This work is distributed under the CC BY-SA license, unless stated otherwise.

Hardware Security

11. Physical side-channel attacks 3 (active)

Ján Jančár

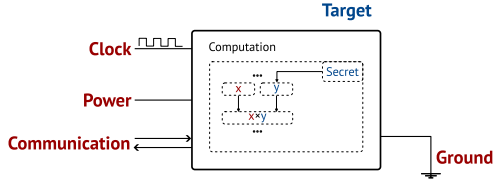
Summer 2026

Today

- Fault injection
  - What
  - How
- Attacks
  - Generic
  - Symmetric cryptography
  - Asymmetric cryptography

Fault injection/induction (FI)

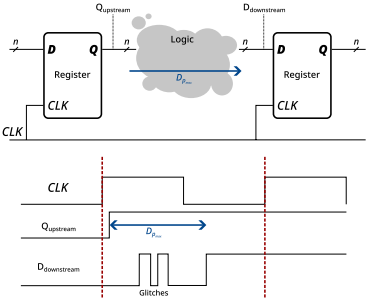
- Introduction of **errors** in the computation
  - Changing the environment the device works in to be **out-of-spec** *somehow*
- **Fault attacks** (FA)
  - Use fault injection to break some security property
  - Confidentiality or integrity
- Many techniques
  - Voltage glitching
  - Clock glitching
  - EM
  - Laser
  - Software (Plundervolt/RowHammer/CLKSCREW)
  - Tearing



Fault injection

Timing (motivation)

- Logic (stateless) surrounded by registers
- Device circuits clocked by some clock signal with **clock period** T_{clk}
- One clock period
  - Outputs of one register pass through logic and into another register
- Logic has some **propagation time** (D_{pmax})
  - How long it takes an input change to fully propagate and output to stabilize
- Logic has **glitches**
  - Output value may oscillate and only holds correct output after the propagation time



- Propagation time needs to be shorter than clock period ($D_{pmax} < T_{clk}$)
- Why?
  - If longer, next register may sample and store wrong result (glitch)
- Many fault injection techniques break this constraint

Fault injection

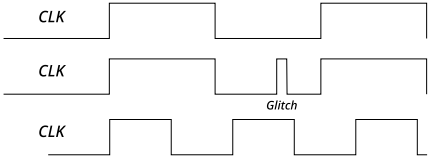
Voltage glitching

- Attacker controls the power supply of the target
- Attacker injects positive or negative **voltage glitch**
  - Short pulse different from the target operating voltage
  - Negative voltage glitch
    - $D_{p_{max}}$ increases temporarily
  - Positive voltage glitch
    - $D_{p_{max}}$ decreases temporarily, then increases (bounce)
- Or, attacker continuously **underpowers** the target
  - $D_{p_{max}}$ increases globally
- Example
  - HHH Chapter 6 – Piezoelectric lighter spark (high voltage pulse)

Fault injection

Clock glitching

- Attacker controls the clock of the target
  - Not always possible!
  - Internal clocks / PLLs
- Attacker introduces a **clock glitch**
  - T_{clk} decreases temporarily
- Attacker **overclocks** the target
  - T_{clk} decreases globally



Fault injection

EM (EMFI)

- Attacker creates a strong EM pulse near a target
  - Often requires decapping to get close enough (square of distance dropoff)
  - Uses an EM injection probe (a coil) with a pulse generator
- EMFI is localized
  - Probe couples to nearby chip circuitry
  - Position matters, precise control needed (XYZ table)
- Mechanism of fault injection is varied
  - Depends on where/when the fault is injected
  - Time constraint break: Localized voltage glitch or clock glitch
  - Sampling fault: Changes logic value being sampled by register
- Example
  - ChipWhisperer ChipShouter
  - ChipWhisperer PicoEMP

Fault injection

Laser

- Attacker hits a decapped target with a short laser pulse
  - Requires decapping
- Photo-electric effect
  - Laser pulse induces some current in a transistor
- Very localized
  - Even more than EMFI
  - Single-bit faults
- Good for targeting and flipping SRAM cells
- Difficult for smaller chip technologies

Fault injection

Software (Plundervolt/RowHammer/CLKSCREW)

- Plundervolt: Software-based Fault Injection Attacks against Intel SGX
  - Injected faults into SGX by underpowering the CPU while the enclave was computing
- RowHammer: Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors
  - Class of attacks on DRAM
  - Accessing a memory row discharges neighboring rows slightly
  - Hammering a row above and below target can induce a fault
- CLKSCREW: Exposing the Perils of Security Oblivious Energy Management
  - Uses software clock frequency and voltage control to inject faults into ARM Trustzone

*Fault injection can happen **from** software.*

Fault injection

Others

- Body Biasing Injection (BBI)
  - Like EMFI, but attacker contacts the backside substrate directly
- FIB
  - Modifies the target chip (also kind of a fault)
- Active microprobe
  - Injects logical values (drive high or pull low) almost anywhere
  - Requires decapping and maybe FIB for access
- X-Ray
  - Focused X-ray beam introduces a semi-permanent fault
  - Very localized, passes through material effortlessly
- Tearing
  - Attacker cuts power supply fully
  - Not *really* a fault injection
  - Why is it interesting for the attacker?
    - Device may expect a full “transaction” to go through, tearing breaks that
    - Example: PIN/password check and then retry counter decrement
    - Attacker may be able to detect correct/wrong PIN (via SCA) and tear before the decrement

Fault models

Describes the kind of fault a given method creates or a given attack needs.

- What does the fault do to the target?
  - Low-level (1 .. n bits)
    - Bit set(s)
    - Bit reset(s)
    - Bit flip(s)
    - Random bit flip(s)
  - High-level
    - Instruction skip
    - Instruction corruption
    - Instruction replay
- Localization
  - Where does the fault occur?
- Probability
  - With what probability does the fault occur?
- Bias
  - How does the distribution of faulted values look like?
  - Is it data-dependent?
- Duration
  - How long does the fault last?

Fault attacks (FA)

- Use fault injection
- Often coupled with side-channel analysis
  - When to fault?
- Target varies
  - Generic
  - Cryptographic
    - More complex, key recovery/verification bypass as a goal
    - Even more algorithm-specific than SCA
    - Symmetric cryptography
    - Asymmetric cryptography
- Techniques vary with it

Fault attacks

Generic fault attack

Example

```
def checkPassword(password):
    correct = "hunter2"
    if len(correct) != len(password):
        return False
    for i in range(len(correct)):
        if password[i] != correct[i]:
            return false
    return True
```

- What would you fault?
  - Depends on the fault model
  - Depends on the goal: Pass the check or get the password
- Instruction skip → skip the loop
- Instruction corruption → modify return to `true`
- Bit set/reset → set loop bound to 0 (or some small value you can guess)
- Bit set/reset → zero out first byte of `correct` (zero length password)

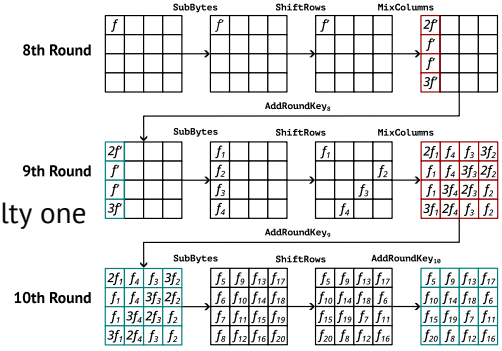
Fault attacks

Symmetric cryptography

- Cipher
  - Has rounds
  - Operates on state
  - Has round keys
  - Fault introduces change in state
    - *state differential* = $correct \oplus faulty$
- Attacks focus on finding ways to reduce keyspace left to brute-force
- Examples
  - DFA
    - Differential Fault Analysis
    - Like differential cryptanalysis
    - Correct ciphertext + faulty ciphertext(s)
  - SFA
    - Statistical Fault Analysis
    - Only faulty ciphertexts
  - SIFA
    - Statistically Ineffective Fault Analysis
    - Only correct ciphertexts

DFA

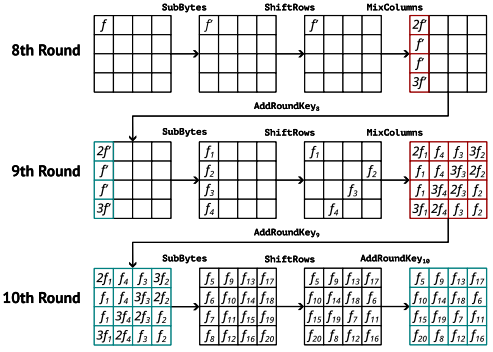
Biham97



- Assumes fixed key and plaintext
- Attacker collects one correct ciphertext a faulty one
- Fault creates a *state differential*
  - Propagates to the end
  - Affects the ciphertext
- **Idea**: Express intermediate value in terms of the ciphertext and round key
  - Correct intermediate = f(round key subpart, correct ciphertext)
  - Faulty intermediate = f(round key subpart, faulty ciphertext)
  - Fault = Correct intermediate ⊕ Faulty intermediate
- Fault spreads and so it allows us to build equations on the key

DFA (detail)

Biham97



- Express t_0 (correct ciphertext)
  - $t_0 = SBOX^{-1}(b_0)$
  - $b_0 = a_0$
  - $a_0 = C_{0,0} \oplus K^{10}_{0,0}$
  - Thus, $t_0 = SBOX^{-1}(C_{0,0} \oplus K^{10}_{0,0})$
- Express t^*_0 (faulty ciphertext)
  - The same, but $t^*_0 = SBOX^{-1}(C^*_{0,0} \oplus K^{10}_{0,0})$
- We know that $t_0 \oplus t^*_0 = 2f_1$, so
  - $2f_1 = SBOX^{-1}(C_{0,0} \oplus K^{10}_{0,0}) \oplus SBOX^{-1}(C^*_{0,0} \oplus K^{10}_{0,0})$
- We can repeat for t_1, t_2, t_3 and obtain:

$$2f_1 = SBOX^{-1}(C_{0,0} \oplus K^{10}_{0,0}) \oplus SBOX^{-1}(C^*_{0,0} \oplus K^{10}_{0,0})$$
$$f_1 = SBOX^{-1}(C_{1,3} \oplus K^{10}_{1,3}) \oplus SBOX^{-1}(C^*_{1,3} \oplus K^{10}_{1,3})$$
$$f_1 = SBOX^{-1}(C_{2,2} \oplus K^{10}_{2,2}) \oplus SBOX^{-1}(C^*_{2,2} \oplus K^{10}_{2,2})$$
$$3f_1 = SBOX^{-1}(C_{3,1} \oplus K^{10}_{3,1}) \oplus SBOX^{-1}(C^*_{3,1} \oplus K^{10}_{3,1})$$

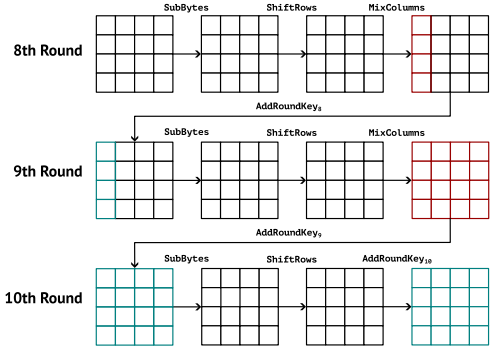
DFA (detail)

$$
\begin{aligned}
2f_1 &= SBOX^{-1}(C_{0,0} \oplus K^{10}_{0,0}) \oplus SBOX^{-1}(C^*_{0,0} \oplus K^{10}_{0,0}) \\
f_1 &= SBOX^{-1}(C_{1,3} \oplus K^{10}_{1,3}) \oplus SBOX^{-1}(C^*_{1,3} \oplus K^{10}_{1,3}) \\
f_1 &= SBOX^{-1}(C_{2,2} \oplus K^{10}_{2,2}) \oplus SBOX^{-1}(C^*_{2,2} \oplus K^{10}_{2,2}) \\
3f_1 &= SBOX^{-1}(C_{3,1} \oplus K^{10}_{3,1}) \oplus SBOX^{-1}(C^*_{3,1} \oplus K^{10}_{3,1})
\end{aligned}
$$

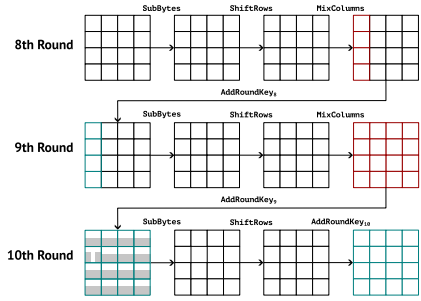
- In general, there will be only 2^8 solutions for the 4 key bytes instead of 2^{32}
- Repeat for other columns, whole key with 2^{32} brute-forcing
- Potentially, more equations involving K^9
- 👍 Only one correct + faulty pair
- 👍 No assumption on fault
- 👎 Needs faulty ciphertext
- 👎 Needs repeated encryption of fixed plaintext

AES-DFA

DFA



- Why fault 8th round?
- Why not first round? Or rounds 1 – 7?
  - The fault gets diffused too soon
  - Equations involve the whole key (not only one byte per equation)
- Why not last round?
  - Does not go through `MixColumns`: 1-byte fault only changes 1-byte of ciphertext → Needs at least 16 faults
- Why not 9th round?
  - Works, but obtains less equations
  - 1-byte fail only spreads to 4 ciphertext bytes



- **Biased fault**
  - Introduces bias into otherwise uniform value
  - Example
    - Bit reset on part of value bits → HW reduced
    - Bit set on part of value bits → HW increased
- **Idea**: Use biased faults (those that bias the value)
- Guess key subpart, uncompute to faulted intermediate value
  - Correct guess: Value is biased
  - Incorrect guess: Value is uniform
- 👍 No control or knowledge of plaintext necessary
- 👍 Only access to faulty ciphertexts
- 👎 Requires many faulty ciphertexts (bias detected via statistical methods)

- **Ineffective fault**
  - Fault happened yet value did not change
  - Thus, result ciphertext does not change (is correct)
  - Example
    - Bit reset fault, but bit was already zero
- **Idea**: These depend on the value
- Assume that target detects wrong ciphertexts and does not give them out
  - Detection/infection countermeasure
- **Idea**: Use the correct ciphertext, knowing that an ineffective fault happened
  - Guess key subpart, uncompute to faulted intermediate value
    - Correct guess: Value is biased
    - Incorrect guess: Value is uniform
  - Bias comes from target filtering effective faults (wrong ciphertexts)
- Site with demo
- No control or knowledge of plaintext necessary
- Uses only ineffective faults (correct ciphertexts)
- Requires many ciphertexts (bias detected via statistical methods)

Fault attacks

Asymmetric cryptography

- Also very algorithm-specific
- Different to symmetric cryptography fault attacks
  - Less *state differential*
  - More algebra
  - More high-level attacks
- Examples
  - RSA-CRT
    - Bellcore attack
  - ECDSA
    - Repeated nonces
    - Nonce difference
  - ECDH
    - Safe-error attack

RSA-CRT

- Modular exponentiation by d is the bottleneck (signing/decryption)
- **Idea**: Split via Chinese Remainder Theorem
  - If you know $x \equiv a_1 \pmod{n_1}$, $x \equiv a_2 \pmod{n_2}$ and so on
  - You can reconstruct $x \bmod n$ uniquely, where $n = n_1 \cdot n_2 \ldots$
- Let's split $\bmod p$ and $\bmod q$
- Precompute
  - $d_p \equiv d \bmod \varphi(p)$
  - $d_q \equiv d \bmod \varphi(q)$
  - $q_{inv} \equiv q^{-1} \bmod p$
- Signature ($s \equiv m^d \bmod n$)
  - $s_1 \equiv m^{d_p} \bmod p$
  - $s_2 \equiv m^{d_q} \bmod q$
  - $h \equiv q_{inv}(s_1 - s_2) \bmod p$
  - $s = s_2 + hq$

RSA-CRT (Bellcore attack)

Boneh97

- Two signatures on same message
  - Correct
  - Faulty with **any** fault in s_1 or s_2
- Example: s_1 into s_1'
  - Faulty $s' = s_2 + (q_{inv}(s_1' - s_2) \bmod p)q$
  - Correct $s = s_2 + (q_{inv}(s_1 - s_2) \bmod p)q$
- Express $\delta = s - s'$

$$\begin{aligned}\delta &= s_2 + (q_{inv}(s_1 - s_2) \bmod p)q - (s_2 + (q_{inv}(s_1' - s_2) \bmod p)q) \\ &= (q_{inv}(s_1 - s_2) \bmod p)q - (q_{inv}(s_1' - s_2) \bmod p)q \\ &= q(q_{inv}(s_1 - s_2) \bmod p) - (q_{inv}(s_1' - s_2) \bmod p))\end{aligned}$$

- Now, why was I doing all of this?
  - δ is divisible by q but **not** by p!
  - $gcd(\delta, n) = q$, then $p = \frac{n}{q}$

ECDSA (Repeated nonces)

- ECDSA signature (r, s)
  - $r \equiv ([k]G).x \bmod n$
  - $s \equiv k^{-1}(h(m) + dr) \bmod n$
- Recall ECDSA sensitivity to nonce reuse and bias
- Faults?
  - RNG stuck-at-0
  - Skip over RNG completely
  - RNG request buffer length = 0
- Same nonce for two different messages
  - $r_1 \equiv ([k]G).x \equiv r_2 \equiv r$
  - $s_1 \equiv k^{-1}(h(m_1) + dr)$
  - $s_2 \equiv k^{-1}(h(m_2) + dr)$
  - Two equations, two unknowns, solve for privkey d

Deterministic ECDSA (Nonce difference)

- RFC6979: Derives nonce deterministically from message m and privkey d
  - $k = PRF(m, d)$
  - No RNG issues, right?
- Faults?
  - Nonce bit set/reset/flip/randomize
- Same message, two related nonces
  - $k_2 = k_1 + \delta$
  - $s_1 \equiv k_1^{-1}(h(m) + dr_1)$
  - $s_2 \equiv k_2^{-1}(h(m) + dr_2) \equiv (k_1 + \delta)^{-1}(h(m) + dr_2)$
  - Two equations, two unknowns, solve for privkey d
- What do you need from δ?
  - To be brute-forceable

ECDH (Safe error attacks)

Example

```
def scalarmult(P, k):
    R = infty
    for i in range(256, -1, -1):
        R = dbl(R)
        if k & (1 << i):
            R = add(R, P)
        else:
            dummy = add(R, P)
    return R
```

- Where do we fault?
  - The add
- What do we see?
  - $k_i = 0 \rightarrow$ correct result (fault was ineffective)
  - $k_i = 1 \rightarrow$ incorrect result (fault was effective)

Takeaways

- How faults break the assumptions we have on algorithm execution
  - Errors, skips, flips, tears may happen
- Faults are a very strong capability in an attacker's toolkit
- How fault attacks differ (generic, symmetric/asymmetric crypto)

Next weeks

- Week after (13th week)
  - Guest lecture by **Thomas Roche**
- Week after that (14th week)
  - Guest lecture by **Lukasz Chmielewski**
- Last week (15th week)
  - In person lecture: **Certifications & Recap**
  - Tutorial
    - We will be designing exam questions together
    - Some will make it into the exam itself

Recommended reading

- For **this** lecture
  - Skim Embedded Cryptography 1 – Chapter 9 – Introduction to Fault Injection Attacks
- For the **next** lecture
  - Read Embedded Cryptography 1 – Chapter 12 – Fault countermeasures

References

- Plundervolt: Software-based Fault Injection Attacks against Intel SGX
- RowHammer: Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors
- CLKSCREW: Exposing the Perils of Security Oblivious Energy Management
- DFA: Differential fault analysis of secret key cryptosystems
- SFA: Fault Attacks on AES with Faulty Ciphertexts Only
- AES DFA: Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault
- AES DFA original: A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad
- AES DFA revisited: Piret and Quisquater's DFA on AES Revisited
- RSA Bellcore: On the Importance of Eliminating Errors in Cryptographic Computations
- Differential Attacks on Deterministic Signatures
- Attacking Deterministic Signature Schemes Using Fault Attacks
- A Note on Fault Attacks Against Deterministic Signature Schemes