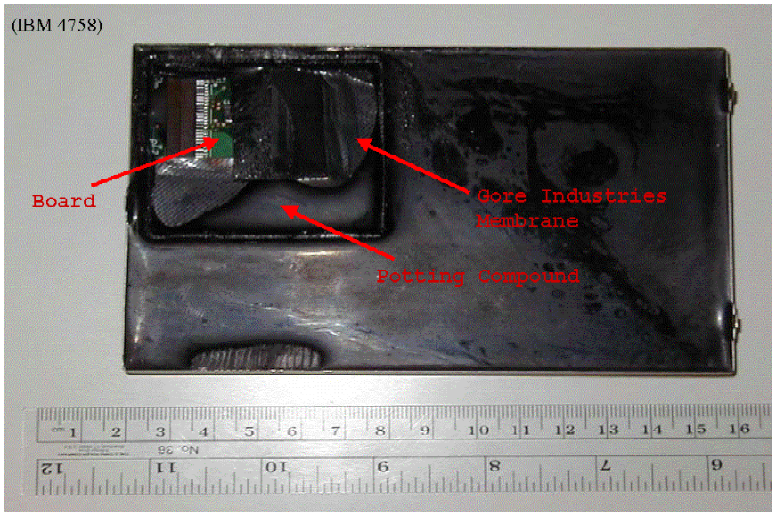
source: Mike Bond

L.I : Data Remanence Attacks

- Severe cooling of memory can dramatically lower response times to operations, and in particular the decay of data in RAM during power off state. Cooling to below 0 degrees centigrate makes data persist from 30 seconds up to four minutes or more in some devices
- Careful measurement of physical characteristics of supposedly erased chips, such as transistor switching threshold can also yield sensitive data

source: Mike Bond

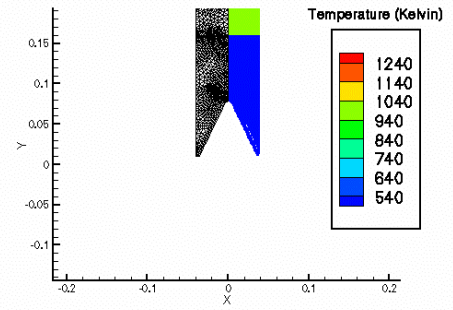
L.I : Drilling/Milling/Dissolving



source: Mike Bond

L.I : Detonating

- To tackle the very most secure devices, use of explosive charges can help. Next to no *academic* research in this area, so details sketchy.
- "Shaped Charges" explode and create a super heated directional jet of plasma, which can vaporise tamper-reactive circuitry extremely rapidly, before it has time to complete an erase operation. The RAM can then be powered back up and the data retrieved before it decays away.
- Forensic analysis of computer equipment damaged by explosions and gunfire can successfully yield secret data.



Remote Attacks: Security APIs

- Security APIs are the interfaces to secure hardware – they present a set of commands (aka transactions) that allow the user to manipulate key material and sensitive data, but enforce a **policy** on usage
- An API attack works by using a sequence of legal commands, but with unusual inputs, to trick the device into revealing secrets in a way not intended by the designer
- Security APIs are the big brother of security protocols, and many of the attacks are similar to protocol attacks.
- We'll focus on this in particular later…

Focus: Non-Invasive Local Attack

Secure devices – smart cards

A Bit of Wardriving

- what can happen
  - you can forget your card in a cash machine
  - one can pickpocket your card
  - one can also return the card to your pocket
  - one can communicate with your card without your knowledge – contactless cards
  - one can retransmit communication with your card somewhere else – to the door of your office?
  - one can get off your PIN when you're entering it at POS – basically very insecure place
  - one can run a protocol with your card and measure power supply
  - one can use warm nitric acid to get to the chip of your card and physically affect the card functionality
  - one can destroy your card without touching it
  - …

Wardriving scenario I

- hacker buys a reader for contactless cards
- he connects the reader to a laptop and puts both into a briefcase
- while walking around a building where contactless cards are used, the reader is trying to establish a connection each time a card is in its range
- it runs a protocol and
  - either finds the key necessary – when the cards are simple
  - gathers basic information for implementing own card

Wardriving scenario II

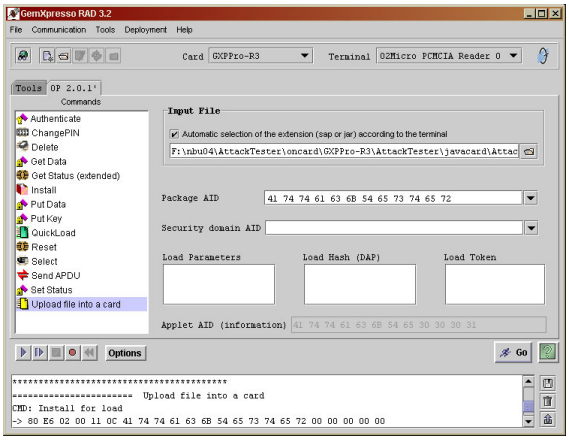
- a hacker creates a device for "measuring" electrical properties (e.g. power supply) of smart-cards
- she picks a victim, pickpockets a card
- connects it to her device and automatic script is started – we are talking about small number of minutes needed
- returns the card back to the victim's pocket
- she analyses data back in a lab
- programs an empty card with the data needed
- new card can be used e.g. at cash-machines

Wardriving scenario III

- Some of you are already using “new technology” Chip & PIN cards – also known as “EMV cards”
- shoulder-surfing – trying to get off your PIN as you’re entering it at POS
- quite easy as POSs are not designed for this sort of activity
- pick pocketing the card
- withdrawing all the money one can get
- Experiment about security of signature v Chip & PIN cards carried out by Masaryk University demonstrated 30-80 % of PINs entered can be obtained by an attacker
  - details soon on our web www.buslab.cz
- more research targeting EMV cards is underway …

We are assuming quite a lot! Or …?

Using smart-cards - JavaCards GemXpresso RADIII



There is also a JCOP development tool from IBM – 57 CHF

JavaCard programming – it's easy

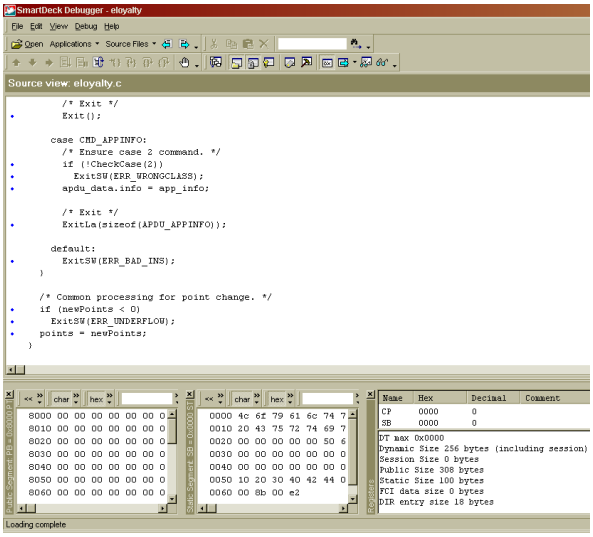
-> **84 12 00 00 04 04 03 02 01** → PIN to be verified

```
public void process(APDU apdu) throws ISOException {
    byte[] apduBuffer = apdu.getBuffer();

    if (authenticationDone &&
        ((apduBuffer[ISO7816.OFFSET_CLA] & 0x0F) == 4))
            enciphered = true;
    else
            enciphered = false;

    switch (apduBuffer[ISO7816.OFFSET_INS]) {
            case INS_VERIFY_PIN :
            if (apduBuffer[ISO7816.OFFSET_CLA] == CLA_OPPURSE ||
               apduBuffer[ISO7816.OFFSET_CLA] == CLA_OPPURSE_SM)
                  verifyPIN(apdu);
...
```

Using smart-cards - MULTOS cards SmartDec tools



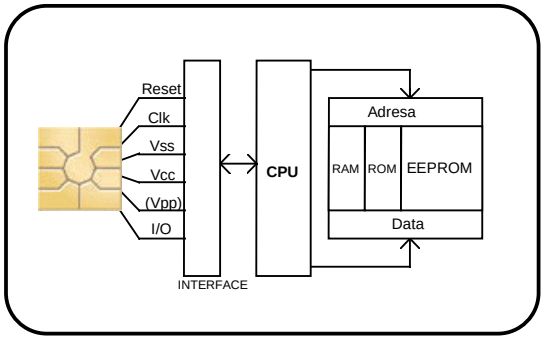
Programming in
- assembler
- Java
- C

full debugger with
- breaks
- memory dumps
- register values

Smart-card hacking

- harm caused to chip (smart-card) itself
  - non-invasion
  - semi-invasion
  - invasion
- type of activity we have to produce
  - passive – we are only watching and we do not interfere correct behaviour and I/O operations
  - active – we interfere with device to produce computational errors

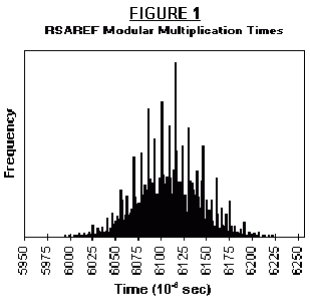
Side-channel analysis

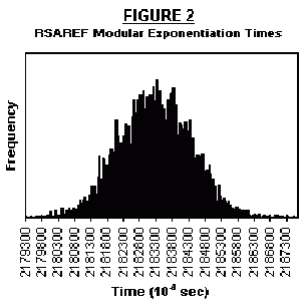


Signal	Model	Attack
reset	0/5 V for time in ms	*20 V*
clk	4 MHz	*20 MHz*
Vcc	5V stabilized DC voltage	*20 V*
I/O	serial protocol – 9600-~150.000 bps	*impulses >2*f*

Timing Attacks

- non-invasion attack, we are looking for secret key and we are able to force token to sign different data many times
- time needed for signature creation of random message has normal distribution
- we can use correlation for “exact” times needed for predefined data – it works even for PCs (in some circumstances)





Timing attack – Montgomery mult

- **function** ModExp (a, e, n) <=> $a^e \bmod n$
- step 1: a' = a.r mod n
- step 2: x' = 1.r mod n
- step 3: for i = j –1 downto 0 //*where j is no of bits in e*
- x = MonPro(x', x')
- ***if e_i = 1 then x = MonPro(x', a')***
- step 4: result => x = MonPro(x',1)

- **function** MonPro(a', b')
- step 1: t = a'. b'
- step 2: u = (t+(t.n' mod r).n)/r
- step 3: if (u>= n) result => u-m, else result => u

Electromagnetic Analysis

- non-invasion attack
- the same source of data as for attacks on screens, cables, keyboards, …
- what results can we get from EM monitoring of smart-cards
  - we can find out what parts of chip work in a given time
  - it is possible to create a chip map for further invasive attacks
  - it is possible to apply the same statistical attacks as with power analysis

Fault Analysis

- non-invasion (usually) and active attack
- the goal is to interfere with a token in such a way that it produces computational errors
  - temperature
  - power supply
  - clock
  - microwave radiation
  - long-time change of operational conditions out of permitted range
    - can be defended
  - short-time, but strong, brute changes

Fault Analysis – an example

- smartcard with RSA
  - we find time point when RSA is computed during first run
  - we create short voltage peak during the computation (4x-5x of normal voltage for several clock cycles)
  - signature is not correct and we can compute private key <- algebraic properties of RSA
- $n = p.q$
- $d.e = 1 \bmod (p-1).(q-1)$
- signature $x^d \bmod n$
  - where p and q and Chinese reminder theorem may be used to speed-up the operation
  - we can get $p = gcd(n, S^e - M)$

Attack on RSA - decryption

- q = 11, p=3, n = 33, fi = 20
- k^{-1} = (33, 3) e = 3, k = (33, 7) d = 7
- m = 10 => c=10^e mod n = 10^3 mod 33 = 10
- $v_1 = c^{d \bmod (q-1)} \bmod q = 10^1 \bmod 3 = 1$
- $v_2 = c^{d \bmod (p-1)} \bmod p = 10^7 \bmod 11 = 10$
- $c_2 = q^{-1} \bmod p = 4$, and
- $u = (v_2 - v_1)c_2 \bmod p = 9.4 \bmod 11 = 3$
- $c^d \bmod n = v_1 + u\,q = 1 + 3.3 = 10$

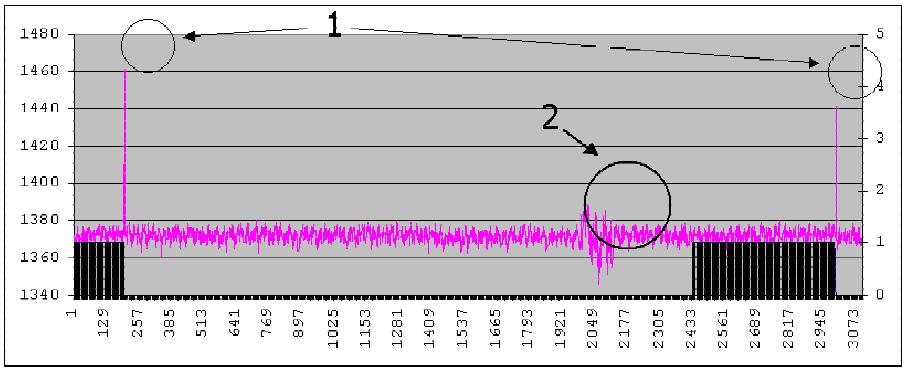
correct computation

1 bit error

- $v_2 = 8$
- $c_2 = 4$
- $u = (8 - 1).4 = 28 \bmod 11 = 7$
- $c^d \bmod n = v_1 + u\,q = 1 + 7.3 = 22$
- $p = gcd(n, S^e - M) = gcd(33, 22\text{-}10) = gcd(33,12) = 3$

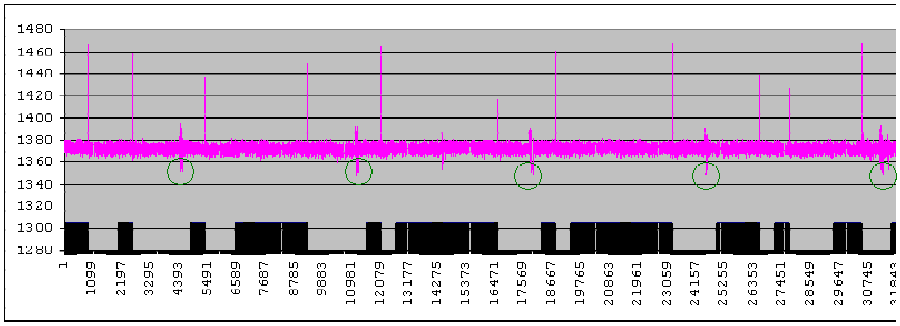
12=2.2.3 – small factors

Power analysis
16bit “very safe” Crypto Smart-card



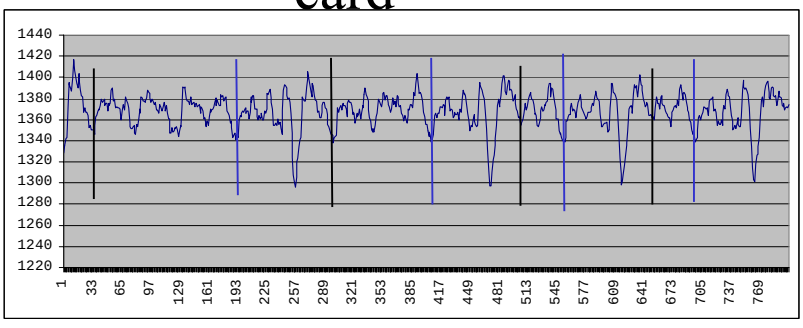
serial communication
1. peaks when physical bit “1” is to be transmitted
2. power consumption change after byte transmition

Power analysis
16bit “very safe” Crypto Smart-card



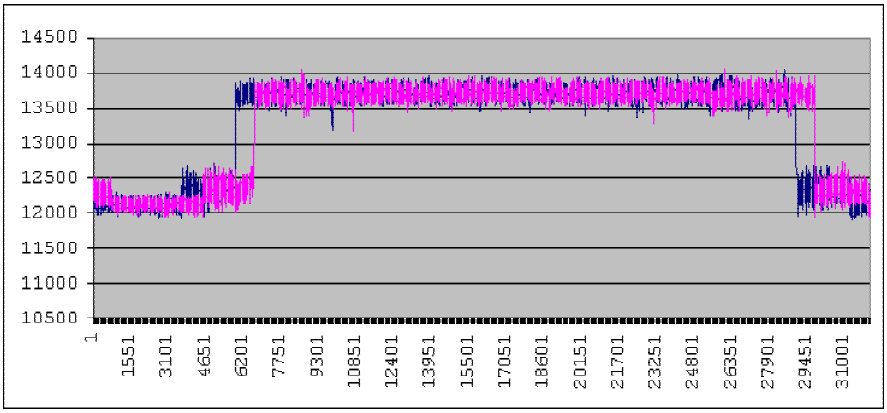
global sight on serial communication

16bit “very safe” Crypto Smart-card

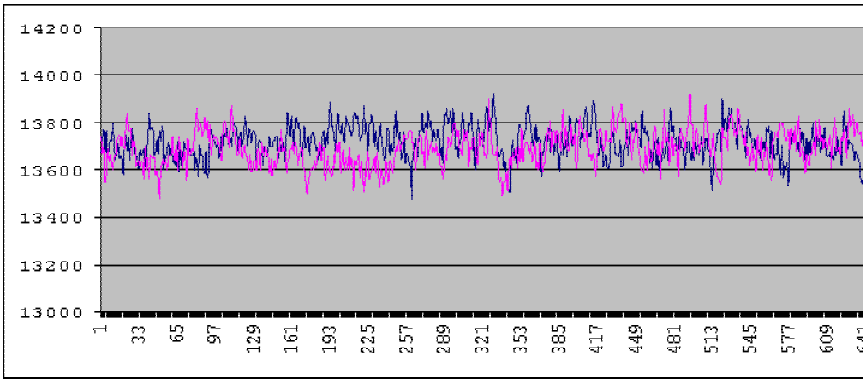


several rounds of symmetric cipher can be detected

Another card …



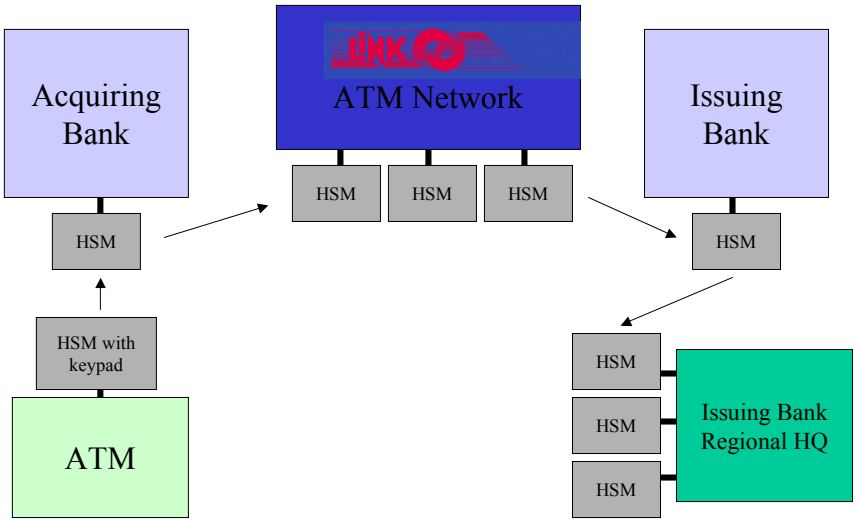
… detail – 1 bit key change



Introduction

- Basic terminology
  - Hardware Security Module (HSM)
    - Example: IBM 4758 (depicted below)
  - Host device
  - Application Programming Interface (API)
  - Attack
    - PIN Recovery Attacks
  - Clear PIN-block (CPB)
  - Encrypted PIN-block (EPB)
  - Personal Account Number (PAN)
- Insufficient checking of function parameters

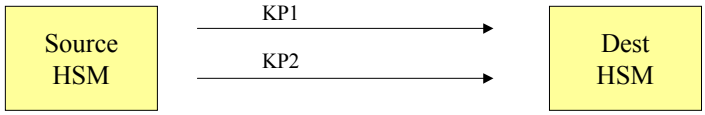
Secure Hardware in Banks



source: Mike Bond

XOR To Null Key Attack

- Top-level crypto keys exchanged between banks in several parts carried by separate couriers, which are recombined using the exclusive-OR function



Repeat twice…

```
User->HSM    : Generate Key Component
HSM->Printer : KP1
HSM->User    : { KP1 }ZCMK
```

Combine components…

```
User->HSM    : { KP1 }ZCMK , { KP2 }ZCMK
HSM->User    : { KP1 xor KP2 }ZCMK
```

Repeat twice…

```
User->HSM    : KP1
HSM->User    : { KP1 }ZCMK
```

Combine components…

```
User->HSM    : { KP1 }ZCMK , { KP2 }ZCMK
HSM->User    : { KP1 xor KP2 }ZCMK
```


source: Mike Bond

XOR To Null Key Attack

- A single operator could feed in the same part twice, which cancels out to produce an ‘all zeroes’ test key. PINs could be extracted in the clear using this key

Combine components…

```
User->HSM    : { KP1 }ZCMK , { KP1 }ZCMK
HSM->User    : { KP1 xor KP1 }ZCMK
```

KP1 xor KP1 = 0

source: Mike Bond

Key Part Import

- Thee key-part holders, each have KPA, KPC, KPC
- Final key **K** is **KPA ⊕ KPB ⊕ KPC**
- All must collude to find K, but any one key-part holder can choose difference between desired K and actual value.

source: Mike Bond

4758 Key Import Attack

KEK1 = KORIG

KEK2 = KORIG ⊕ (old_CV ⊕ new_CV)

Normally ...

$$D_{KEK1 \oplus old\_CV}(E_{KEK1 \oplus old\_CV}(KEY)) = KEY$$

Attack ...

$$D_{KEK2 \oplus new\_CV}(E_{KEK1 \oplus old\_CV}(KEY)) = KEY$$

source: Mike Bond

PIN Generation and Verification

- 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 0100 0000 0001`
  - 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 4467 8901 2344`
  - 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)
- ECI-2 format for 4 digits PINs
  - `ECI-2 CPB = ppppprrrrrrrrrr`
- Visa-3 format for 4–12 digits PINs
  - `Visa-3 CPB = ppppFxxxxxxxxxxx`
- **ANSI X9.8 format for 4–12 digits PINs**
  - P_1 = `Zlpppppfffffffff`
  - P_2 = `ZZZZaaaaaaaaaaaa`
  - `ANSI X9.8 CPB =` P_1 `xor` P_2

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

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 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