PV204 Security technologies

Trust, trusted element, usage scenarios, side-channel attacks

Łukasz Chmielewski *chmiel@fi.muni.cz*
Centre for Research on Cryptography and Security, Masaryk University

Slides for comments (Thank you!)
https://drive.google.com/file/d/1AA_8P8IV-1o0p7mkYNIt6THa4_KLWznU/view?usp=sharing

Centre for Research on Cryptography and Security

What is untrusted, trusted and trustworthy

Trusted Element (TE)

Modes of usage of TE

Attacks against trusted element → **Protections and testing**

Logical attacks

Timing side-channels

Classical side-channel

- **Power**
- **Electromagnetic**
- **Light, etc. …**

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 wants to use the system)
  - E.g., you and your bank

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

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 (TPM)
  5. …

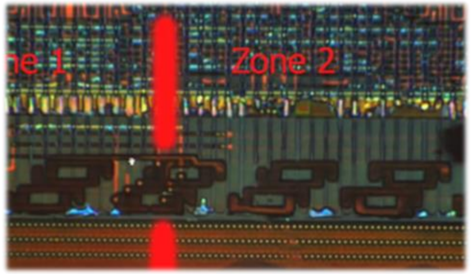
Typical examples

- 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
- Complex Scenarios: trusted element with (even more) trusted (crypto) hardware
  - TE for device manufacturer – secure derived keys, TE for chip manufacturer – secure root keys

For whom is TE trusted?

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 the attack, $200k
   – https://www.youtube.com/watch?v=WXX00tRKOlw (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 newly updated firmware

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 forgot after the war
- 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 Trusted Element

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
2. Semi-invasive attacks
   - Passive side-channel attacks
     - Timing/power/EM/acoustic/cache-usage/error… analysis attacks
   - Active side-channel attacks: fault injection
     - Power/light/clock glitches…
3. Invasive attacks
   - Dismantle chip, microprobes…

Break Once, Run Everywhere (BORE)
?

Where are the frequent problems with crypto algs nowadays?

- Security mathematical algorithms
  - OK, we have very strong ones (AES, SHA-3, RSA…) (but quantum computers)
- Post-quantum algorithms
  - Too “young”, many schemes broken or questioned recently, e.g., Rainbow
- Implementation of algorithm
  - Problems $\rightarrow$ implementation attacks
- Randomness for keys
  - Problems $\rightarrow$ achievable brute-force attacks
- Key distribution
  - Problems $\rightarrow$ old keys, untrusted keys, key leakage
- Operation security
  - Problems $\rightarrow$ where we are using crypto, key leakage

NON-INVASIVE LOGICAL ATTACKS

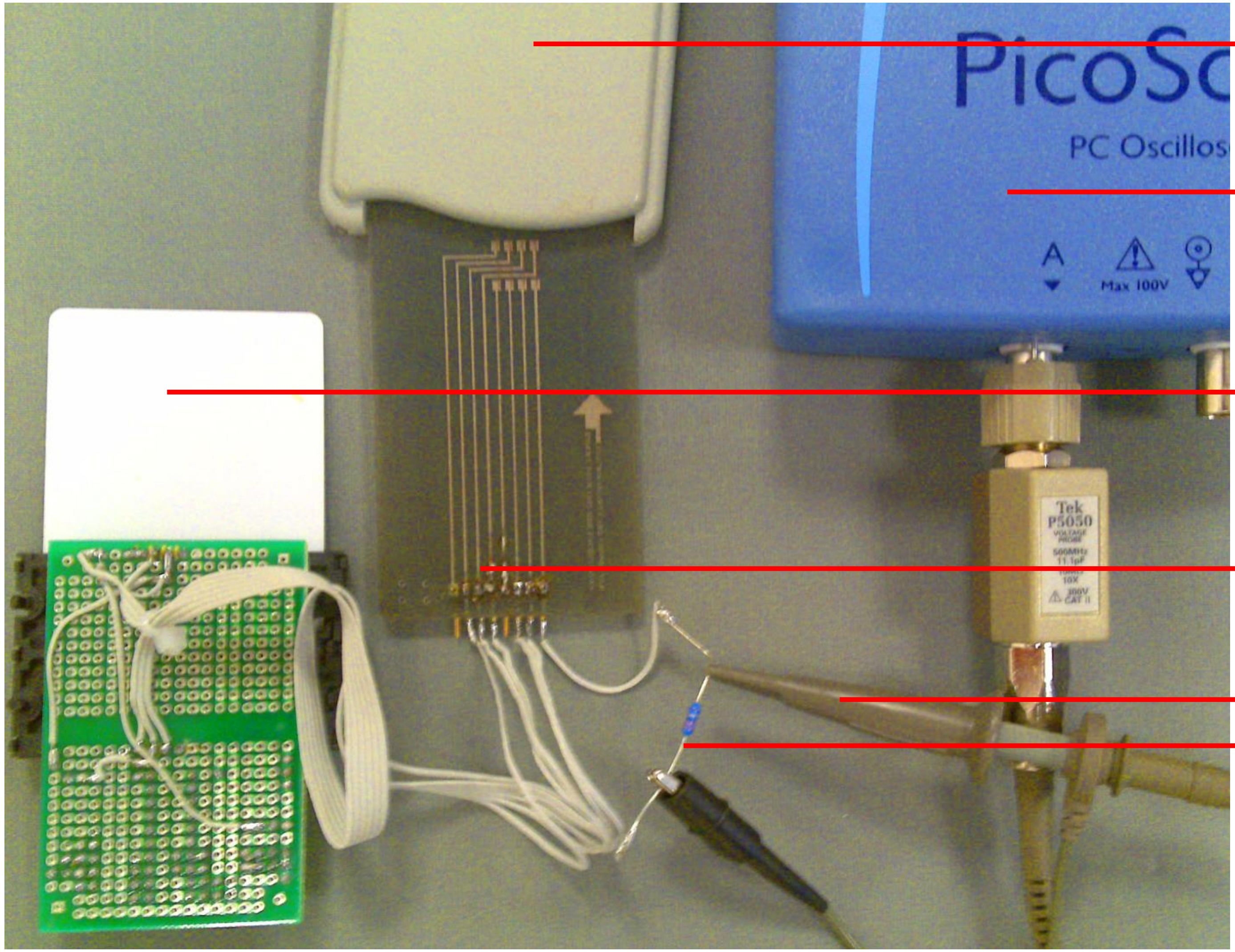
Non-complete list

- Algorithmic flaw in Infineon’s RSALib (CVE-2017-15361)
  - RSA public / private key generation on many Infineon cards (huge impact)
  - https://keychest.net/roca, https://github.com/crocs-muni/roca/
- Not enforcing secure memory protections
  - A complete exploit on Set-top Boxes
  - Presented for two ST chips, but with impact on other ST chips too
  - https://www.youtube.com/watch?v=WF1wSzTTqdg&ab_channel=HackInTheBoxSecurityConference
- Shortening Key (against hardware key stores or key ladders):
  - Using half of an AES key as a DES key or using 3DES with half of the key (i.e., single DES key)
- TEE (e.g., ARM Trustzone) issues
  - Configuration, Memory Ranges, Boot ROM…
  - https://www.slideshare.net/CristofaroMune/euskalhack-2017-secure-initialization-of-tees-when-secure-boot-falls-short
  - …

Passive Side-Channel

SIDE-CHANNEL ANALYSIS

Basic setup for power analysis



Smart card reader

Oscilloscope

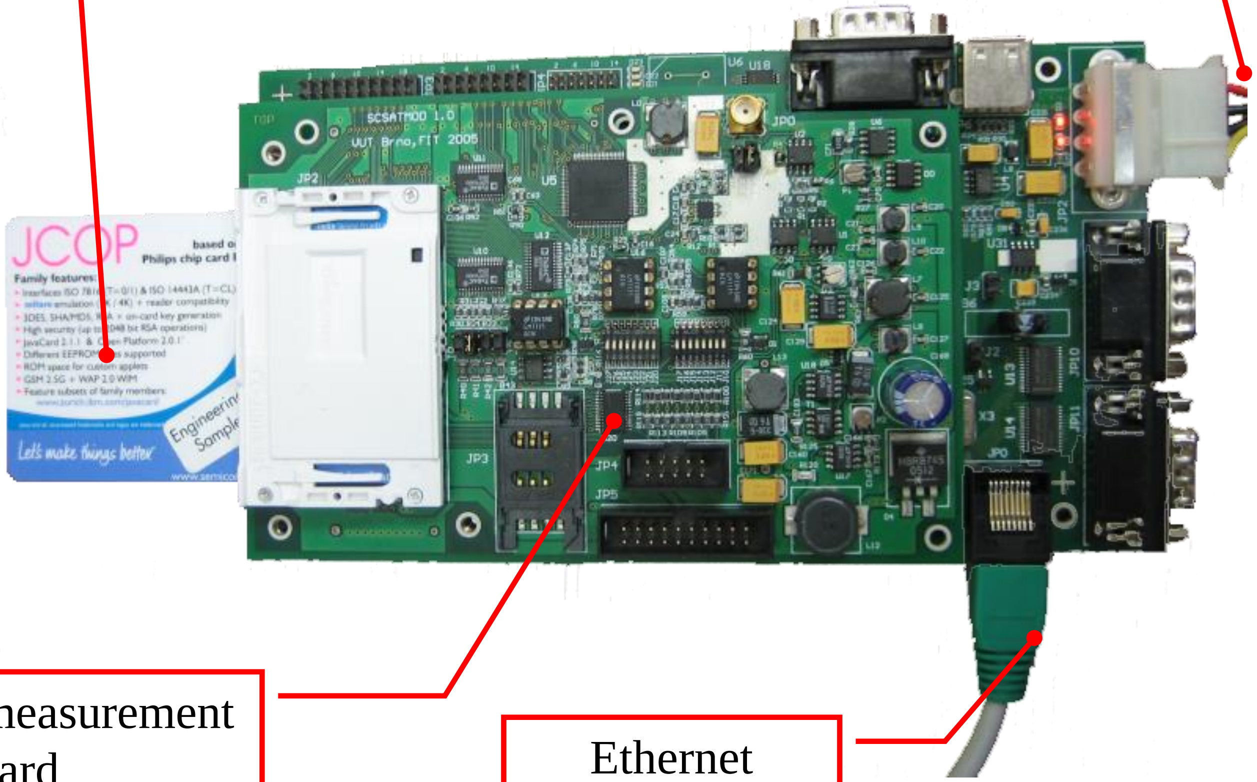
Smart card

Inverse card connector

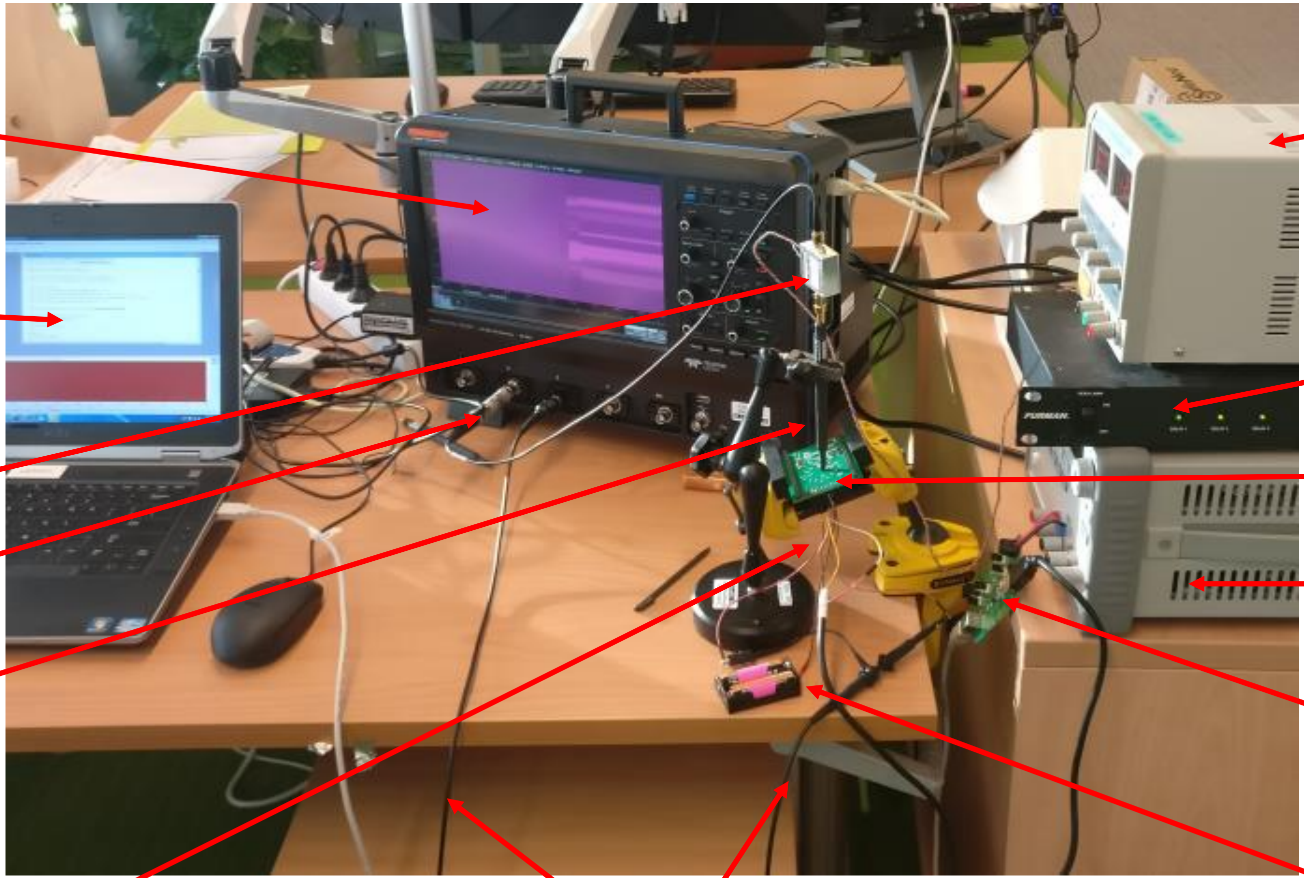
Probe

Resistor 20-80 ohm

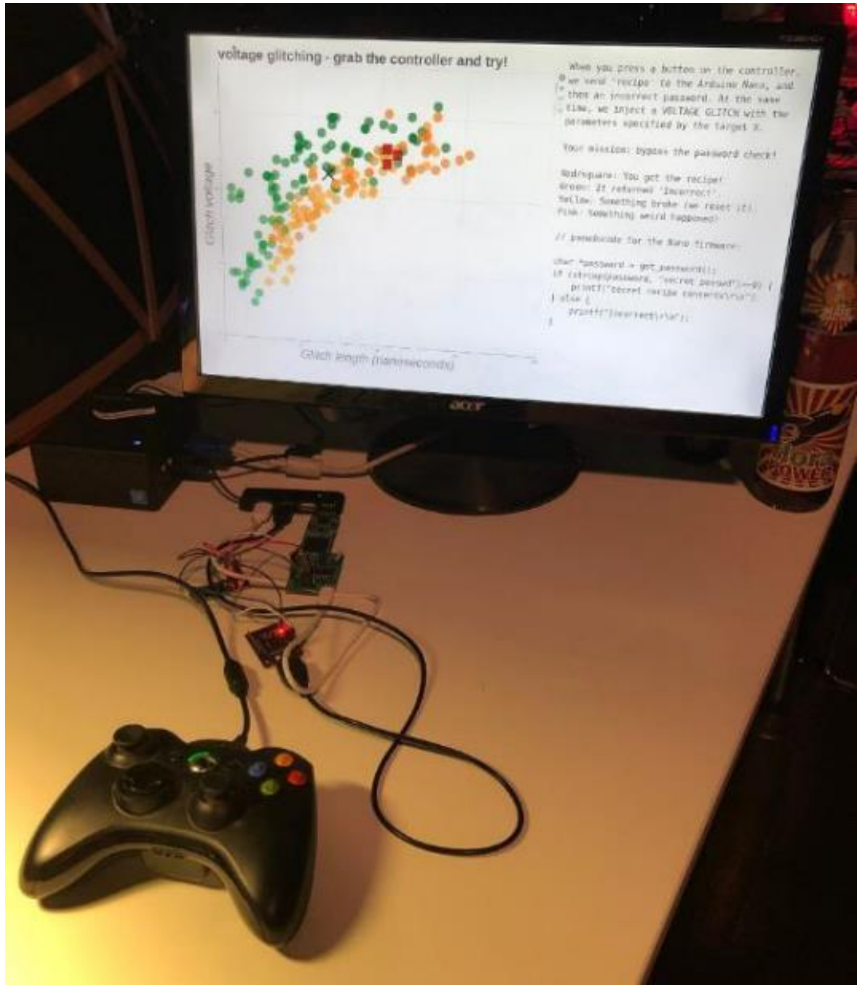
More advanced setup for power analysis



Even more advanced setup for EM analysis



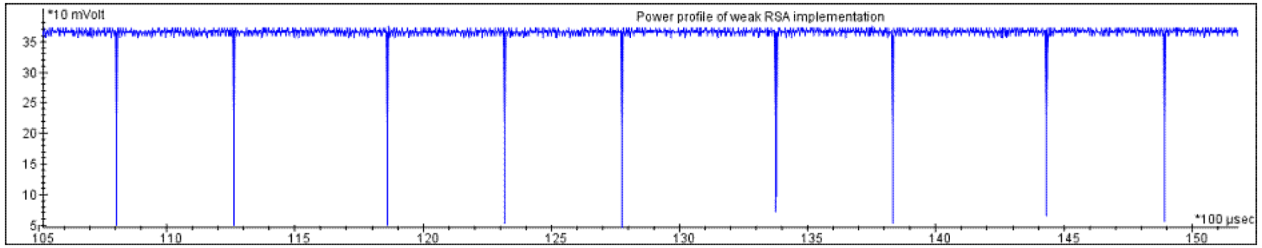
Simple (Cheap) Power Fault Injection setup



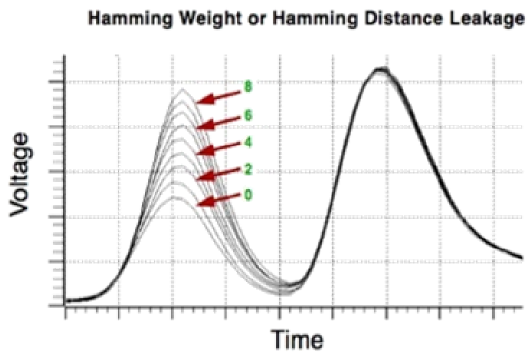
https://github.com/noopwafel/iceglitch

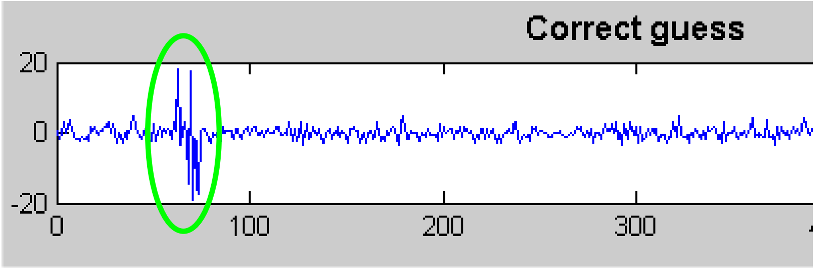
Simple vs. differential power analysis

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



https://www.riscure.com/uploads/2018/11/201708_Riscure_Whitepaper_Side_Channel_Patterns.pdf





Reverse engineering of JavaCard bytecode

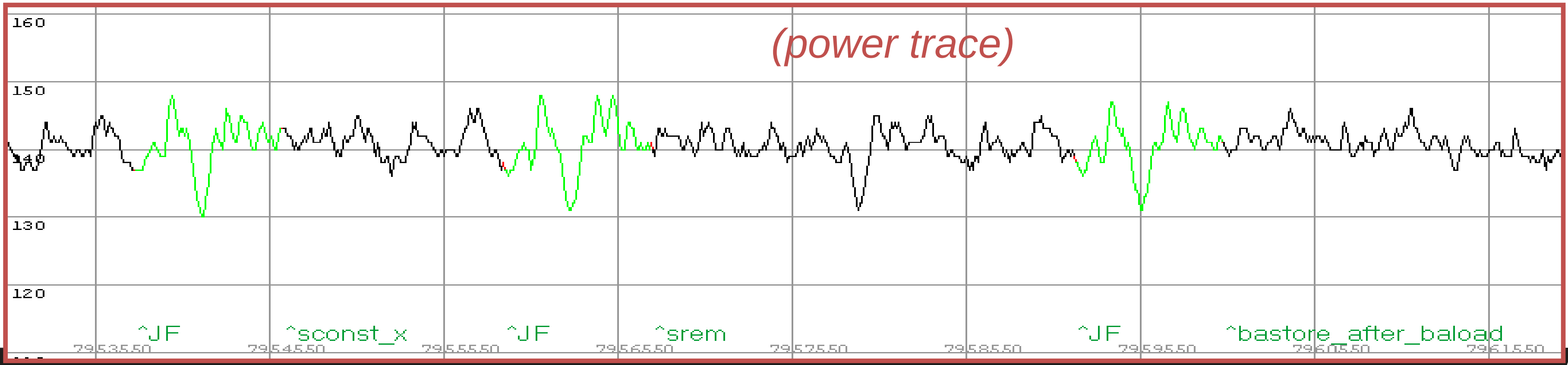
- Goal: obtain code back from smart card
  - JavaCard defines around 140 bytecode instructions
  - JVM fetch instruction and execute it

(source code)
m_ram1[0] = (byte) (m_ram1[0] % 1);

compiler →

(bytecode)
getfield_a_this 0;
sconst_0;
baload;
sconst_1;
srem;
bastore;

oscilloscope →



Conditional jumps

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

(source code)

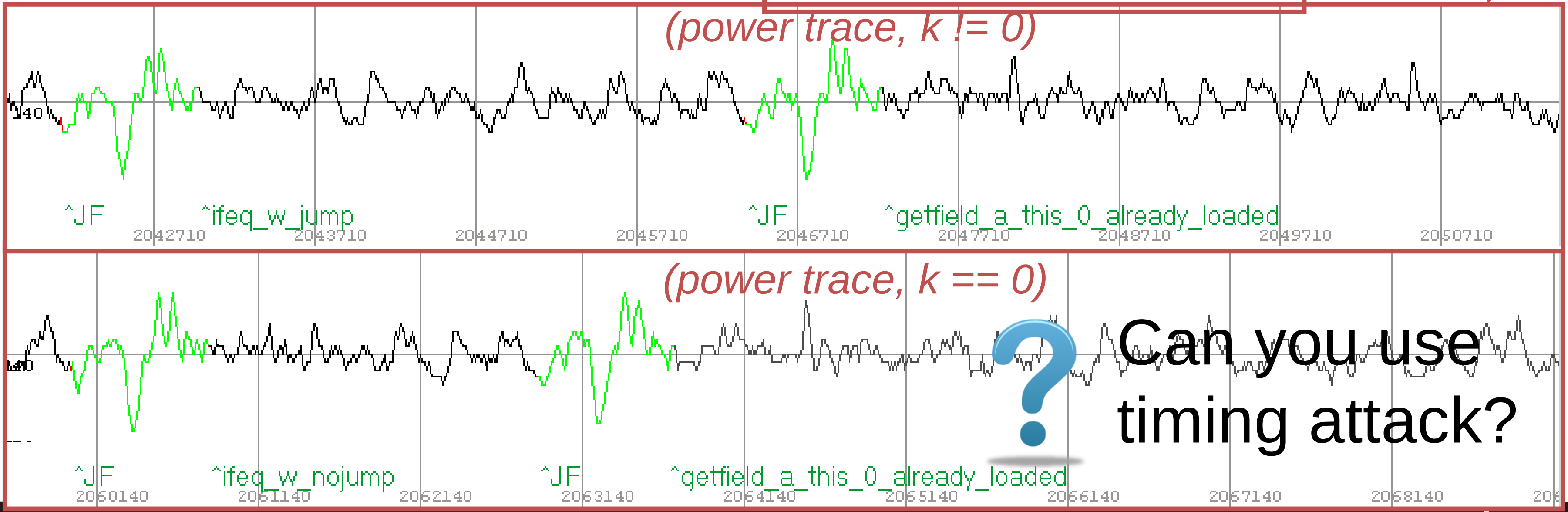
```
if (key == 0) m_ram1[0] = 1;
else m_ram1[0] = 0;
```

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: …
```

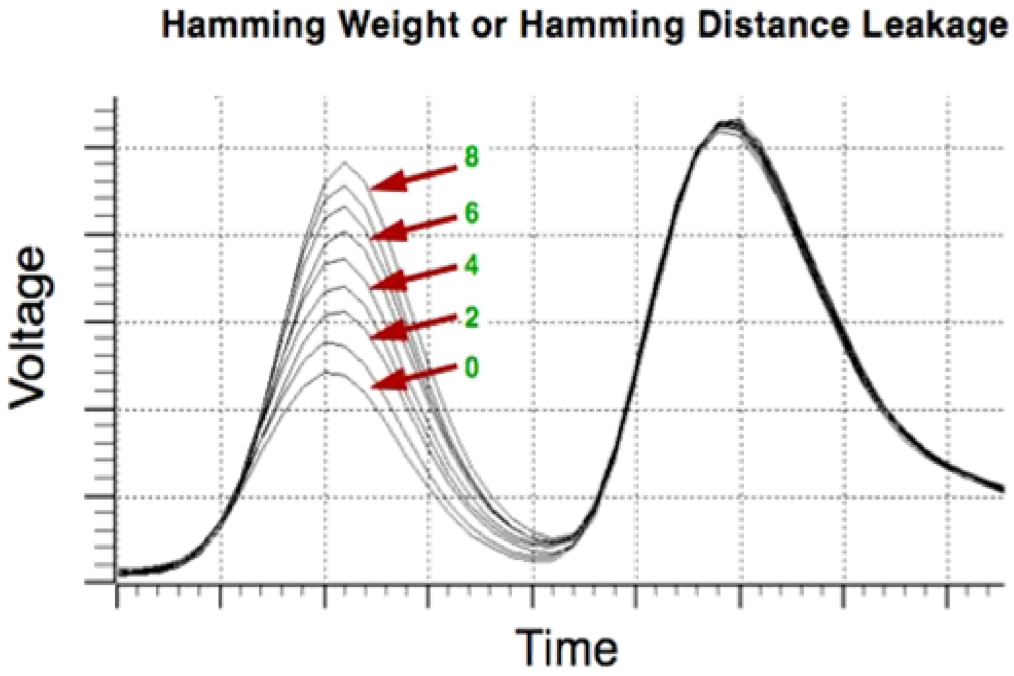
oscilloscope



Can you use timing attack?

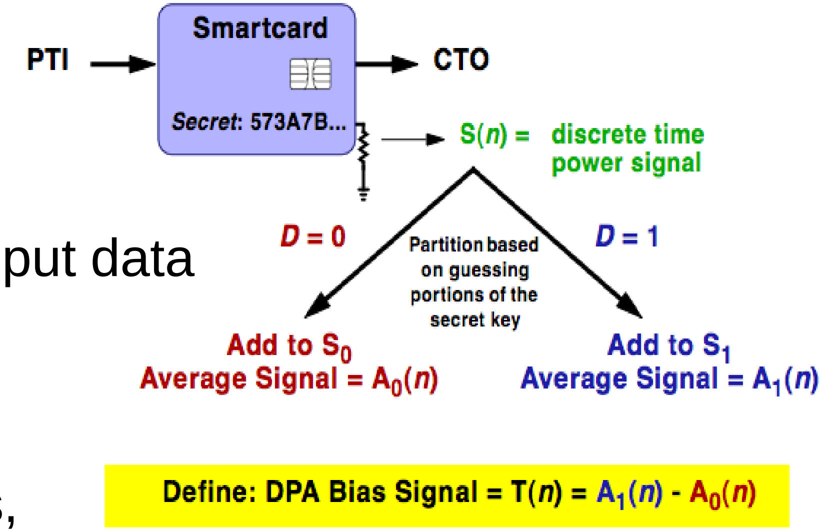
Simple power analysis – data leakage

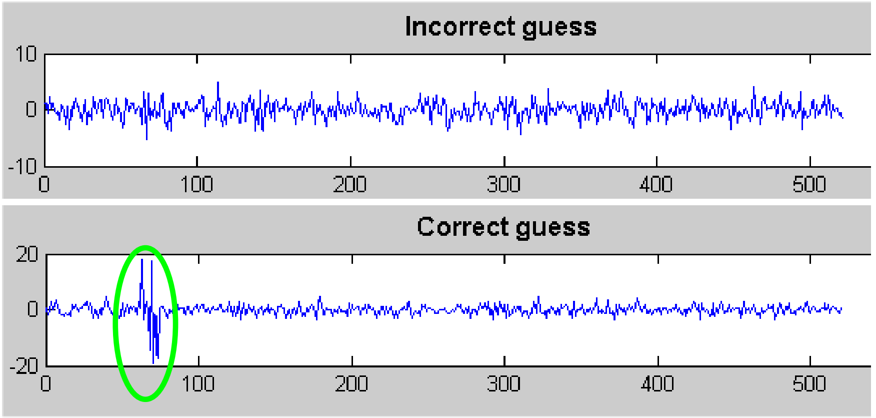
- Data revealed directly when processed
  - e.g., Hamming weight of instruction argument
    - hamming weight of separate bytes of key ($2^{56} \rightarrow 2^{38}$), how severe it is?



Differential power analysis (DPA)

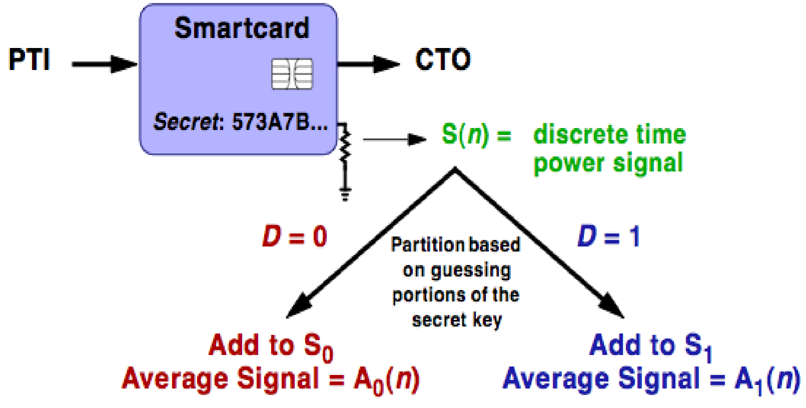
- DPA attack recovers secret key (e.g., AES)
- Requires large number of power traces (10^2-10^6)
  - Every trace measured on AES key invocation with different input data
- Key recovered iteratively
  - One recovered byte at the time Sbox($KEY_i \oplus INPUT\_DATA_i$)
  - Guess possible key byte value (0-255), group measurements, compute average, determine match



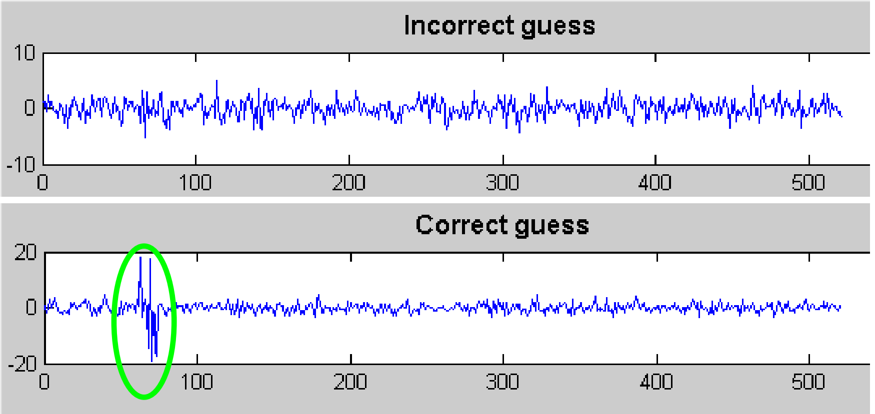


Differential power analysis

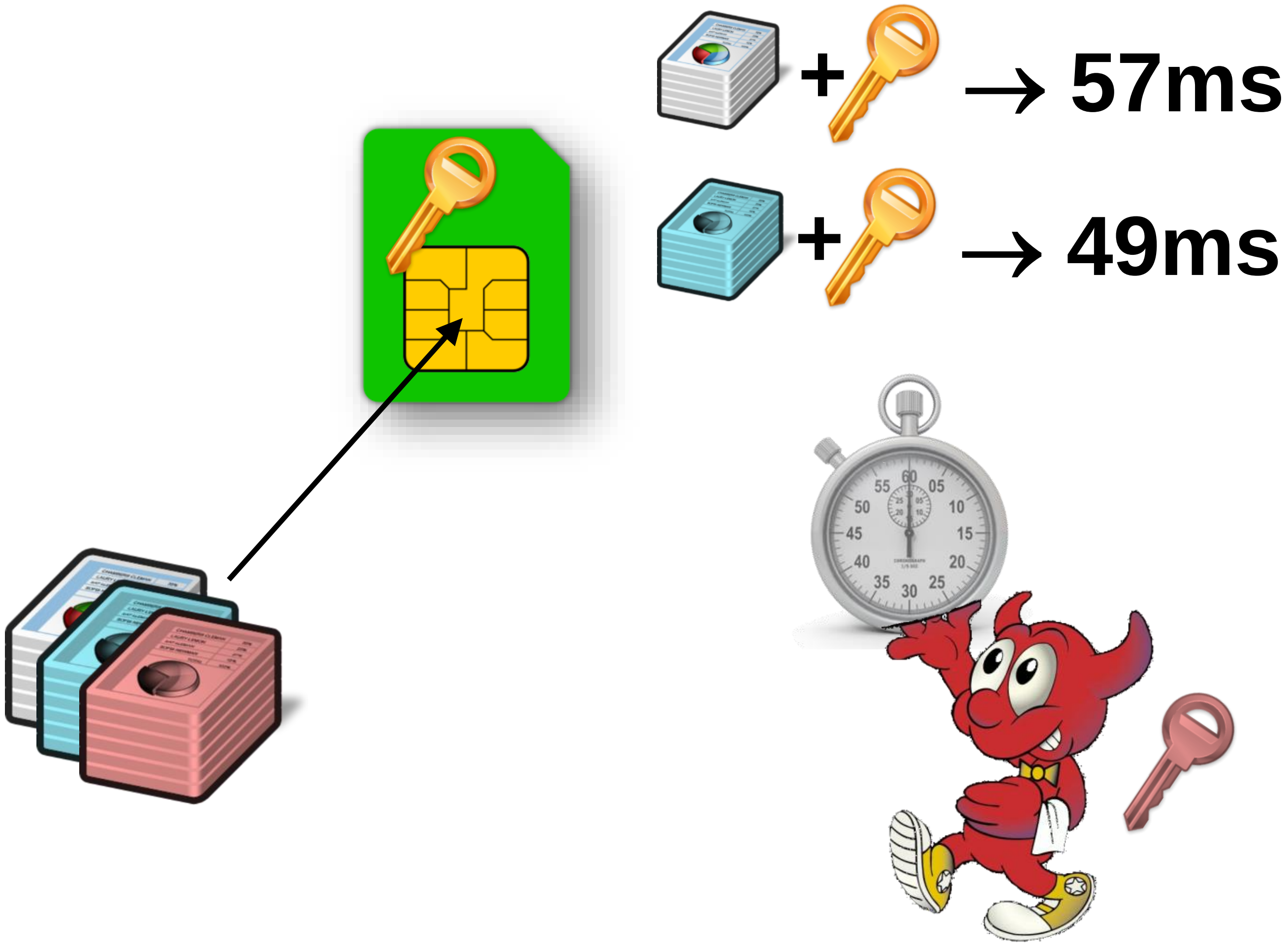
- Very Powerful attack on secret values (keys)
  - E.g., Sbox(KEY ⊕ INPUT_DATA)
1. Obtain multiple power traces with (fixed) key usage and variable data
   - 10^3-10^6 traces with known I/O data => S(n)
   - Sbox(KEY ⊕ KNOWN_DATA)
2. Guess key byte-per-byte
   - All possible values of single byte tried (256)
   - D = HammWeight(Sbox(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 and 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



Define: DPA Bias Signal = T(n) = $A_1(n)$ - $A_0(n)$

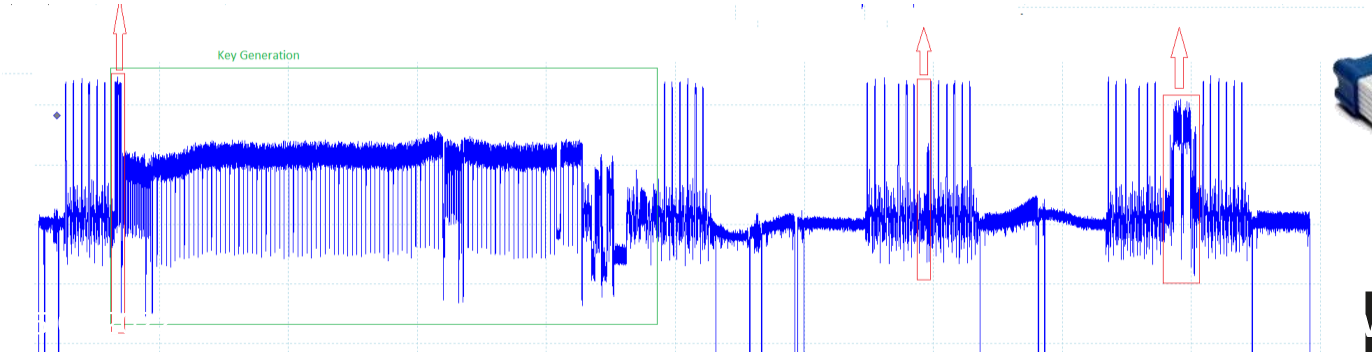


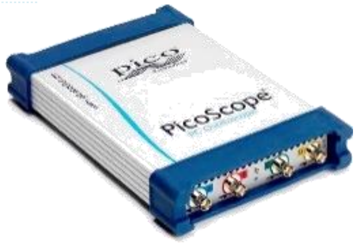
Timing attack: principle



Timing attacks

- Execution of crypto algorithm takes different time to process input data with some dependence on secret value (secret/private key, secret operations…)
  1. Due to performance optimizations (developer, compiler)
  2. Due to conditional statements (branching)
  3. Due to cache misses or other microarchitectural effects
  4. Due to operations taking different number of CPU 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 (modexp) (RSA/DH…)

- $M = C^d \mod N$

Is there any dependency of time on secret value?

- $M = \underbrace{C * C * C * \ldots * C}_{\text{d-times}} \mod N$

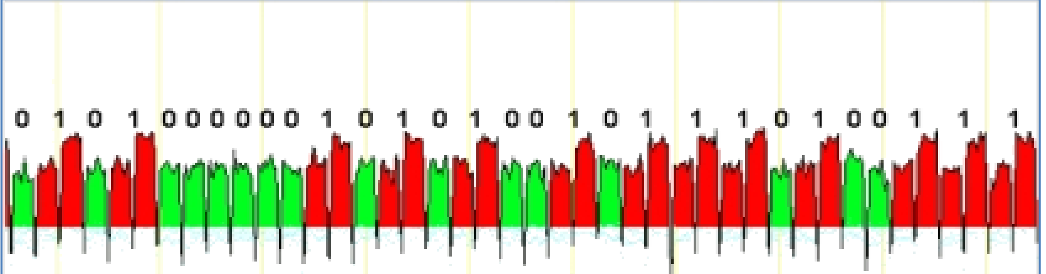
- Easy, but extremely slow for large d (e.g., >1000s bits for RSA)
  – Faster algorithms exist

Faster modexp: 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 (dj == 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

Faster and more secure modexp: Montgomery ladder

- Computes x^d mod N
- Create binary expansion of d as $d = (d_{k-1}...d_0)$ with $d_{k-1}=1$

```
x0=x; x1=x^2
for j=k-2 to 0 {
  if dj=0
    x1=x0*x1; x0=x0^2
  else
    x0=x0*x1; x1=x1^2
  x1=x1 mod N
  x0=x0 mod N
}
return x1
```

Both branches with the same number and type of operations (unlike square and multiply on previous slide)

- Be aware: timing leakage still possible via cache side channel, non-constant time CPU instructions, variable k-1…

Faster and more secure modexp: Montgomery ladder

- Computes x^d mod N
- Create binary expansion of d as $d = (d_{k-1}...d_0)$ with $d_{k-1}=1$

```
x0=x; x1=x^2
for j=k-2 to 0 {
  b=dj
  x(1-b)=x0*x1; xb=xb^2
  x1=x1 mod N
  x0=x0 mod N
}
return x1
```

Memory access often is not contact time!
Especially in the presence of caches.

- Is it constant time?
  - Solution: conditional swap or conditional move, arithmetic-based procedures

Faster and more secure modexp: Montgomery ladder

- Computes x^d mod N
- Create binary expansion of d as d = $(d_{k-1}...d_0)$ with d_{k-1}=1

```
x0=x; x1=x2; sw = 0
for j=k-2 to 0 {
  b=dj
  cswap(x0,x1,b⊕sw)
  sw = sw⊕di
  x1=x1*x2; x2=xc2
  x2=x2 mod N
  x1=x1 mod N
}
return x1
```

Depends on the cswap…
but it can be ☺

- Is it constant time?

Cswap based on arithmetic of field operands

```
void fe25519_cswap(fe25519* in1, fe25519* in2, int condition)
{
    int32 mask = condition;
    uint32 ctr;
    mask = -mask;
    for (ctr = 0; ctr < 8; ctr++)
    {
        uint32 val1 = in1->as_uint32[ctr];
        uint32 val2 = in2->as_uint32[ctr];
        uint32 temp = val1;
        val1 ^= mask & (val2 ^ val1);
        val2 ^= mask & (val2 ^ temp);
        in1->as_uint32[ctr] = val1;
        in2->as_uint32[ctr] = val2;
    }
}
```

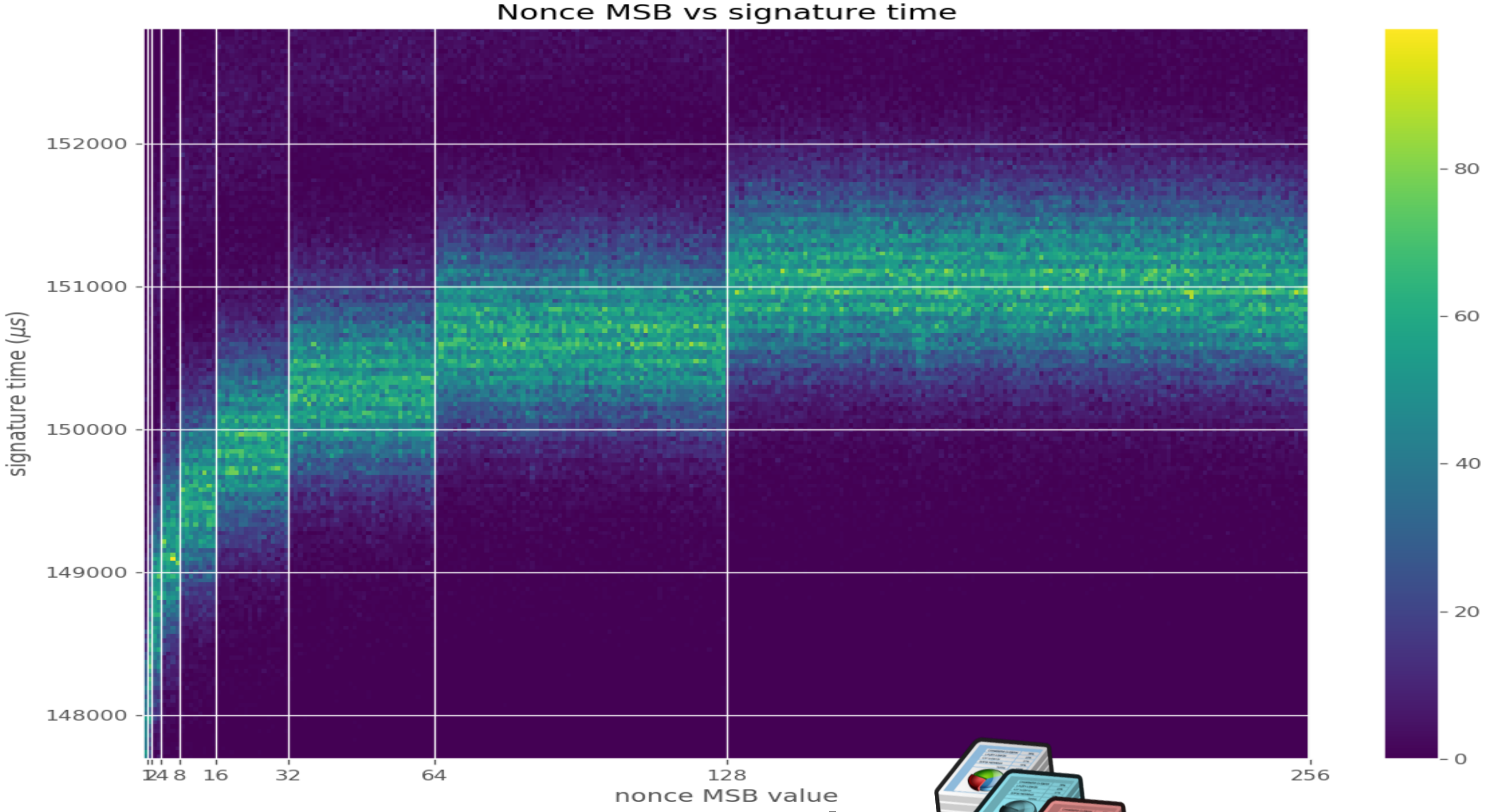
More advanced attacks
(template, deep learning, and clustering attacks)

```
void fe25519_cswap(fe25519* in1, fe25519* in2, int condition)
{
    int32 mask = condition;
    uint32 ctr;
    mask = -mask;
    for (ctr = 0; ctr < 8; ctr++)
    {
        uint32 val1 = in1->as_uint32[ctr];
        uint32 val2 = in2->as_uint32[ctr];
        uint32 temp = val1;
        val1 ^= mask & (val2 ^ val1);
        val2 ^= mask & (val2 ^ temp);
        in1->as_uint32[ctr] = val1;
        in2->as_uint32[ctr] = val2;
    }
}
```

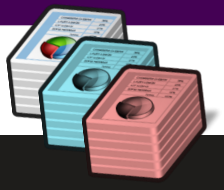
Gather data → Analyse → Bias found → Impact

Run ECC operations → MSB/time → Bias found in ECDSA → CVE-2019-15809

Signature time (µs)

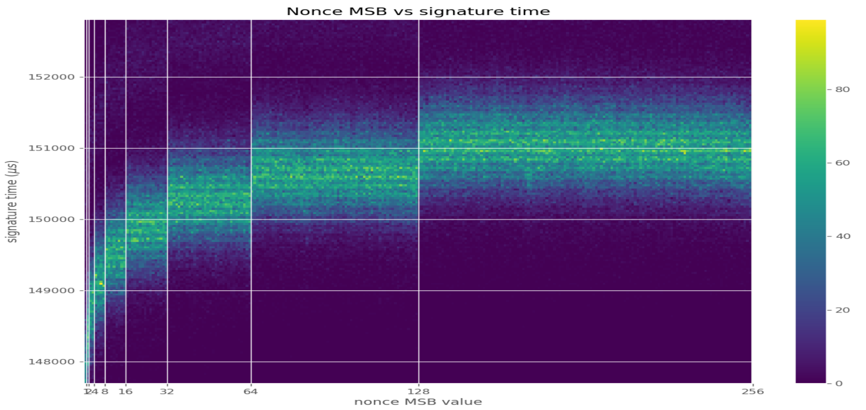


Nonce MSB value



Minerva vulnerability CVE-2019-15809 (10/2019)

- Discovered by ECTester (https://github.com/crocs-muni/ECTester)
- Athena IDProtect smartcard (CC EAL 4+)
  - FIPS140-2 #1711, ANSSI-CC-2012/23
  - Inside Secure AT90SC28872 Microcontroller
  - (possibly also SafeNet eToken 4300…)
- Libgcrypt, wolfSSL, MatrixSSL, Crypto++
- SunEC/OpenJDK/Oracle JDK
- Small time difference leaking few top bits of nonce
- Enough to extract whole ECC private key in 20-30 min
  - ~thousands of signatures + lattice-based attack



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 mod N
- Blinding of ciphertext *c* before decryption
  1. Generate random value *r* and compute r^e mod N
  2. Compute blinded ciphertext $b = c * r^e \mod N$
  3. Decrypt *b* and then divide result by *r*
     - *r* is removed and only decrypted plaintext remains

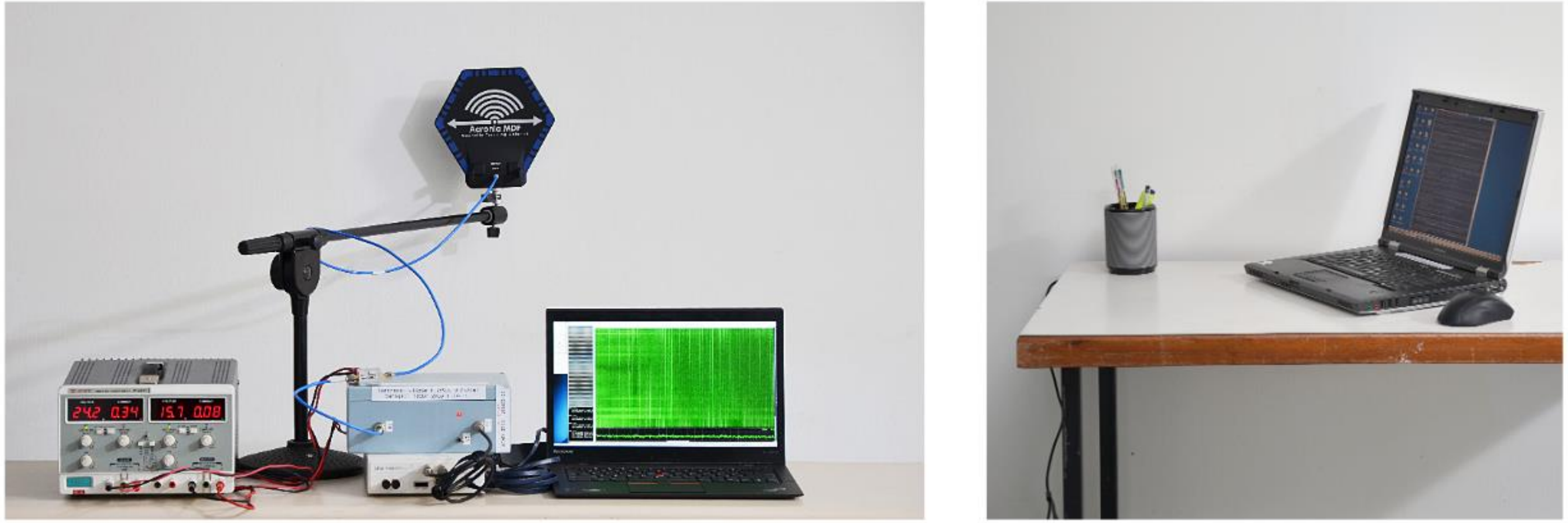
$$(r^e \cdot c)^d \cdot r^{-1} \mod n = r^{ed} \cdot r^{-1} \cdot c^d \mod n = r \cdot r^{-1} \cdot c^d \mod n = m.$$

Is RSA_blinding_on sufficient?

- No, more advanced attacks are possible
  - Cross-correlation attack on OpenSSL,
    - https://www.youtube.com/watch?v=Ah98QIPT8Y4&ab_channel=SHA2017
- What about adding RSA blinding: $c = m^{d+r*\varphi(n)}\ mod\ n$?
- That is not sufficient either, more advanced attacks:
  - Template Attacks,
  - Deep Learning, and
  - Clustering attacks.
- For every countermeasure there is / will be an attack and vice versa…

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)

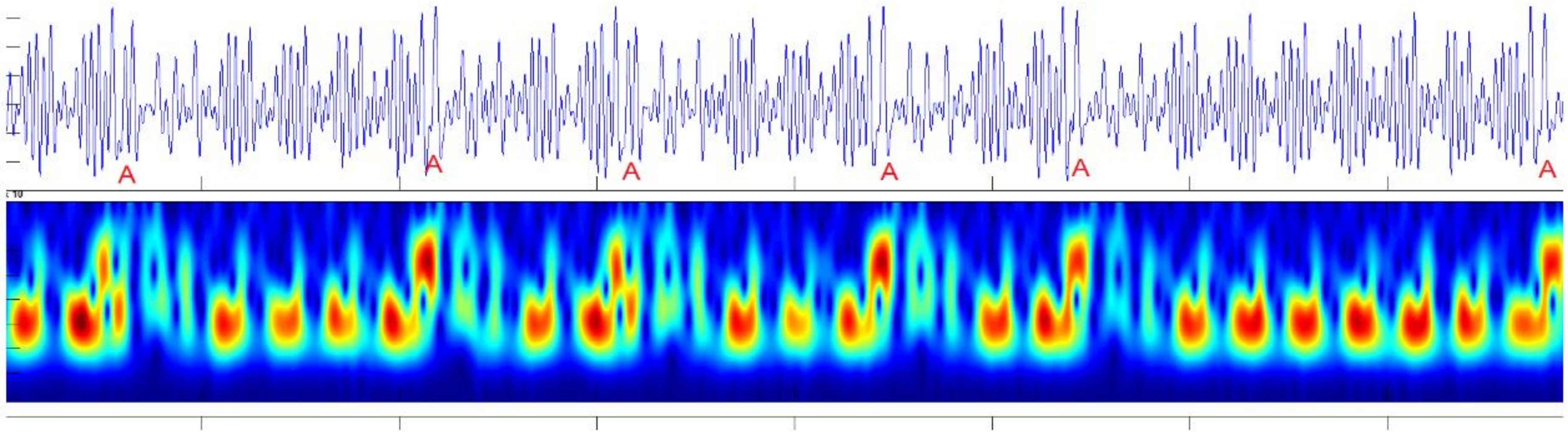


(a) Attacker's setup for capturing EM emanations. Left to right: power supply, antenna on a stand, amplifiers, software defined radio (white box), analysis computer.

(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

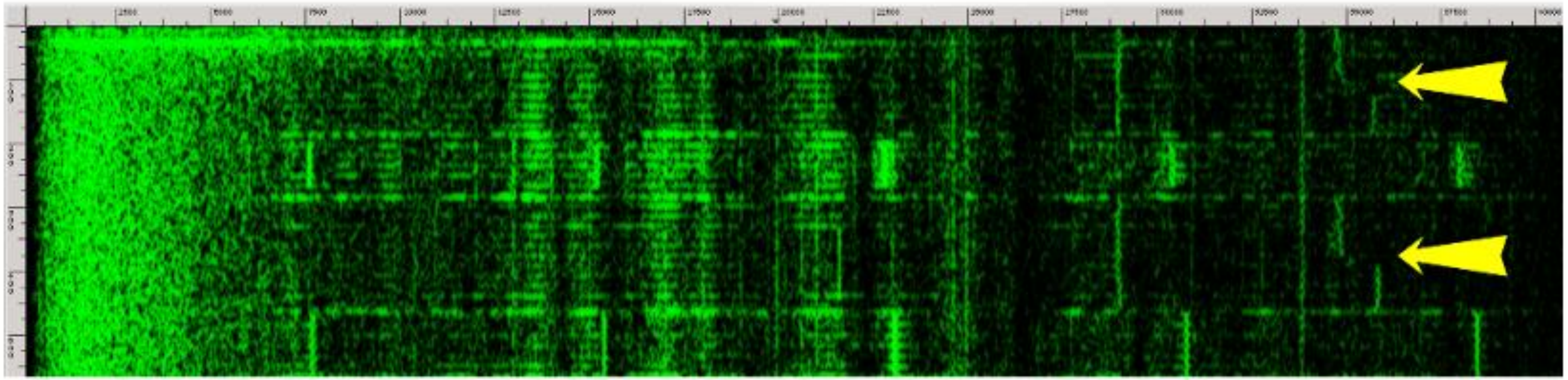


Example: How to evaluate attack severity?

- What was the cost?
  - Not particularly high: $3000
- What was the targeted implementation?
  - Widely used implementation: latest GnuPG's Libgcrypt
- What were preconditions?
  - Local 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
  - Acoustic signal picked by mobile phone microphone up to 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 + 2^{27} x 400B random packets + one minute brute-force search
- Very difficult to write high-speed but constant-time AES
  - Problem: table lookups are not constant-time
  - Not recognized / required by NIST during AES competition

- Cache-time attacks now more relevant due to processes co-location (cloud)

Other types of side-channel attacks

- Acoustic emanation
  - Keyboard clicks, capacitor noise
  - Speech eavesdropping based on high-speed camera
- Cache-occupation side-channel
  - Cache miss has impact on duration of operation
  - Other process can measure own cache hits/misses if cache is shared
  - https://github.com/defuse/flush-reload-attacks
  - http://software.imdea.org/projects/cacheaudit/
- Branch prediction side-channel (Meltdown, Spectre)
  - (2 lectures later in semester)

MITIGATIONS

Generic protection techniques

1. Do not leak
   – Constant-time crypto, bitslicing…
2. Shielding - preventing leakage outside
   – Acoustic shielding, noisy environment
3. Creating additional “noise”
   – Parallel software load, noisy power consumption circuits
4. Compensating for leakage
   – Perform inverse computation/storage
5. Prevent leaking exploitability
   – Ciphertext and key blinding, key regeneration, masking of the operations

Example: NaCl (“salt”) library

libsodium

- Relatively new cryptographic library (2012)
  – Designed for usable security and side-channel resistance (mostly time!)
  – D. Bernstein, T. Lange, P. Schwabe
  – https://cr.yp.to/highspeed/coolnacl-20120725.pdf
  – Actively developed fork is libsodium https://github.com/jedisct1/libsodium
- Designed for usable security (hard to misuse)
  – Fixed selection of good algorithms (AE: Poly1305, Sign: EC Curve25519)
  – `C = crypto_box(m,n,pk,sk), m = crypto_box_open(c,n,pk,sk)`
- Implemented to have constant-time execution
  – No data flow from secrets to load addresses
  – No data flow from secrets to branch conditions
  – No padding oracles (recall CBC padding oracle in PA193)
  – Centralizing randomness and avoiding unnecessary randomness

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; is it enough?
   - Same input data and key; group A
   - Same key and different data; group B
   - Different keys and same data…
3. Compare groups of measured data
   - Is difference visible? => potential leakage
   - Is distribution uniform? Is distribution normal?
   - More advanced methods, for example: Test Vector Leakage Assessment:
     - https://docplayer.net/45501976-Test-vector-leakage-assessment-tvla-methodology-in-practice.html
4. Try to measure again with better precision ☺

Active Side-Channel

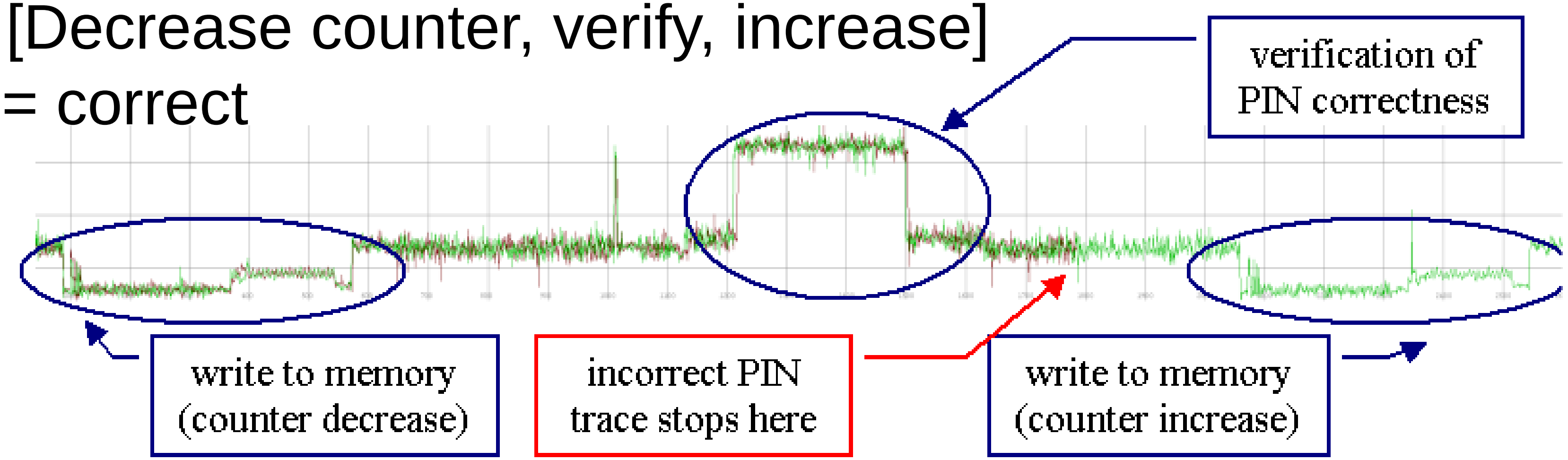
FAULT INJECTION ATTACKS

Semi-invasive attacks

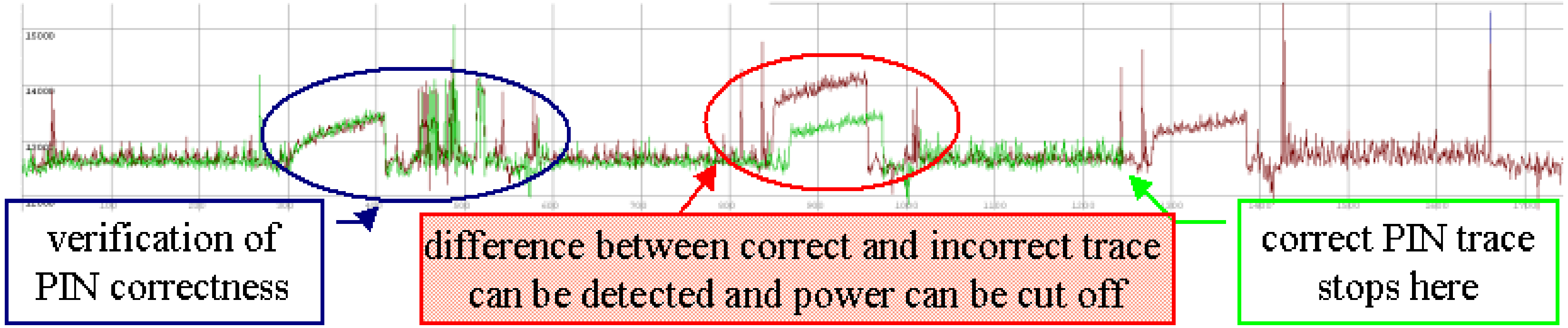
- "Physical" manipulation (but card still working)
- Micro probes placed on the bus
  - After removing epoxy layer
- Fault induction
  - liquid nitrogen, power glitches, light flashes…
  - modify memory (RAM, EEPROM), e.g., PIN counter
  - modify instruction, e.g., conditional jump

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;

01010000

10101010

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

- Robust protection, but cumbersome for developer

FI Example: the “unlooper” device

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
     • Dangerous optimizations (Roca: Infineon primes)
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
   – Try to prevent large number of queries

Mandatory reading

- Constant-time crypto: https://bearssl.org/constanttime.html
- Focus on:
  – What can cause cryptographic implementation to be non-constant?
  – Is there any impact by compiler?
  – How is bitslicing technique improving situation?
  – What particular techniques are used by BearSSL?

Optional reading

- Why Trust is Bad for Security, D. Gollman, 2006
  - http://www.sciencedirect.com/science/journal/15710661/157/3
- Focus on:
  - Which definition of Trust Gollman uses?
  - Why Gollman claims that Trust is bad for security?

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