PV204 Security technologies

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

Petr Švenda *svenda@fi.muni.cz* *@rngsec*
Centre for Research on Cryptography and Security, Masaryk University

CRoCS 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

Power analysis

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. …

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

We need more precise specification of Trust

UNTRUSTED
VS.
TRUSTED
VS.
TRUSTWORTHY

Untrusted system

- System itself 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
  - E.g., End-to-end encryption in instant messaging

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

Trustworthy system (computer)

- *“Computer system where software, hardware, and procedures are secure, available and functional and adhere to security practices”* (Black's Law Dict.)
- 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

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

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, the loss is bounded”
- Example: medical database
  - central governmental DB vs. doctor’s local DB

Good design practice is to allow for risk management

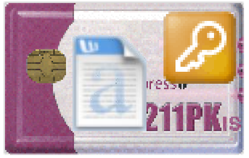
TRUSTED ELEMENT MODES OF USAGE

Trusted (hardware) element - modes of usage

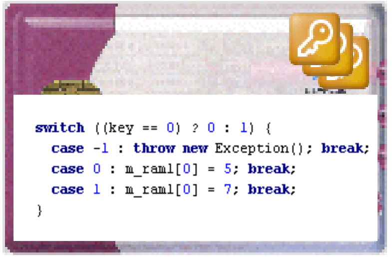
1. Element carries fixed information

2. Element as a secure carrier

3. Element as encryption/signing device



4. Element as programmable device

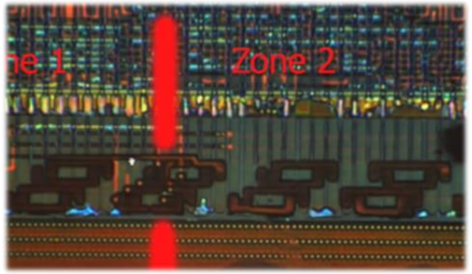


5. Element as root of trust (TPM)

Is secure hardware trusted element a silver bullet?

1. Trusted element shall be small (TCB) => Not whole system => How to extend desirable security properties from TE to whole system?
2. The trusted element itself can still be directly attacked

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://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 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
   - 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…

How to reason about attack and countermeasures?

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

Important: Consider Break Once, Run Everywhere (BORE)

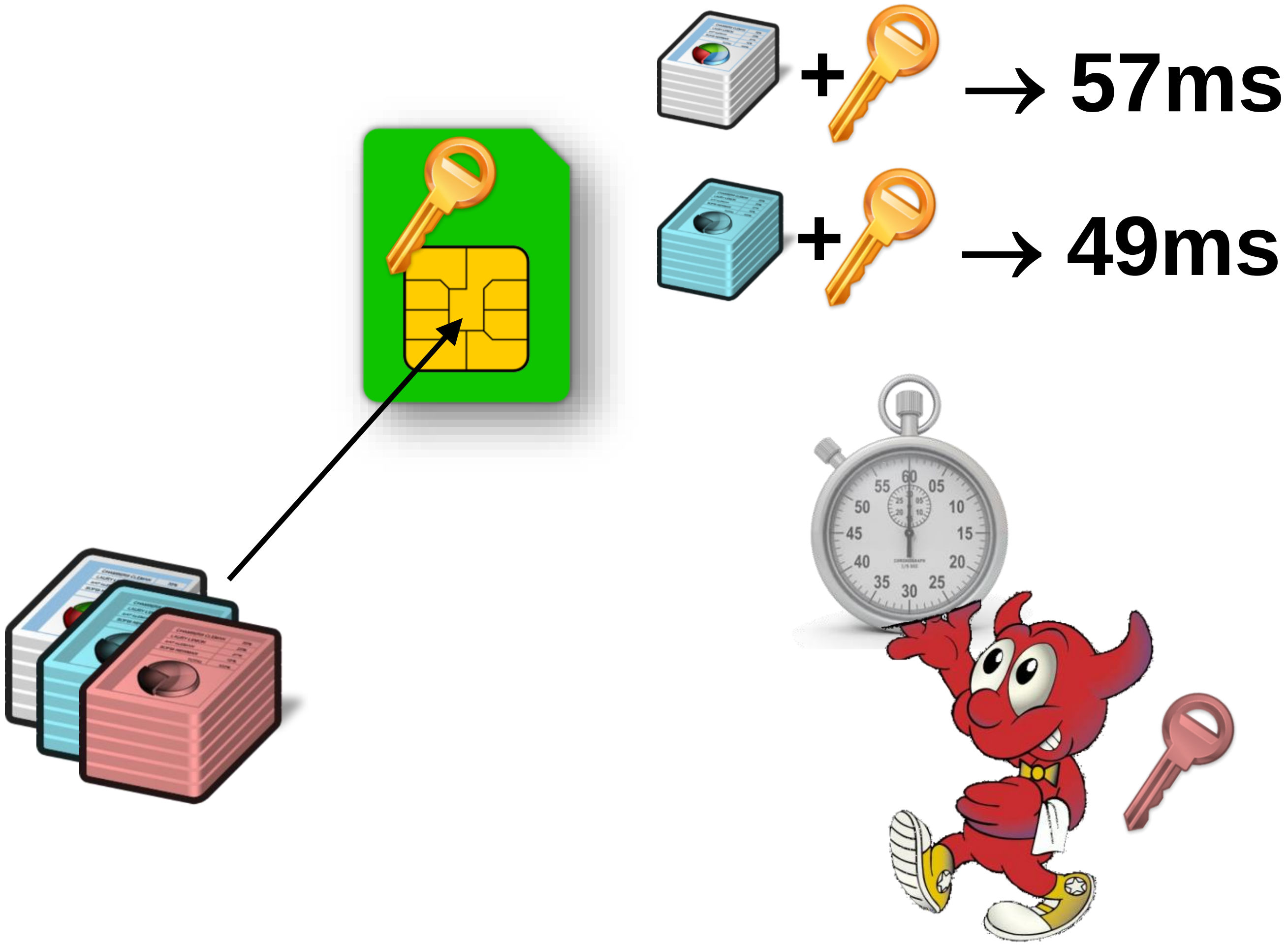
Where are the frequent problems with crypto algs nowadays?

- Security mathematical algorithms
  - OK, we have very strong ones (AES, SHA-3, RSA…) (but quantum computers)
- 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 attacks

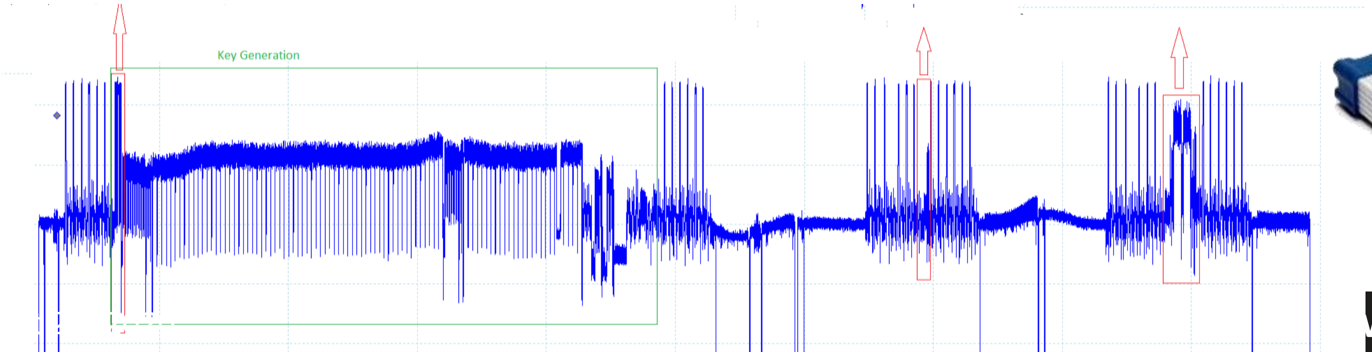
NON-INVASIVE LOGICAL ATTACKS

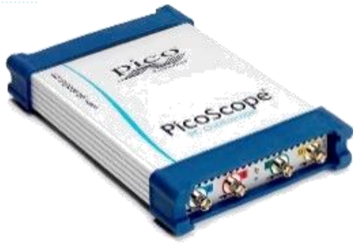
Timing attack: principle



Timing attacks

- Execution of crypto algorithm takes different time to process input data with some dependence on secret value (secret/private key, secret operations…)
  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 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…)

- $M = C^d \bmod N$

Is there any dependency of time on secret value?

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

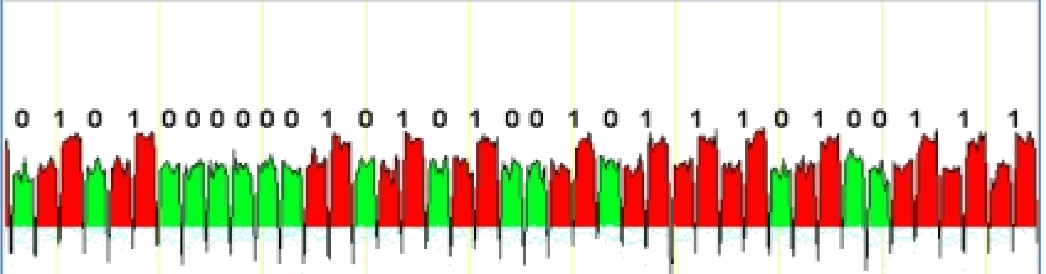
- Easy, but extremely slow for large d (e.g., >1000s bits for RSA)
  – Faster algorithms exist

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 (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

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 $d_{k-1}=1$

```
x1=x; x2=x^2
for j=k-2 to 0 {
  if dj=0
    x2=x1*x2; x1=x1^2
  else
    x1=x1*x2; x2=x2^2
  x2=x2 mod N
  x1=x1 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…

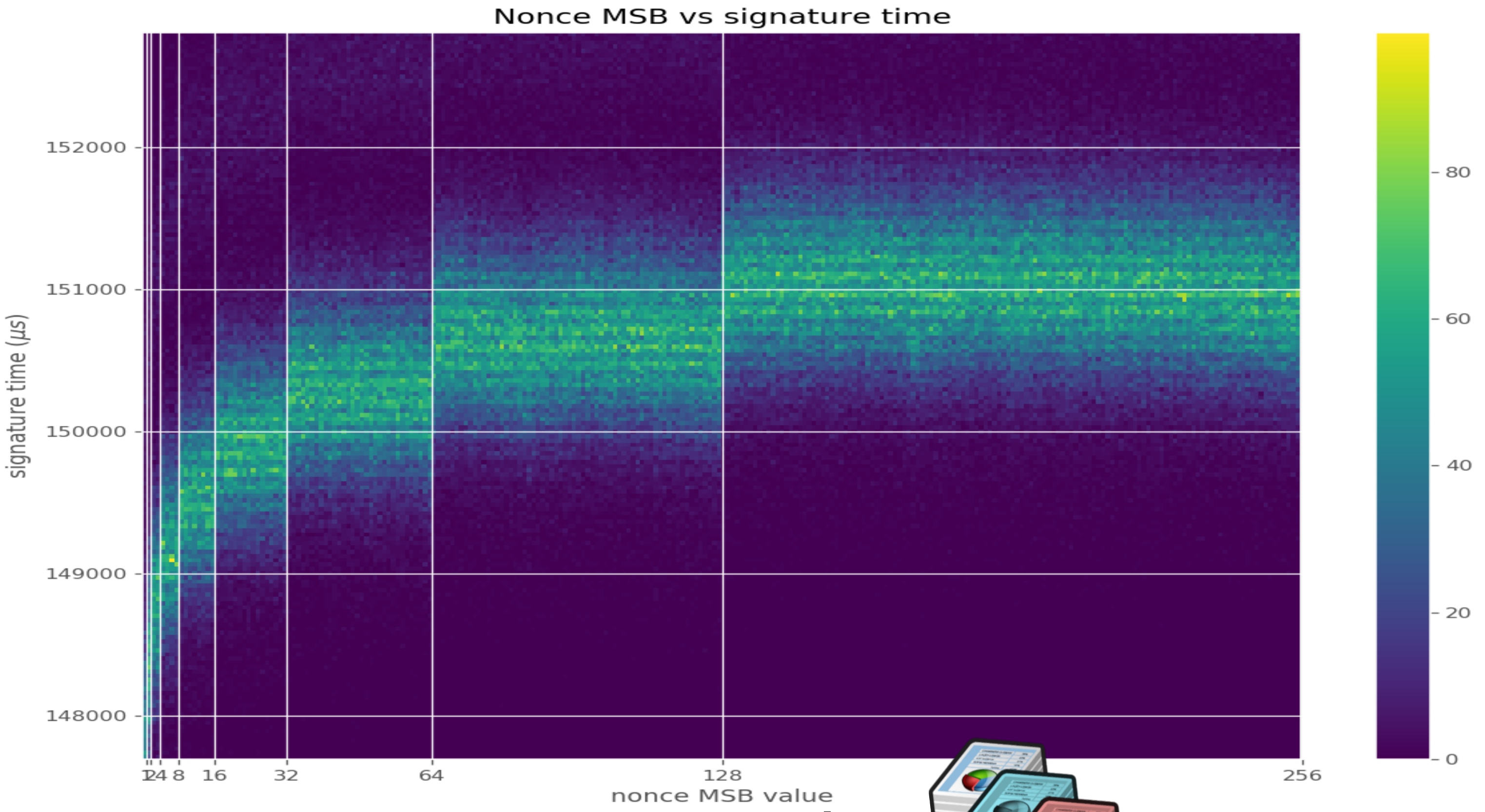
Gather data → Analyse → Bias found → Impact

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

ARM mbed, FlexiProvider, Nettle, GnuPG, cryptlib, OpenSSL, OpenJDK, Microsoft, wolfSSL, LibTom, Botan, Crypto++, PGP, FIPS Validated 140-2

G&D Crypto Java Card, Infineon Crypto Java Card, Gemalto Crypto Java Card, NXP Crypto Java Card, Oberthur Crypto Java Card, Feitian Crypto Java Card

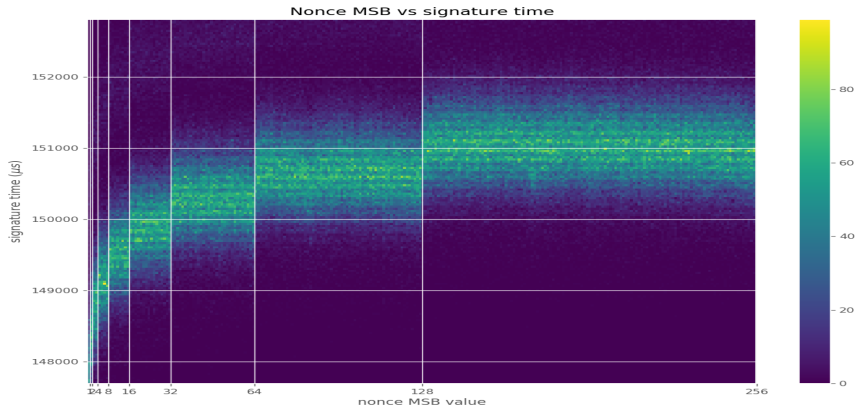
Signature time (μs)



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

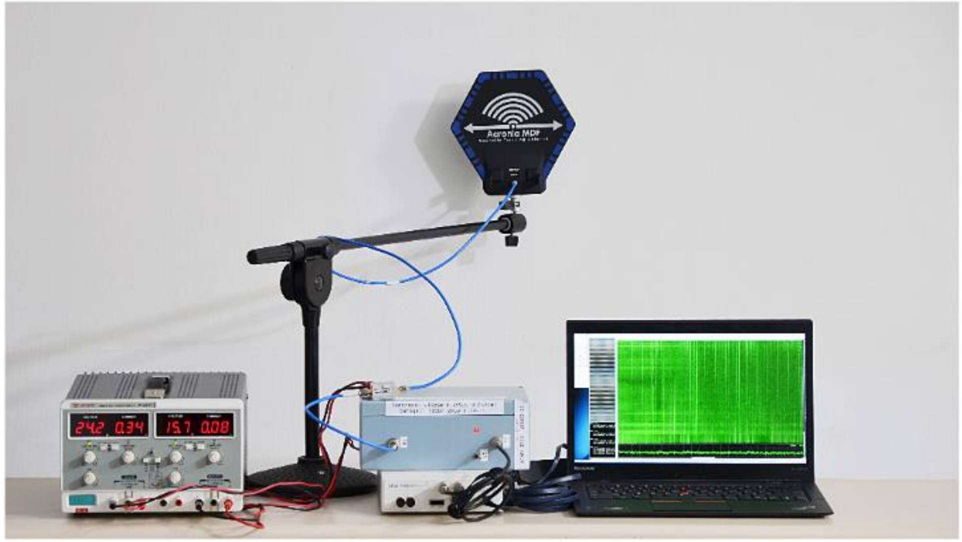
Defense introduced by OpenSSL

- RSA blinding: RSA_blinding_on()
  - https://www.openssl.org/news/secadv_20030317.txt
- Decryption without protection: $M = c^d \bmod N$
- 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

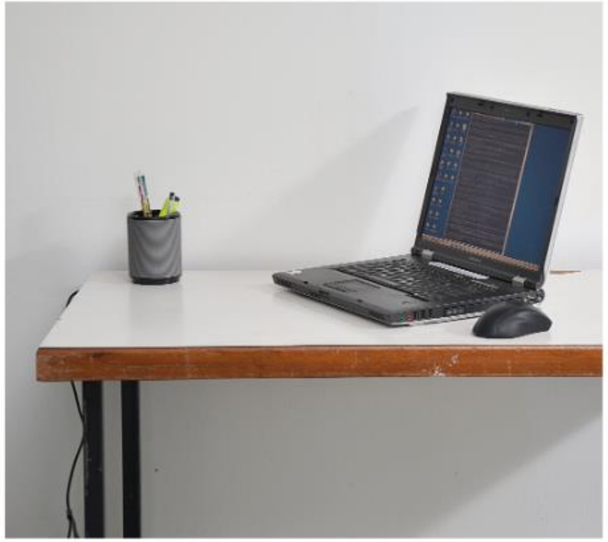
$$(r^e \cdot c)^d \cdot r^{-1} \mod n = r^{ed} \cdot r^{-1} \cdot c^d \mod n = r \cdot r^{-1} \cdot c^d \mod n = m.$$

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)



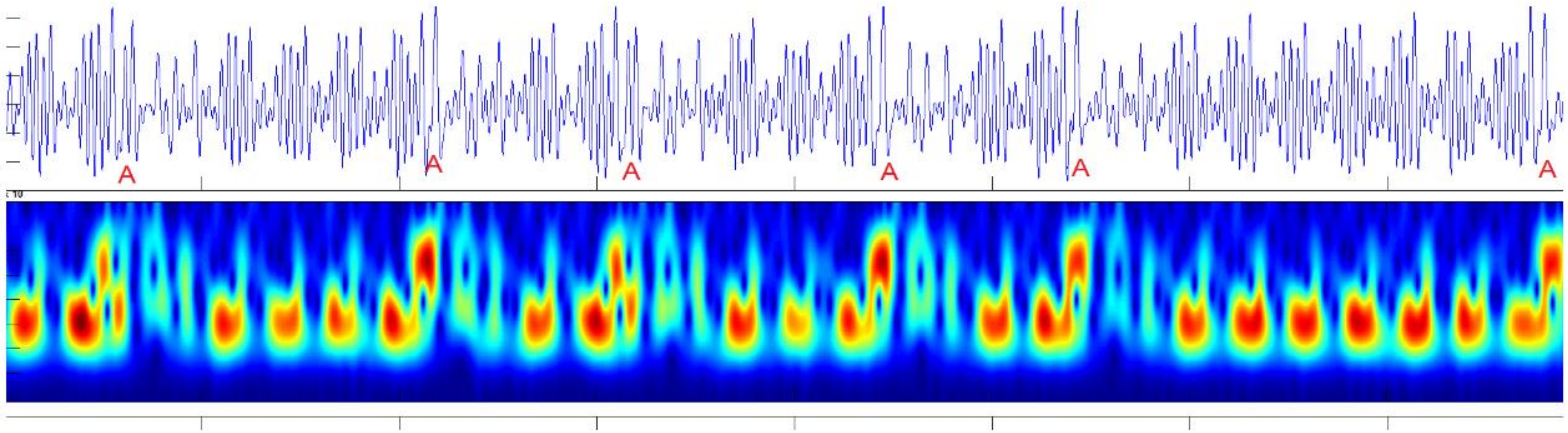
(a) Attacker's setup for capturing EM emanations. Left to right: power supply, antenna on a stand, amplifiers, software defined radio (white box), analysis computer.



(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

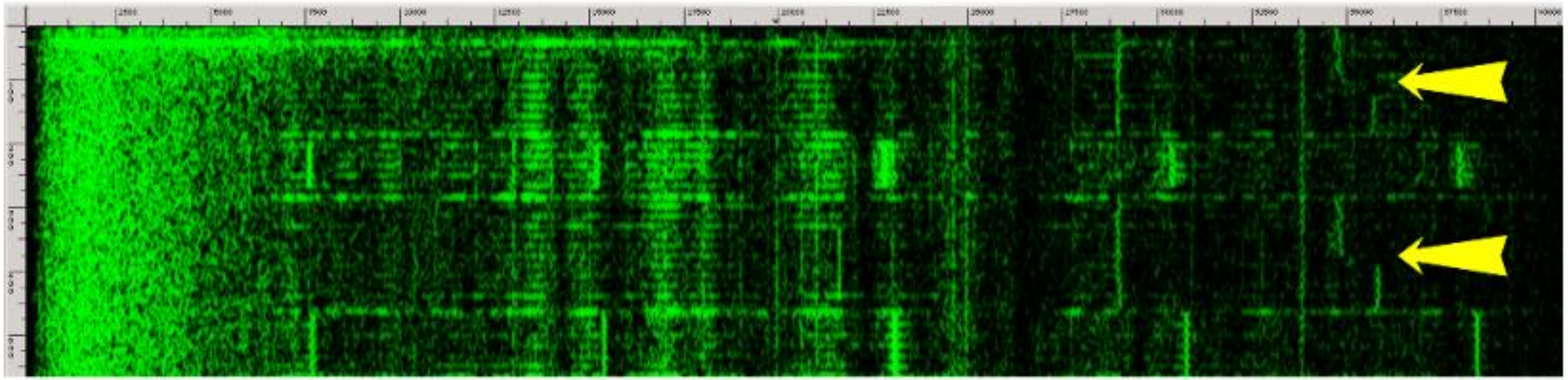


Example: How to evaluate attack severity?

- What was the cost?
  - Not 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 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 / 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 blinding, key regeneration…

Example: NaCl ("salt") library

libsodium

- Relatively new cryptographic library (2012)
  - Designed for usable security and side-channel resistance
  - 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
   - 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 ☺

Activity: Side-channels (10 minutes)

1. Power consumption of memory write instruction depends on the Hamming weight of stored byte
2. Time required to execute **`inc`** instruction (a++) is faster than **`add`** instruction (a+b)
3. Temperature of CPU increases with every instruction executed (and CPU is cooled by fan)

- For every listed side-channel, argue within the group (Google if necessary):
  - Propose an attack(s) based on the particular side-channel
  - What is the cost of required equipment?
  - What are possible options to mitigate the attack?
- Order given side-channels by
  - Seriousness with respect to security impact
  - Difficulty to systematically mitigate the side-channel leakage

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 (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?

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