PV204 Security technologies

Trusted element, side channels attacks

Petr Švenda svenda@fi.muni.cz
Faculty of Informatics, Masaryk University

CROCS
Centre for Research on Cryptography and Security

COURSE TRIVIA

Introduction

- *See PV204_overview.ppt*

TRUSTED ELEMENT

What is “Trusted” system (plain language)

- Many different notions

1. System trusted by someone
2. System that you can’t verify and therefore must trust not to betray you
   - If a trusted component fails, security can be violated
3. System build according to rigorous criteria so you are willing to trust it
4. …

We need more precise specification of Trust

- Why Trust is Bad for Security, D. Gollman, 2006
  - http://www.sciencedirect.com/science/journal/15710661/157/3

UNTRUSTED
VS.
TRUSTED
VS.
TRUSTWORTHY

Untrusted system

- System explicitly unable to fulfill specified security policy
- Additional layer of protection must be employed
  - E.g., Encryption of data before storage
  - E.g., Digital signature of email before send over network

Trusted system

- *"…system that is relied upon to a specified extent to enforce a specified security policy. As such, a trusted system is one whose failure may break a specified security policy."* (TCSEC, Orange Book)
- Trusted subjects are those excepted from mandatory security policies (Bell LaPadula model)
- User must trust (if likes to use the system)
  - E.g., your bank

Trustworthy system (computer)

- Computer system where software, hardware, and *procedures are secure, available and functional and adhere to security practices*
- User have reasons to trust reasonably
- Trustworthiness is subjective
  - Limited interface and hardware protections can increase trustworthiness (e.g., append-only log server)
- Example: Payment card - Trusted? Trustworthy?

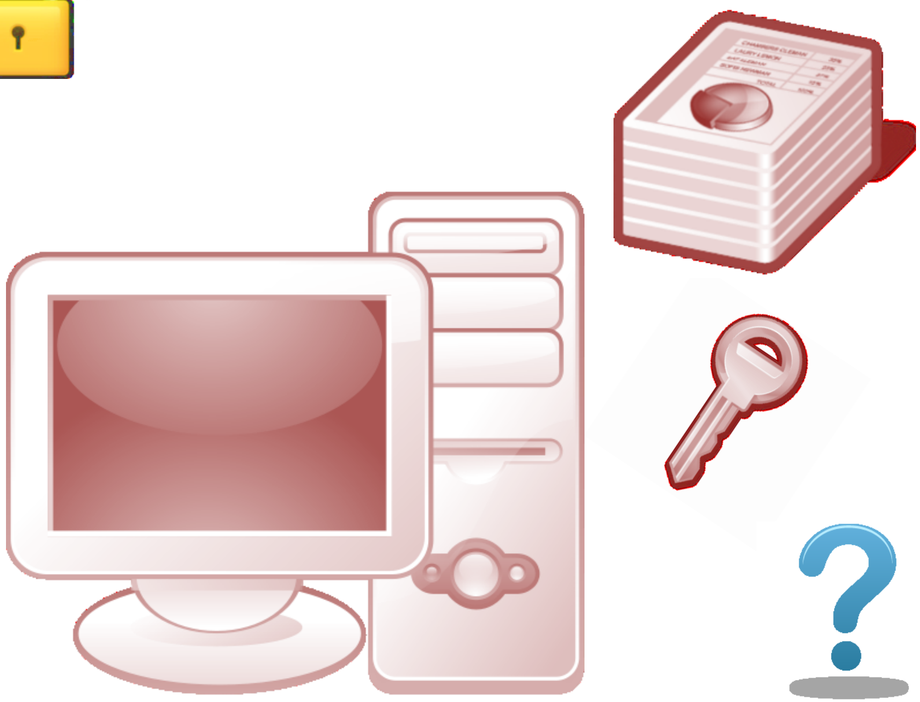
Trusted does not mean automatically *Trustworthy*

Trusted computing base (TCB)

- The set of all hardware, firmware, and/or software components that are critical to its security
- The vulnerabilities inside TCB might breach the security properties of the entire system
  - E.g., server hardware + virtualization (VM) software
- The boundary of TCB is relevant to usage scenario
  - TCB for datacentre admin is around hw + VM (to protect against compromise of underlying hardware and services)
  - TCB for web server client also contains Apache web server
- Very important factor is size and attack surface of TCB
  - Bigger size implies more space for bugs and vulnerabilities

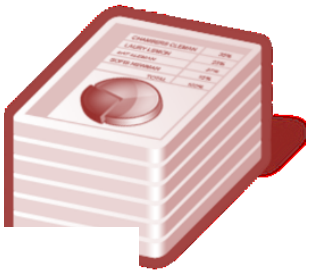
https://en.wikipedia.org/wiki/Trusted_computing_base

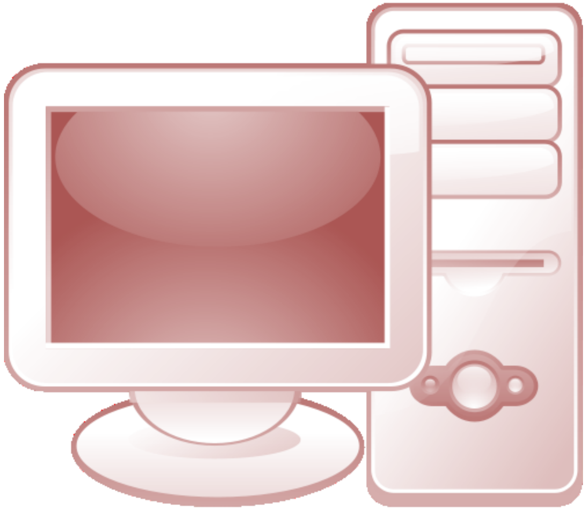
Cryptography on client



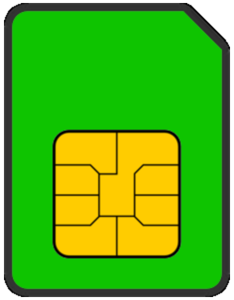
Which parts are trusted?
What are threads?
What are attacker models?
What is trusted computing base?

On client, but with secure hardware





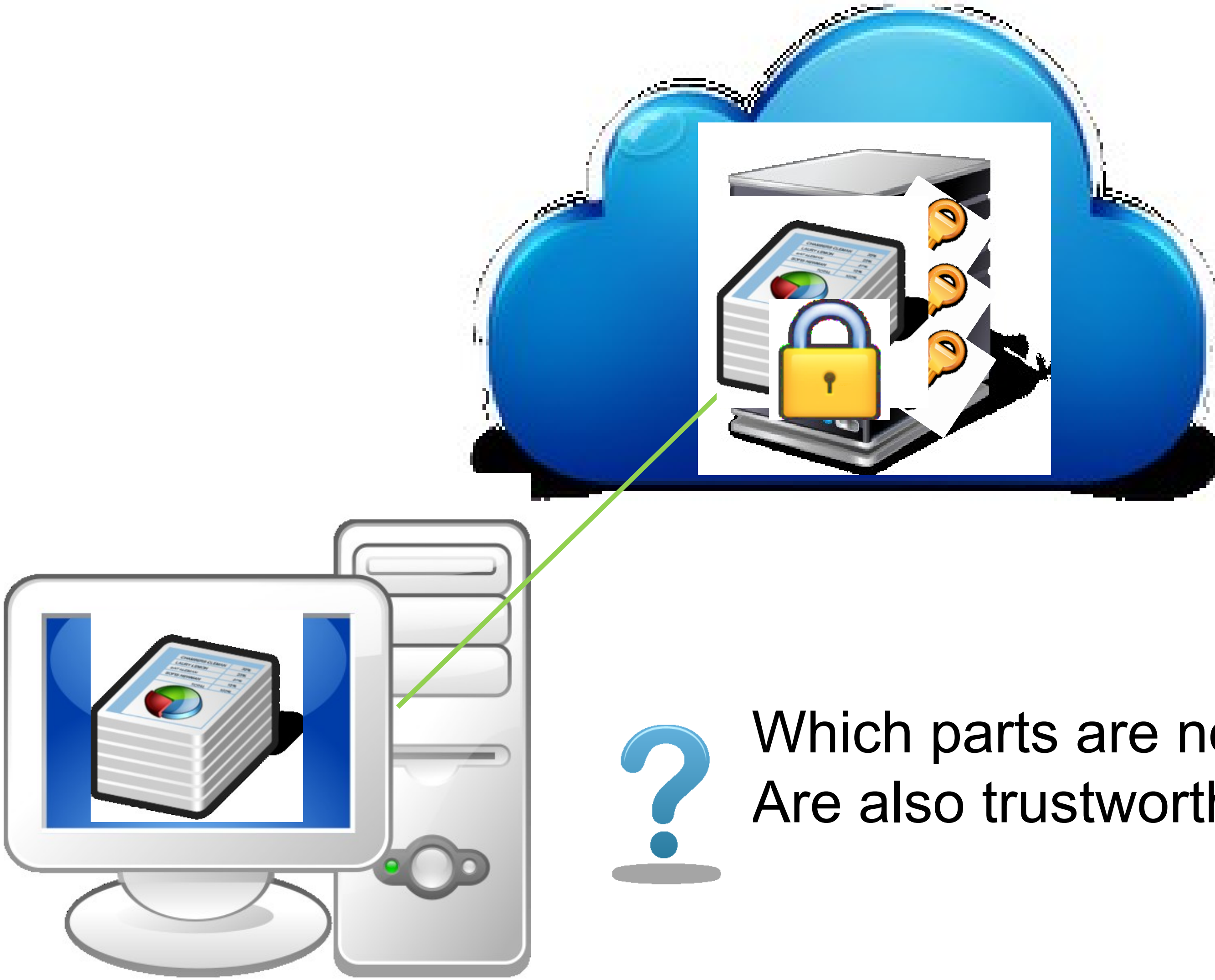
Which parts are trusted?
What are threads?
What are attacker models?
What is trusted computing base?



Cryptograpl

Which parts are trusted?
What are threads?
What are attacker models?
What is trusted computing base?

Cryptography in cloud in secure hardware



Which parts are now trusted?
Are also trustworthy?

TRUSTED ELEMENT

What exactly can be trusted element (TE)?

- Recall: Anything user entity of TE is willing to trust ☺
  - Depends on definition of “trust” and definition of “element”
  - We will use narrower definition
- Trusted element is element (hardware, software or both) in the system intended to increase security *level* w.r.t. situation without the presence of such element
  1. By storage of sensitive information (keys, measured values)
  2. By enforcing integrity of execution of operation (firmware update)
  3. By performing computation with confidential data (DRM)
  4. By providing unforged reporting from untrusted environment
  5. …

Typical examples

- Payment smart card
  - TE for issuing bank
- SIM card
  - TE for phone carriers
- Trusted Platform Module (TPM)
  - TE for user as storage of Bitlocker keys, TE for remote entity during attestation
- Trusted Execution Environment in mobile/set-top box
  - TE for issuer for confidentiality and integrity of code
- Hardware Security Module for TLS keys
  - TE for web admin
- Energy meter
  - TE for utility company
- Server under control of service provider
  - TE for user – private data, TE for provider – business operation

For whom is TE trusted?

Risk management

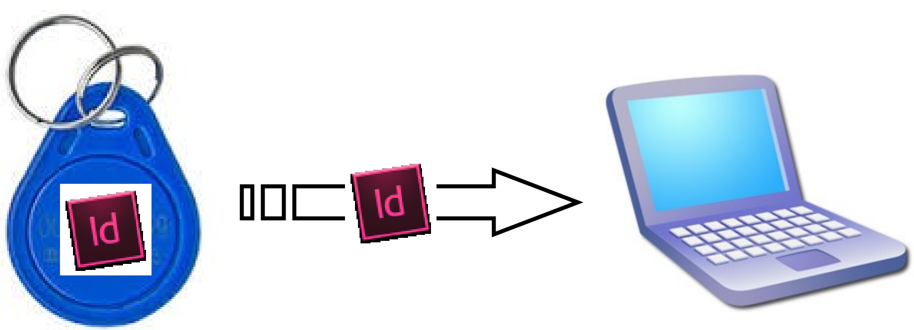
- No system is completely secure (→ risk is present)
- Risk management allows to evaluate and eventually take additional protection measures
- Example: payment transaction limit
  – My account/card will never be compromised vs. even if compromised, then loss is bounded
- Example: medical database
  – central governmental DB vs. doctor’s local DB

Good design practice is to allow for risk management

TRUSTED ELEMENT MODES OF USAGE

Element carries fixed information

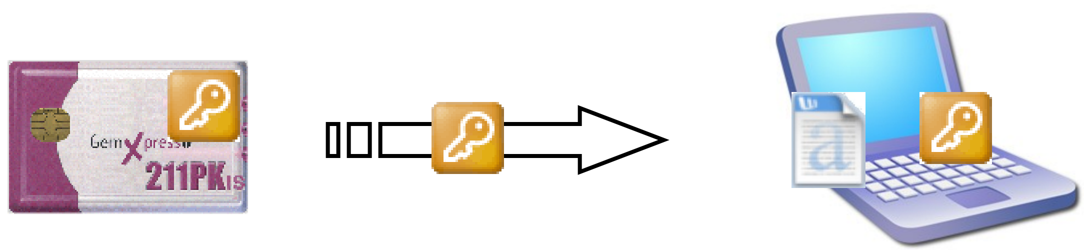
- Fixed information ID transmitted, no secure channel
- Low cost solution (nothing “smart” needed)
- Problem: Attacker can eavesdrop and clone chip



Element is trusted with ID carriage
But is it trustworthy?

Element as a secure carrier

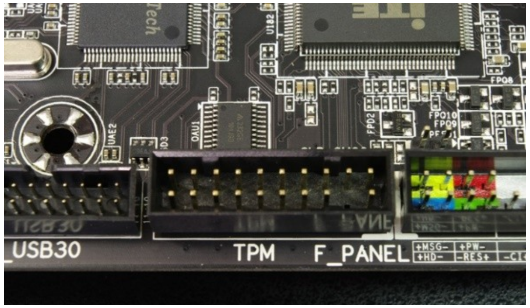
- Key(s) stored on a card, loaded to a PC before encryption/signing/authentication, then erased
- High speed usage of key possible (>>MB/sec)
- Attacker with an access to PC during operation will obtain the key
  - key protected for transport, but not during the usage

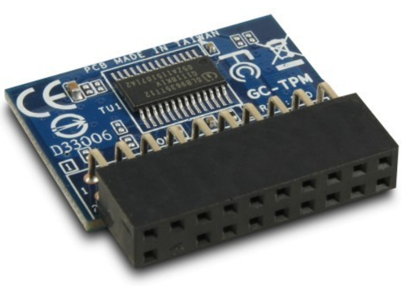


Element is trusted as confidential key storage, but cannot perform (or not trusted with) operation

Element as root of trust (TPM)

- Secure boot process, remote attestation
- Element provides robust storage with integrity
- Application can verify before pass control (measured boot)
- Computer can authenticate with remote entity…





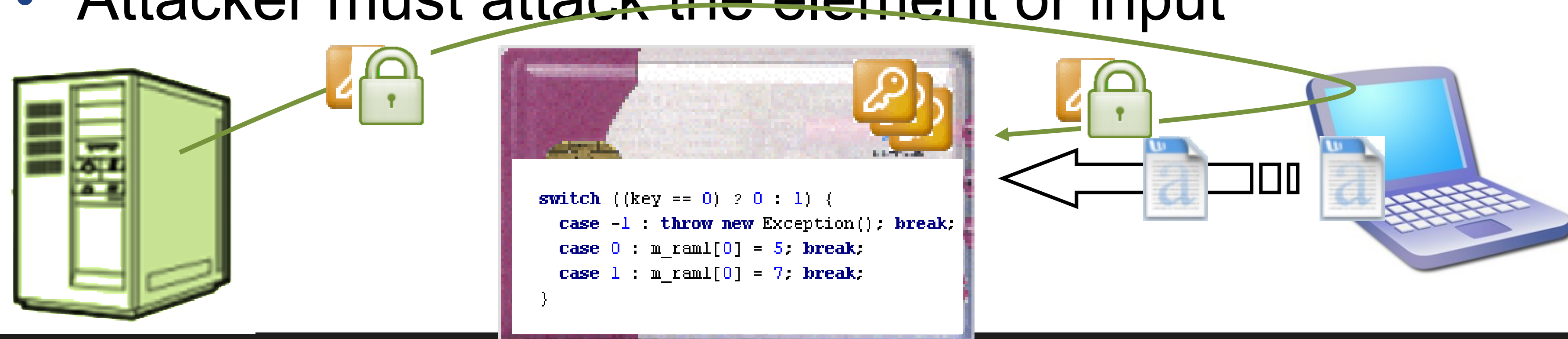
Element is trusted with integrity of stored values

Element as encryption/signing device

- PC just sends data for encryption/signing…
- Key never leaves element
  - personalized in secure environment
  - protected during transport and usage
- Attacker must attack the element
  - or wait until card is inserted and PIN entered!
- Potentially low speed encryption (~kB/sec)
  - low communication speed / limited element performance

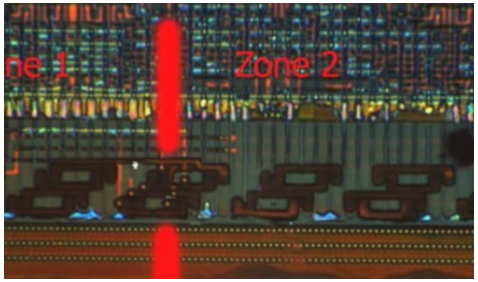
Element as computational device

- PC just sends input for application on smart card
- Application code & keys never leave the element
  - Element can do complicated programmable actions
  - Can open secure channels to other entity
    - secure server, trusted time service…
    - PC act as a transparent relay only (no access to data)
- Attacker must attack the element or input



ATTACKS AGAINST TRUSTED ELEMENT

Trusted hardware (TE) is not panacea!



1. Can be physically attacked
   – Christopher Tarnovsky, BlackHat 2010
   – Infineon SLE 66 CL PE TPM chip, bus read by tiny probes
   – 9 months to carry attack, $200k
   – https://youtu.be/w7PT0nrK2BE (great video with details)
2. Attacked via vulnerable API implementation
   – IBM 4758 HSM (Export long key under short DES one)
3. Provides trusted anchor != trustworthy system
   – weakness can be introduced later
   – E.g., bug in securely updated firmware

How to reason about attack and countermeasures?

1. Where does an attack come from (principle)?
   - Understand principle
2. Different hypothesis for the attack to be practical
   - More ways how to exploit same weakness
3. Attack countermeasures by cancel of hypothesis
   - For every way you are aware of
4. Costs and benefits of the countermeasures
   - Cost of assets protected
   - Cost for attacker to perform attack
   - Cost of countermeasure

Important: Consider Break Once, Run Everywhere (BORE)

Motivation: Bell's Model 131-B2 / Sigaba

- Encryption device intended for US army, 1943
  - Oscilloscope patterns detected during usage
  - 75 % of plaintexts intercepted from 80 feets
  - Protection devised (security perimeter), but later forgot
- CIA in 1951 – recovery over ¼ mile of power lines
- Other countries also discovered the issue
  - Russia, Japan…
- More research in use of (eavesdropping) and defense against (shielding) → TEMPEST

Common and realizable attacks on TE

1. Non-invasive attacks
   - API-level attacks
     - Incorrectly designed and implemented application
     - Malfunctioning application (code bug, faulty generator)
   - Communication-level attacks
     - Observation and manipulation of communication channel
   - Side-channel attacks
     - Timing/power/EM/acoustic/cache-usage/error… analysis attacks
2. Semi-invasive attacks
   - Fault induction attacks (power/light/clock glitches…)
3. Invasive attacks
   - Dismantle chip, microprobes…

Where are frequent problems with crypto nowadays?

- Security mathematical algorithms
  - OK, we have very strong ones (AES, SHA-3, RSA…)
- Implementation of algorithm
  - Problems → implementation attacks
- Randomness for keys
  - Problems → achievable brute-force attacks
- Key distribution
  - Problems → old keys, untrusted keys, key leakage
- Operation security
  - Problems → where we are using crypto, key leakage

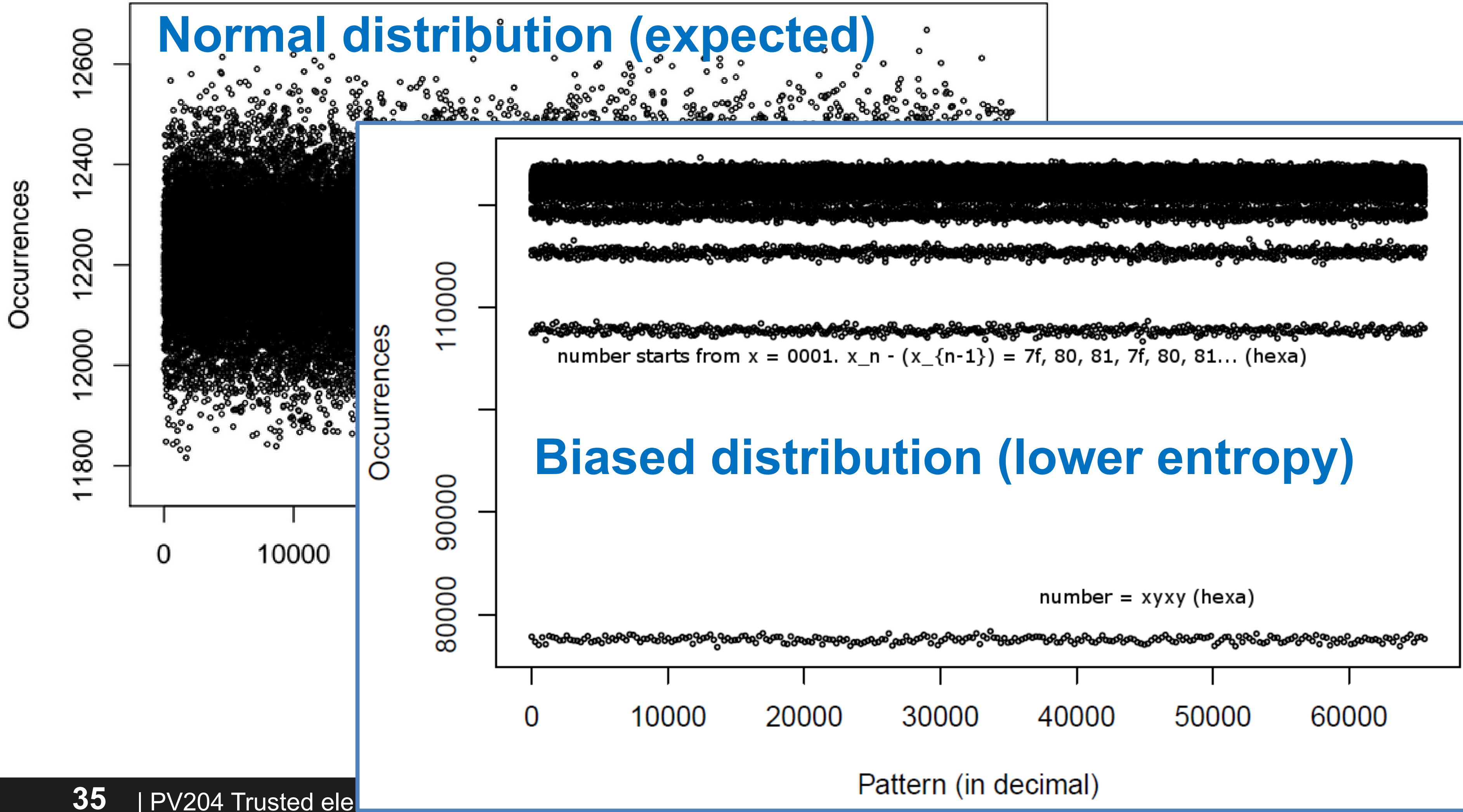
Non-invasive side-channel attacks

NON-INVASIVE ATTACKS

TRNG → Key: What if faulty TRNGs?

- Good source of randomness is critical
  - TRNG can be weak or malfunctioning
- How to inspect TRNG correctness?
  1. Analysis of TRNG implementation (but usually blackbox)
  2. Output data can be statistically tested (100MB-1GB stream, NIST STS, Dieharder, TestU01 batteries) http://www.phy.duke.edu/~rgb/General/dieharder.php
  3. Behaviour in extreme condition (+70/-50° C, radiation…)
     - Analyse data stream gathered during extreme conditions
  4. Simple power analysis of TRNG generation
     - Is hidden/unknown operation present?

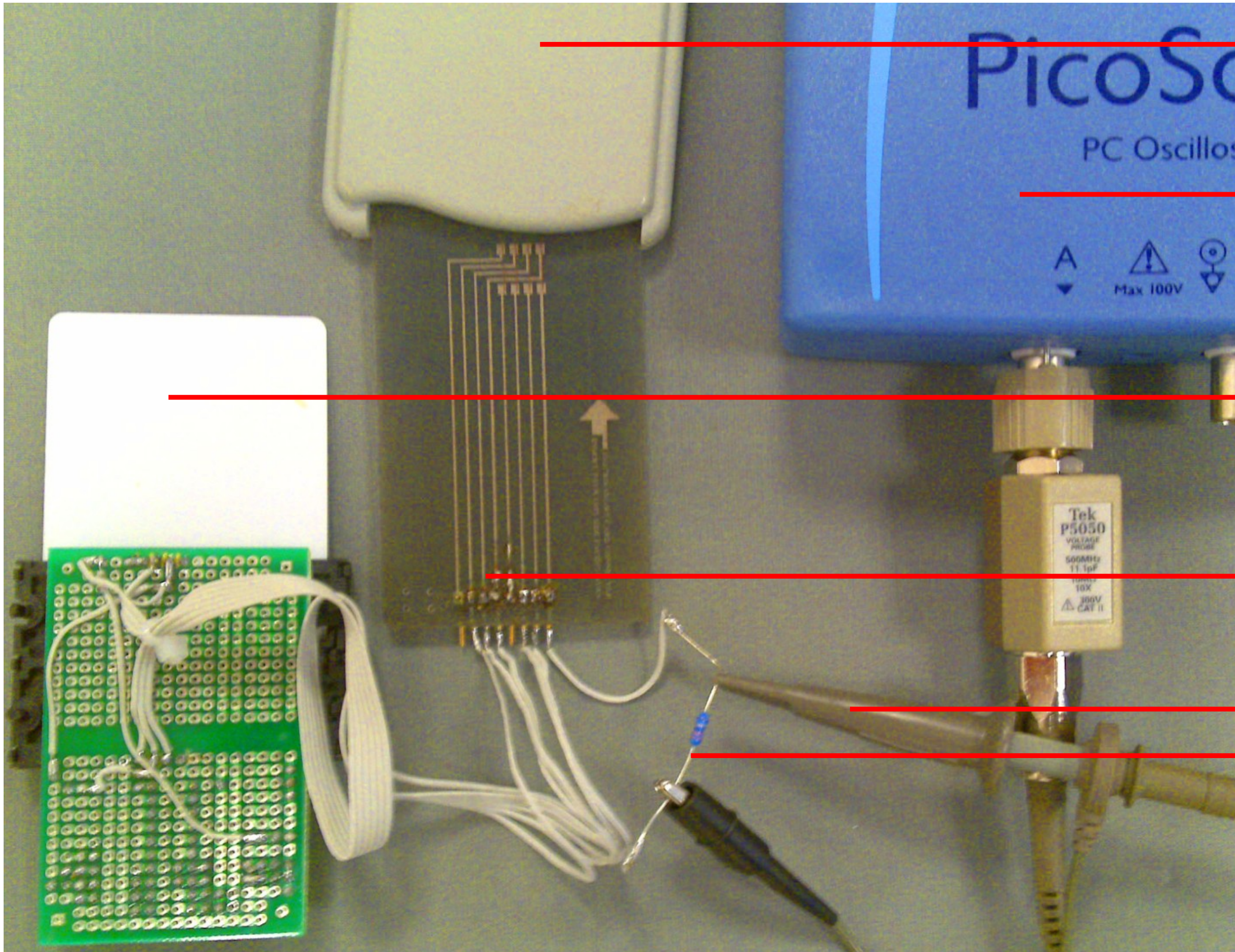
Serial test: Histogram of 16bits patterns



Non-invasive side-channel attacks

POWER ANALYSIS

Basic setup for power analysis



Smart card reader

Oscilloscope

Smart card

Inverse card connector

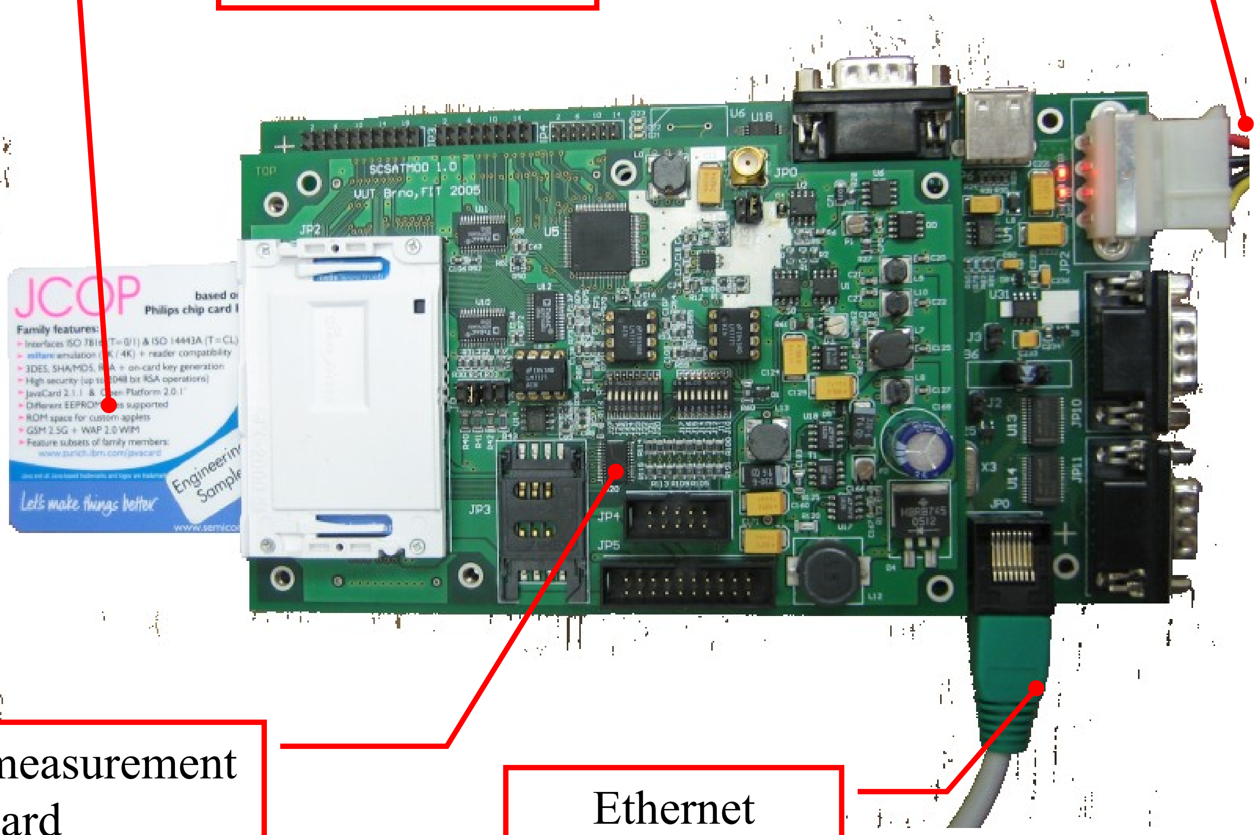
Probe

Resistor 20-80 ohm

More advanced setup for power analysis

Tested smartcard

External power supply



SCSAT04 measurement board

Ethernet

Simple vs. differential power analysis

- Simple power analysis
  - Direct observation of single / few power traces
  - Visible operation => reverse engineering
  - Visible patterns => data dependency
- Differential power analysis
  - Statistical processing of many power traces
  - More subtle data dependencies found

Reverse engineering of Java Card bytecode

- Goal: obtain code back from smart card
  - JavaCard defines around 140 bytecode instructions
  - JVM fetch instruction and execute it

(source code)

```
m_ram1[0] = (byte) (m_ram1[0] % 1);
```

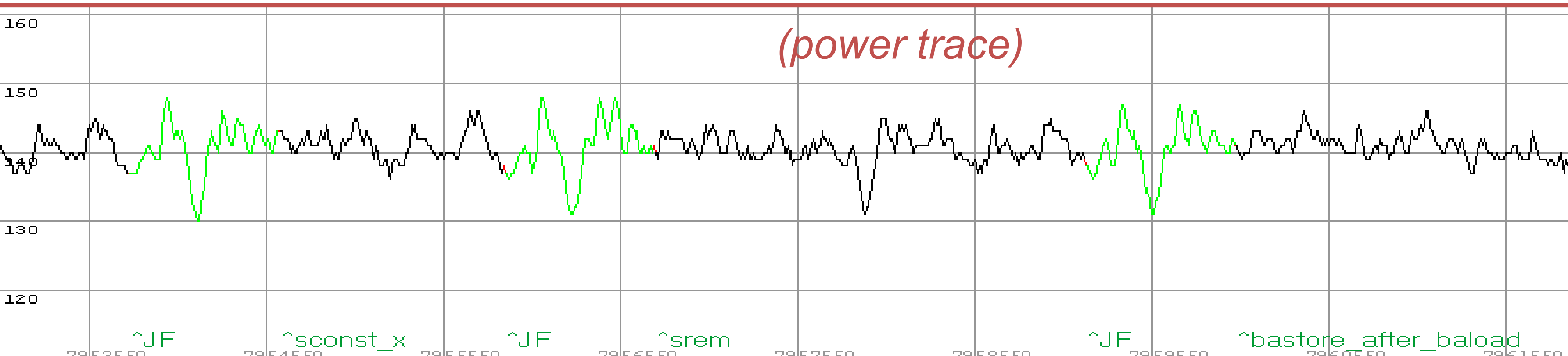
compiler →

(bytecode)

```
getfield_a_this 0;
sconst_0;
baload;
sconst_1;
srem;
bastore;
```

oscilloscope →

(power trace)



Conditional jumps

- may reveal sensitive info
- keys, internal branches, …

(source code)

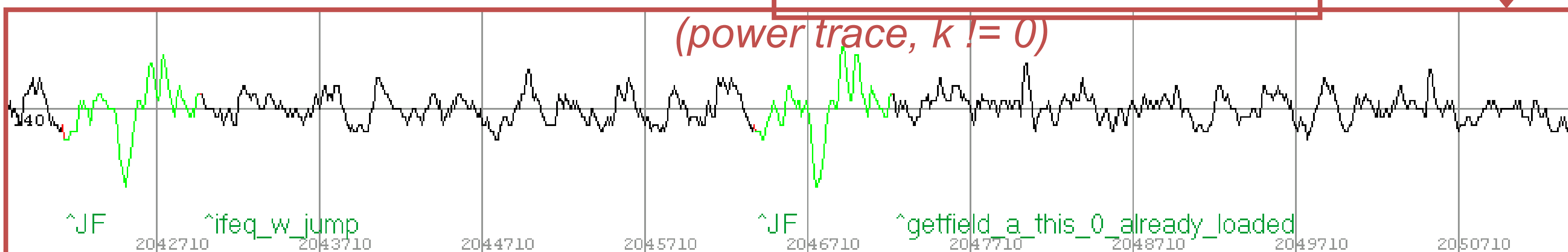
```
if (key == 0) m_ram1[0] = 1;
else m_ram1[0] = 0;
```

compiler →

(bytecode)

```
    sload_1;
    ifeq_w L2;
L1: getfield_a_this 0;
    sconst_0;
    sconst_0;
    bastore;
    goto L3;
L2: getfield_a_this 0;
    sconst_0;
    sconst_1;
    bastore;
    goto L3;
L3: …
```

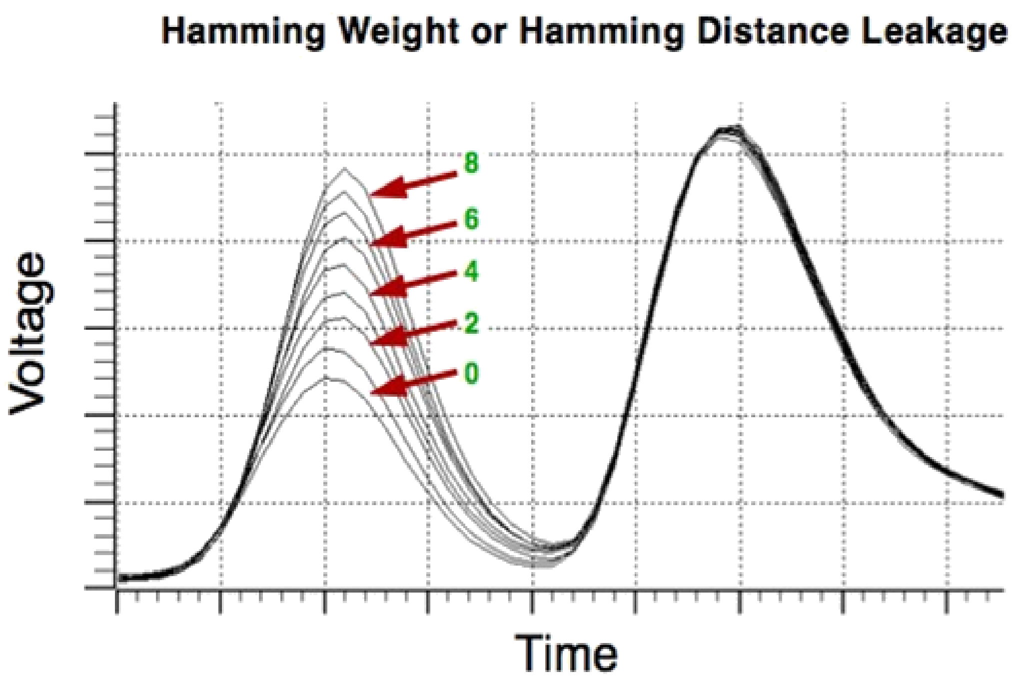
oscilloscope



Can you use timing attack?

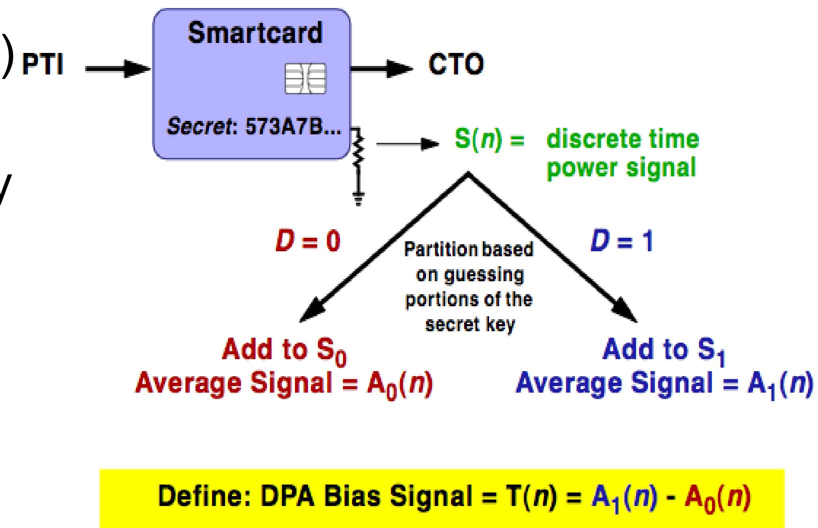
Simple power analysis – data leakage

- Data revealed directly when processed
  - e.g., Hamming weight of instruction argument
    - hamming weight of separate bytes of key ($2^{56} \rightarrow 2^{38}$)

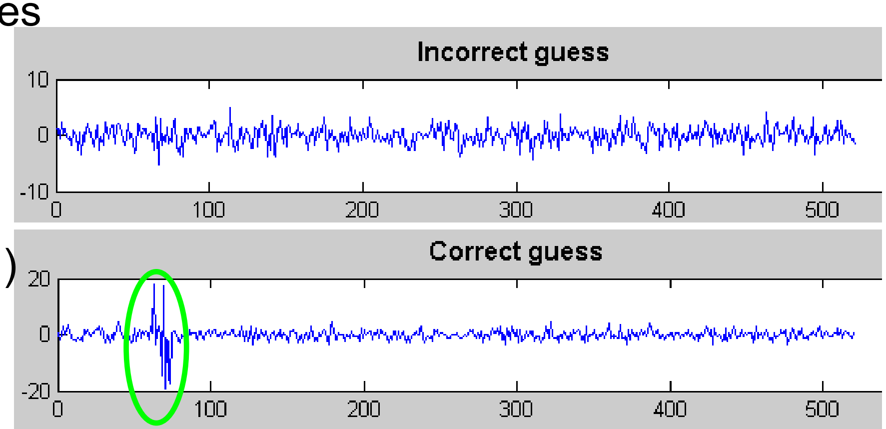


Differential power analysis

- Very Powerful attack on secret values (keys)
  - E.g., KEY ⊕ INPUT_DATA



1. Obtain multiple power traces with (fixed) key usage and variable data
   - 10^3-10^5 traces with known I/O data => S(n)
   - KEY ⊕ KNOWN_DATA
2. Guess key byte-per-byte
   - All possible values of single byte tried (256)
   - D = HammWeight(KEY ⊕ KNOWN_DATA > 4)
   - Correct guess reveals correlation with traces
   - Incorrect guess not
3. Divide and test approach
   - Traces divided into 2 groups
   - Groups are averaged A_0,A_1 (noise reduced)
   - Subtract group's averaged signals T(n)
   - Significant peaks if guess was correct



- No need for knowledge of exact implementation big advantage

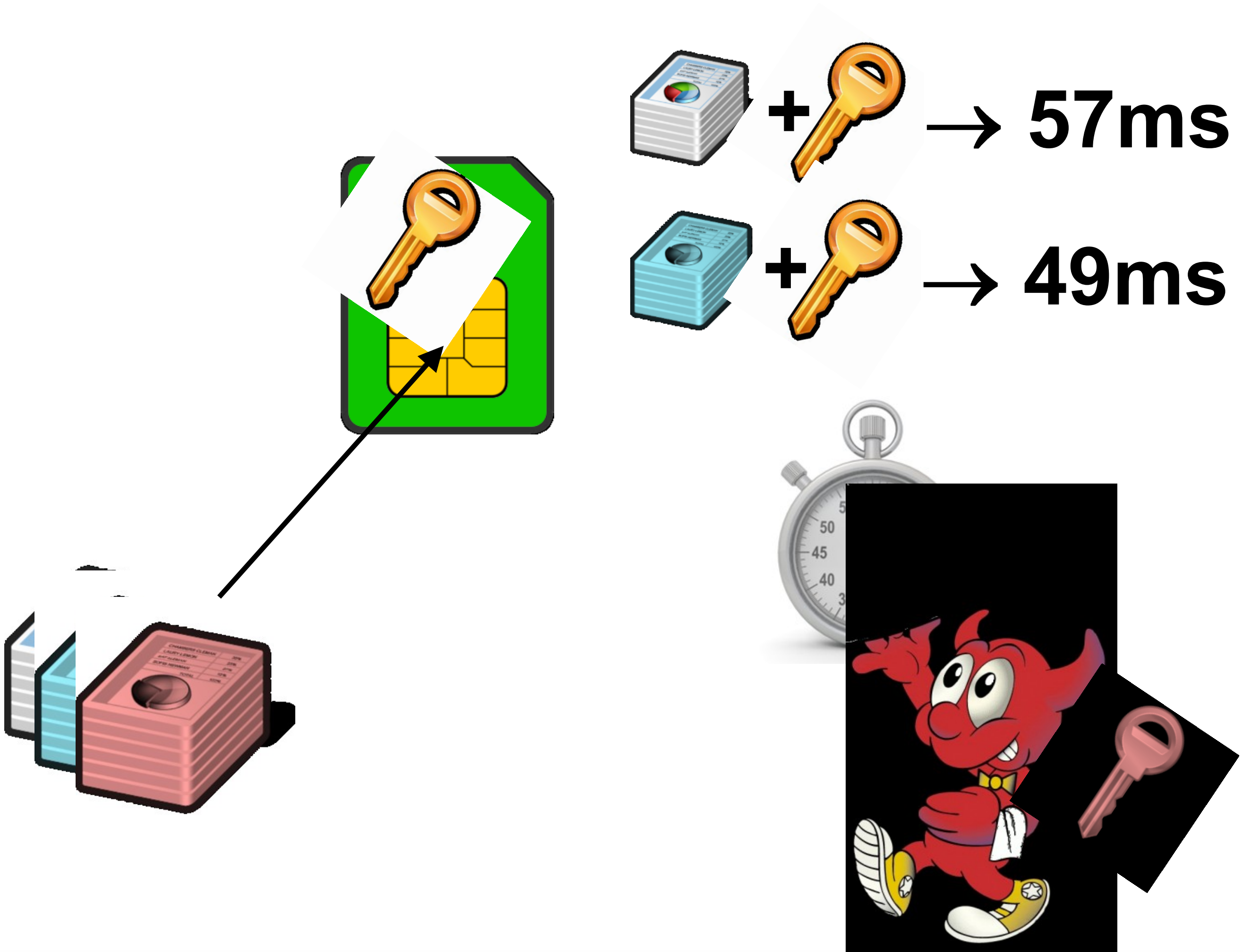
Tool: DPA simulator

- Generate simulated DPA traces
- Perform DPA
- Can be used to inspect influence of noise, number of traces…
- https://github.com/crocs-muni/PowerTraceSimulator

Non-invasive side-channel attacks

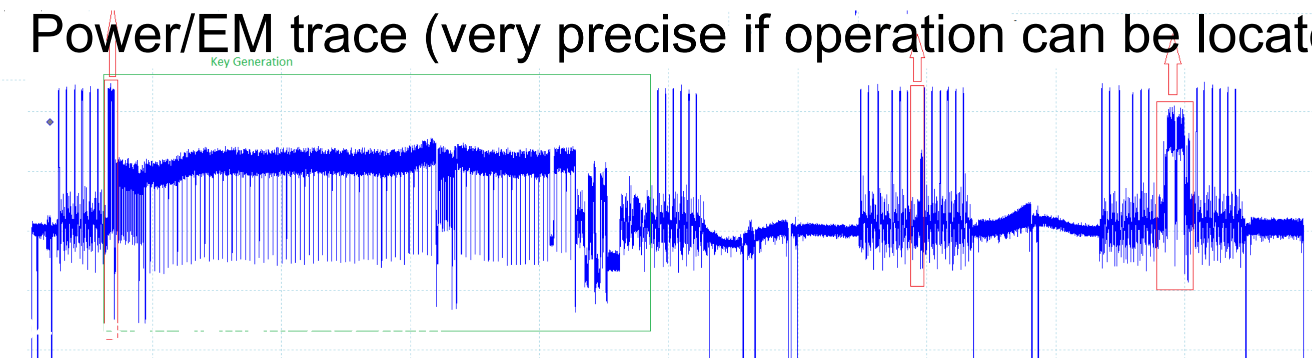
TIMING ATTACKS

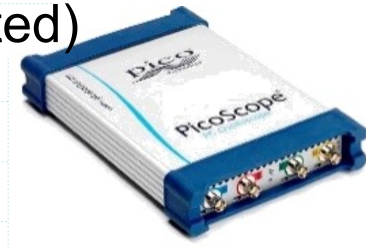
Timing attack: principle



Timing attacks

- Execution of crypto algorithm takes different time to process input data with some dependence on secret value (secret/private key)
  1. Due to performance optimizations (developer, compiler)
  2. Due to conditional statements (branching)
  3. Due to cache misses
  4. Due to operations taking different number of cycles
- Measurement techniques
  1. Start/stop time (aggregated time, local/remote measurement)
  2. Power/EM trace (very precise if operation can be located)





Naïve modular exponentiation (RSA/DH)

- $M = C^d \bmod N$

Is there dependency of time on secret value?

- $M = \overbrace{C * C * C * \ldots * C}^{\text{d-times}} \bmod N$

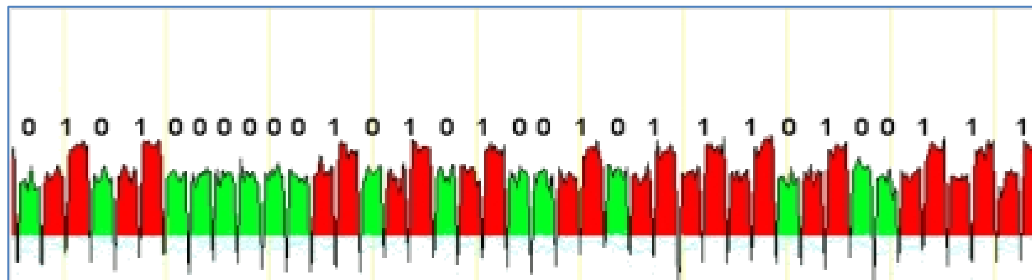
- Easy, but extremely slow for large d (1000s bits)
  - Faster algorithms exist

Square and multiply algorithm

Executed always

Executed only when d_j == 1

```
// M = C^d mod N
// Square and multiply algorithm
x = C                      // start with ciphertext
for j = 1 to n {           // process all bits of private exponent
    x = x*x mod N          // shift to next bit by x * x (always)
    if (d_j == 1) {        // j-th bit of private exponent d
      x = x*C mod N        // if 1 then multiple by Ciphertext
    }
}
return x        // plaintext M
```



Gilbert Goodwill, http://www.embedded.com/print/4408435

- How to measure?
  - Exact detection from simple power trace
  - Extraction from overall time of multiple measurements

Example: Remote extraction OpenSSL RSA

- Brumley, Boneh, Remote timing attacks are practical
  - https://crypto.stanford.edu/~dabo/papers/ssl-timing.pdf
- Scenario: OpenSSL-based TLS with RSA on remote server
  - Local network, but multiple routers
  - Attacker submits multiple ciphertexts and observe processing time (client)
- OpenSSL's RSA CRT implementation
  - Square and multiply with sliding windows exponentiation
  - Modular multiplication in every step: x*y mod q (Montgomery alg.)
  - From timing can be said if normal or Karatsuba was used
    - If x and y has unequal size, normal multiplication is used (slower)
    - If x and y has equal size, Karatsuba multiplication is used (faster)
- Attacker learns bits of prime by adaptively chosen ciphertexts
  - About 300k queries needed

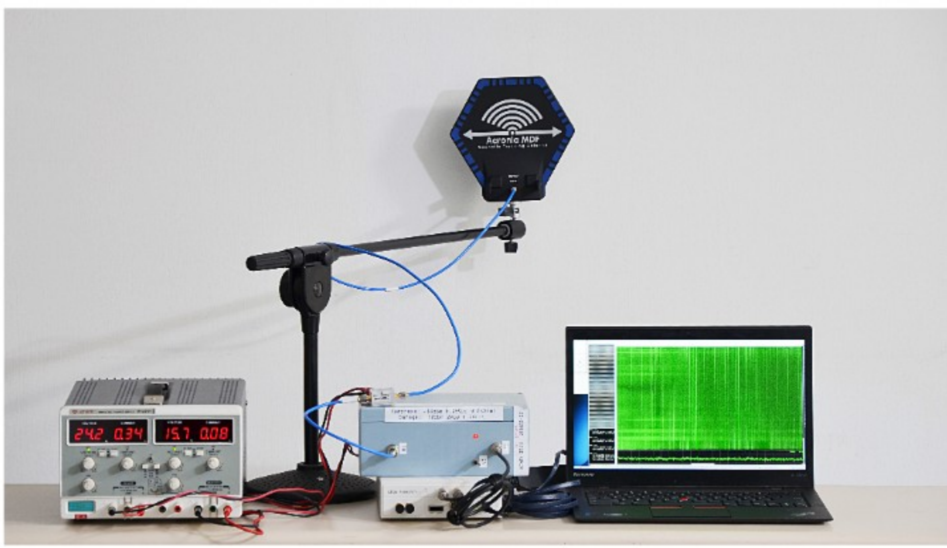
Defense introduced by OpenSSL

- RSA blinding: RSA_blinding_on()
  - https://www.openssl.org/news/secadv_20030317.txt
- Decryption without protection: $M = c^d \bmod N$
- Blinding of ciphertext *c* before decryption
  1. Generate random value *r* and compute $r^e \bmod N$
  2. Compute blinded ciphertext $b = c * r^e \bmod N$
  3. Decrypt *b* and then divide result by *r*
     - *r* is removed and only decrypted plaintext remains

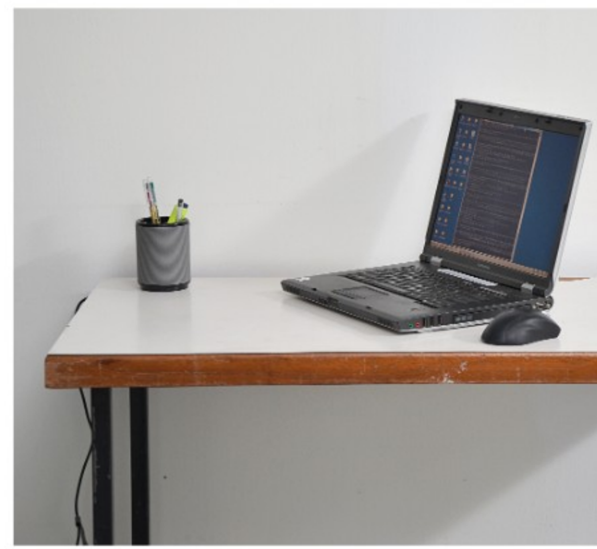
$$(r^e \cdot c)^d \cdot r^{-1} \mod n = r^{ed} \cdot r^{-1} \cdot c^d \mod n = r \cdot r^{-1} \cdot c^d \mod n = m.$$

Example: Practical TEMPEST for $3000

- ECDH Key-Extraction via Low-Bandwidth Electromagnetic Attacks on PCs
  - https://eprint.iacr.org/2016/129.pdf
- E-M trace captured (across a wall)



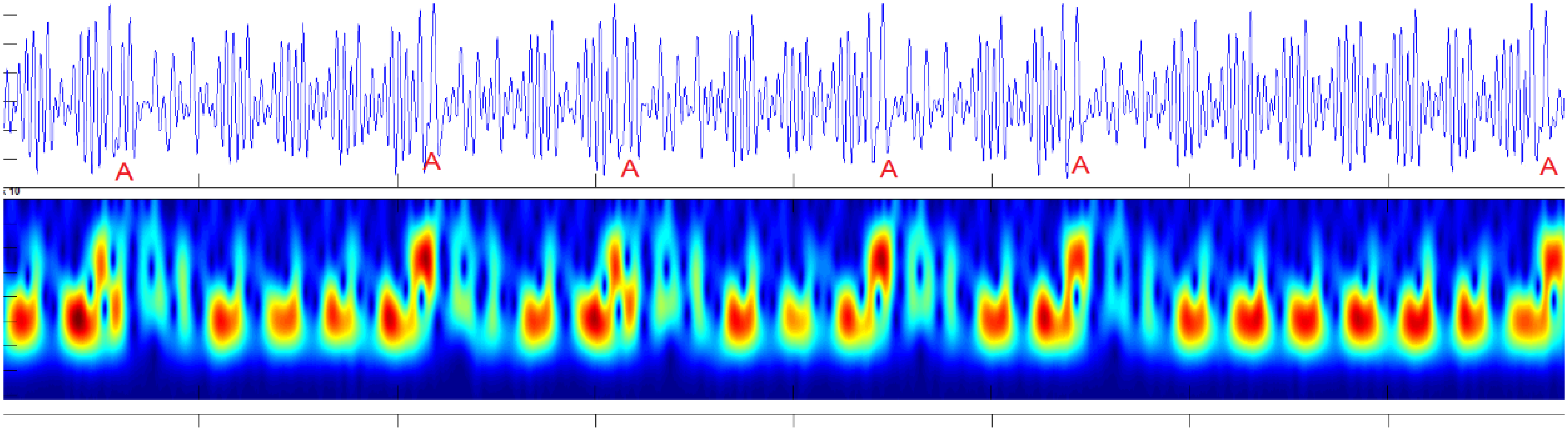
(a) Attacker's setup for capturing EM emanations. Left to right: power supply, antenna on a stand, amplifiers, software defined radio (white box), analysis computer.



(b) Target (Lenovo 3000 N200), performing ECDH decryption operations, on the other side of the wall.

Example: Practical TEMPEST for $3000

- ECDH implemented in latest GnuPG's Libgcrypt
- Single chosen ciphertext – used operands directly visible

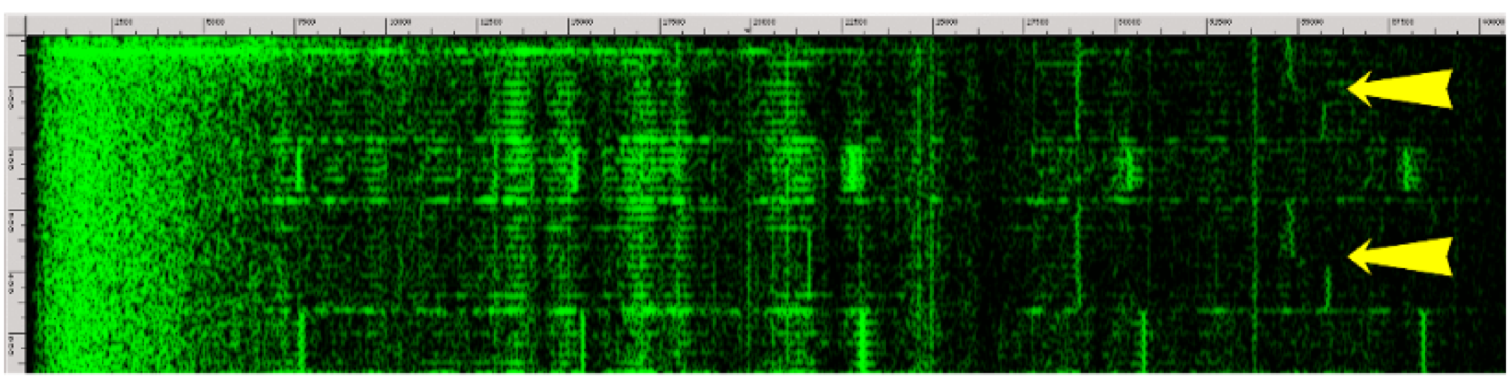


Example: How to evaluate attack severity?

- What was the cost?
  - Not high: $3000
- What was the targeted implementation?
  - Widely used implementation: latest GnuPG's Libgcrypt
- What were preconditions?
  - Physical presence, but behind the wall
- Is it possible to mitigate the attack?
  - Yes: fix in library, physical shielding of device, perimeter…
  - What is the cost of mitigation?

Example: Acoustic side channel in GnuPG

- RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis
  - Insecure RSA computation in GnuPG
  - https://www.tau.ac.il/~tromer/papers/acoustic-20131218.pdf
- Acoustic emanation used as side-channel
  - 4096-bit key extracted in one hour
  - Mobile phone 4 meters away



Example: Cache-timing attack on AES

- Attacks not limited to asymmetric cryptography
  - Daniel J. Bernstein, http://cr.yp.to/antiforgery/cachetiming-20050414.pdf
- Scenario: Operation with secret AES key on remote server
  - Key retrieved based on response time variations of table lookups cache hits/misses
  - 2^{25} x 600B random packets + 2^{27} x 400B + one minute brute-force search
- Very difficult to write high-speed but constant-time AES
  - Problem: table lookups are not constant-time
  - Not recognized by NIST during AES competition

MITIGATIONS

Generic protection techniques

1. Shielding - preventing leakage outside
   – Acoustic shielding, noisy environment
2. Creating additional “noise”
   – Parallel software load, noisy power consumption circuits
3. Compensating for leakage
   – Perform inverse computation/storage
4. Harden algorithm
   – Ciphertext blinding…

How to test real implementation?

1. Be aware of various side-channels
2. Obtain measurement for given side-channel
   – Many times (10^3 - 10^7), compute statistics
   – Same input data and key
   – Same key and different data
   – Different keys and same data…
3. Compare groups of measured data
   – Is difference visible? => potential leakage
   – Is distribution uniform? Is distribution normal?
4. Try to measure again with better precision ☺

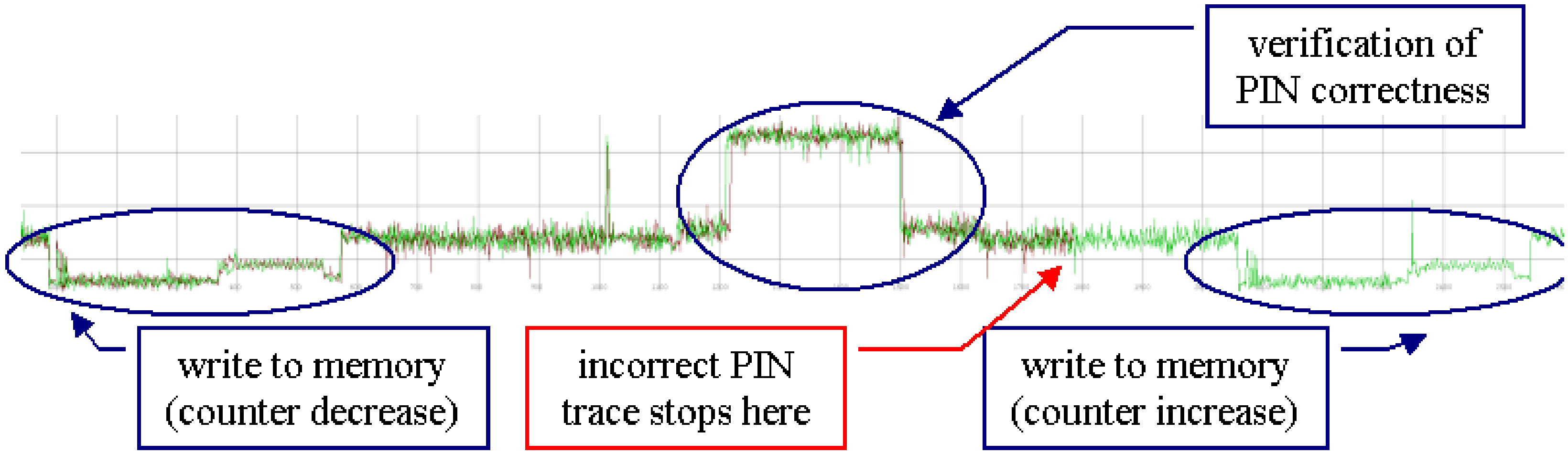
SEMI-INVASIVE ATTACKS

Semi-invasive attacks

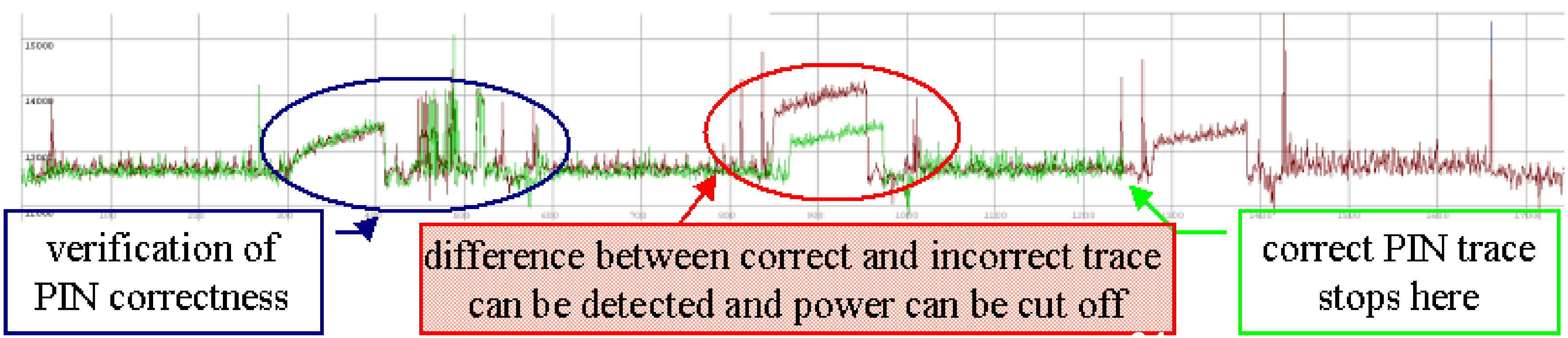
- "Physical" manipulation (but card still working)
- Micro probes placed on the bus
  - After removing epoxy layer
- Fault induction
  - liquid nitrogen, power glitches, light flashes…
  - modify memory (RAM, EEPROM), e.g., PIN counter
  - modify instruction, e.g., conditional jump

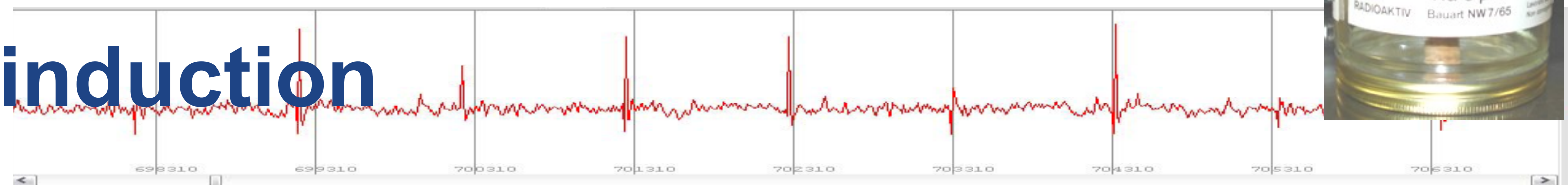
PIN verification procedure

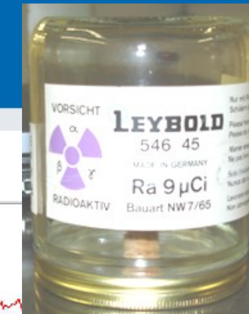
- [Decrease counter, verify, increase] - correct



- [Verify, decrease/increase]







Fault induction

- Attacker can induce bit faults in memory locations
  - power glitch, flash light, radiation...
  - harder to induce targeted then random fault
- Protection with shadow variable
  - every variable has shadow counterpart
  - shadow variable contains inverse value
  - consistency is checked every read/write to memory

01011010

10100101

a **01011010**

a_inv **10100101**

if (a != ~a_inv) Excepti
a = 0x55;
a_inv = ~0x55;

0101000

1010101

(a != ~a_inv) Exception();
= 0x13;

- Robust protection, but cumbersome for developer

CONCLUSIONS

Morale

1. Preventing implementation attacks is extra difficult
   – Naïve code is often vulnerable
     • Not aware of existing problems/attacks
   – Optimized code is often vulnerable
     • Time/power/acoustic… dependency on secret data
2. Use well-known libraries instead of own code
   – And follow security advisories and patch quickly
3. Security / mitigations are complex issues
   – Underlying hardware can leak information as well
   – Don't allow for large number of queries

Mandatory reading

- G. Goodwill, Defending against side-channel attacks
  - http://www.embedded.com/print/4408435
  - http://www.embedded.com/print/4409695
- Focus on:
  - What side channels are inspected?
  - What step in executed operation is misused for attack?
  - What are proposed defenses?

Conclusions

- Trusted element is secure anchor in a system
  - Understand why it is trusted and for whom
- Trusted element can be attacked
  - Non-invasive, semi-invasive, invasive methods
- Side-channel attacks are very powerful techniques
  - Attacks against particular implementation of algorithm
  - Attack possible even when algorithm is secure (e.g., AES)
- Use well-know libraries instead own implementation