Information Security

System Security 3 - Physical Side-Channel and Fault Attacks

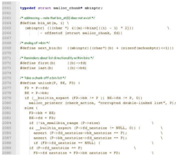
19.11.2021


0

“Human Side-Channel Analysis”

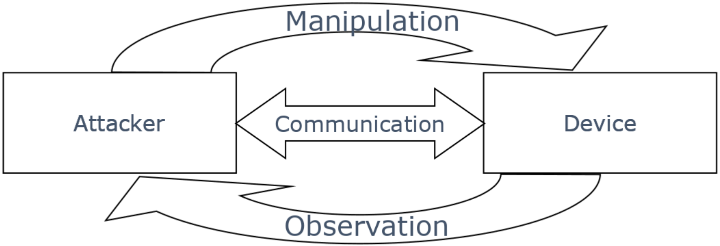
Attacks with Physical Access

"If the attacker can execute code
... they have already won"



```
typedef struct malloc_chunk* mbinptr;

/* addressing -- note that bin_at(0) does not exist */
#define bin_at(m, i) \
  (mbinptr) (((char *) &((m)->bins[((i) - 1) * 2]))              \
             - offsetof (struct malloc_chunk, fd))

/* analog of ++bin */
#define next_bin(b)  ((mbinptr)((char*)(b) + (sizeof(mchunkptr)<<1)))

/* Reminders about list directionality within bins */
#define first(b)     ((b)->fd)
#define last(b)      ((b)->bk)

/* Take a chunk off a bin list */
#define unlink(P, BK, FD) {
  FD = P->fd;
  BK = P->bk;
  if (__builtin_expect (FD->bk != P || BK->fd != P, 0))
    malloc_printerr (check_action, "corrupted double-linked list", P);
  else {
    FD->bk = BK;
    BK->fd = FD;
    if (!in_smallbin_range (P->size)                      \
    && __builtin_expect (P->fd_nextsize != NULL, 0)) {         \
      assert (P->fd_nextsize->bk_nextsize == P);               \
      assert (P->bk_nextsize->fd_nextsize == P);               \
      if (FD->fd_nextsize == NULL) {                       \
    if (P->fd_nextsize == P)                      \
      FD->fd nextsize = FD->bk nextsize = FD;              \
```

Physical Attack Principle



~~SECRET~~

(b)(3)-P.L. 86-36

TEMPEST: A Signal Problem

**The story of the discovery
of various compromising radiations
from communications and Comsec equipment.**

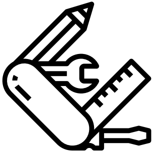
impractical. Hydraulic techniques—to replace the electrical—were tried and abandoned, and experiments were made with different types of batteries and motor generators, in attempts to lick the power-line problem. None was very successful.

During this period, the business of discovering new TEMPEST threats, or refining techniques and instrumentation for detecting, recording, and analyzing these signals, progressed more swiftly than the art of suppressing them. Perhaps the attack is more exciting than the defense—something more glamorous about finding a way to read one of these signals than going through the drudgery necessary to suppress that whacking great spike first seen in 1943. At any rate, when they turned over the next rock, they found the acoustic problem under it. Phenomenon No. 5.

Acoustics

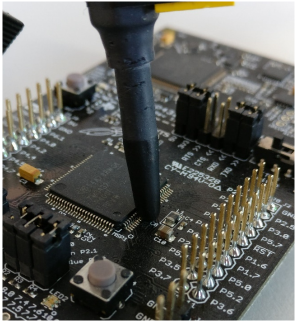
We found that most acoustic emanations are difficult to exploit if the microphonic device is outside of the room containing the source equipment; even a piece of paper inserted between, say, an offending keyboard and a pick-up

Physical Attacks: Categorization

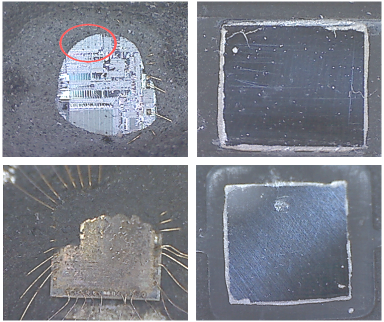


- Behavior of the attacker
  - Side-channel attack: passively observe physical properties
  - Fault attack: actively manipulate device to induce faults
- Degree of invasiveness
  - Non-invasive: Device is not altered physically
  - Semi-invasive: De-packaging, no electrical contact to internal signals
  - Invasive: No limits

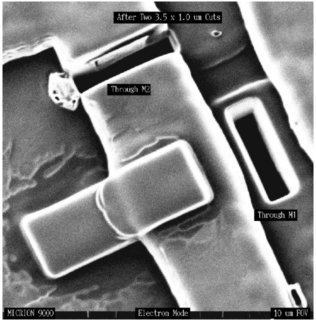
Degree of Invasiveness



Non-Invasive

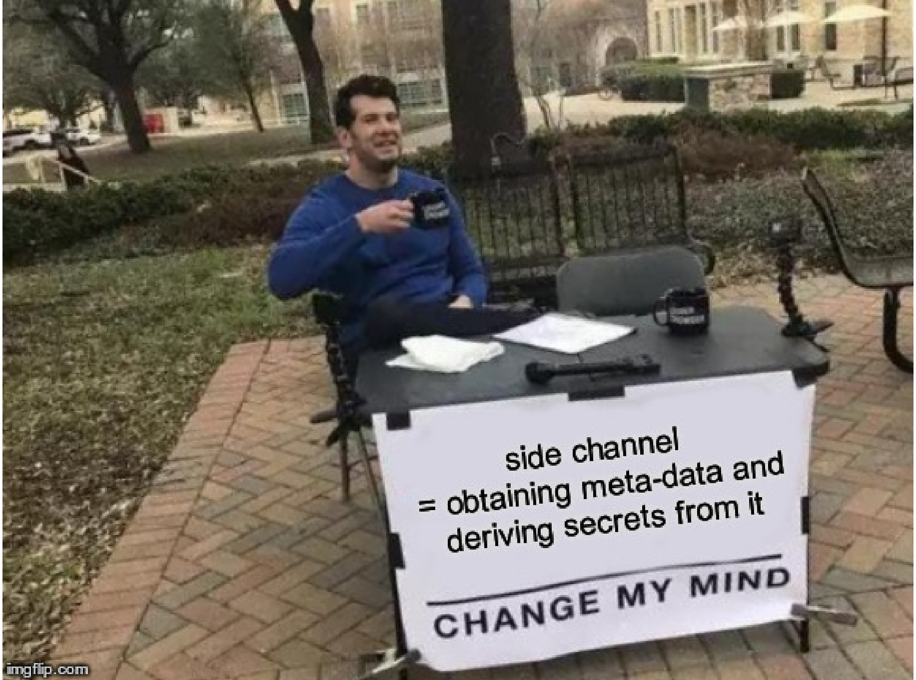


Semi-Invasive

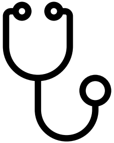


Invasive

Side-Channel Attacks


side channel
= obtaining meta-data and
deriving secrets from it
CHANGE MY MIND
imgflip.com

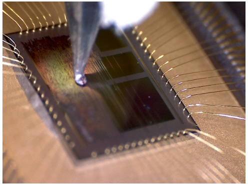
Basic Idea of Side-Channel Attacks

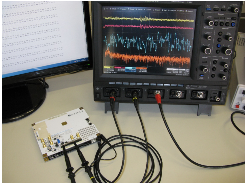


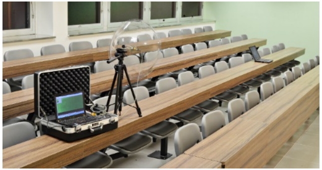
Any computation influences physical properties (meta-data)

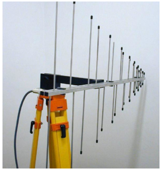
- Computations depend on secrets (data)
- We observe properties (meta-data) to infer secrets (data)

Side-Channel Attacks



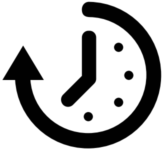






- Timing
- Power consumption
- EM emanations
- Sound
- ...

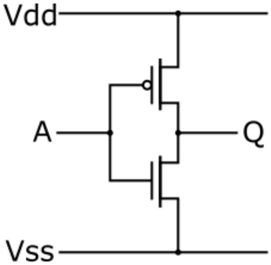
Timing Attacks on Cryptographic Implementations



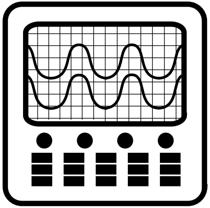
- Often overlooked / ignored
  - "outside of threat model"
  - implementation bugs
- Sometimes even on certified devices (e.g., Minerva and TPM-Fail)

→ Solution: Make everything constant-time?

CMOS Circuits

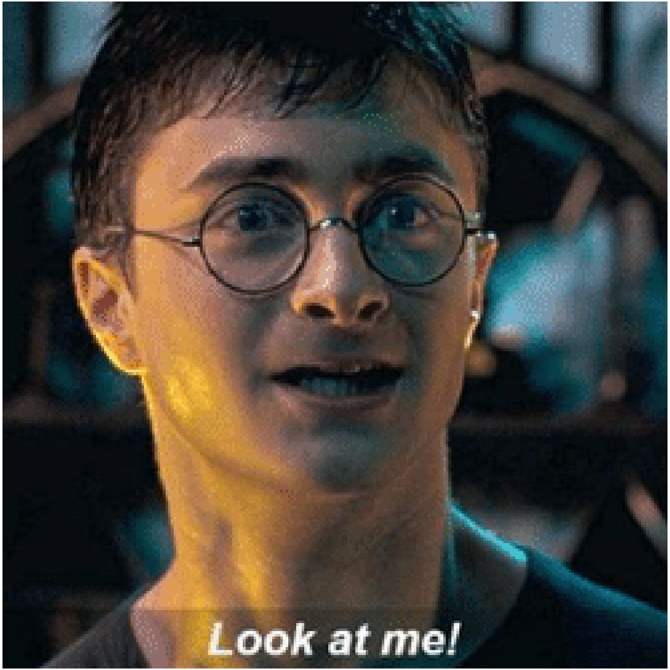


- Complementary Metal Oxide Semiconductor
- Today's digital circuits
- Nice properties:
  - high noise immunity
  - low power consumption
  - (Only switching draws power)
- Wait a second... switching draws power?

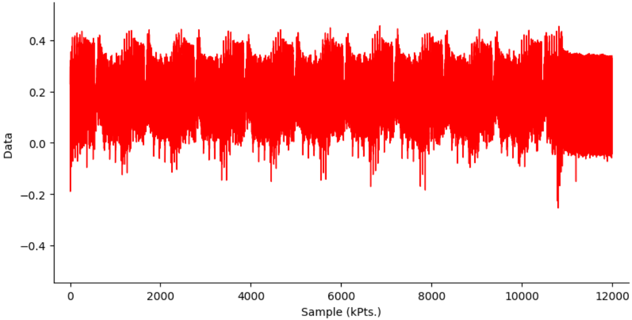


- Different instructions / data $\rightarrow$ different switching
- Idea: Measure power consumption during operation
  - sampling rate up to gigasamples (10^9 measurements per second)

  $\rightarrow$ a measured power-consumption curve is called a **power trace**
- First signal-processing step?


Look at me!

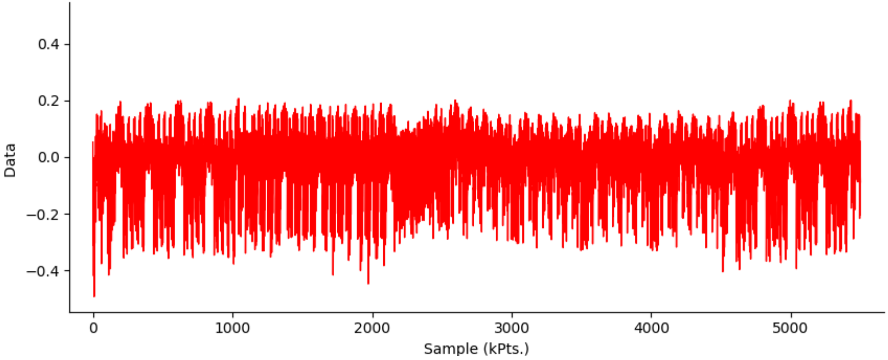
Power Consumption - An Example Trace



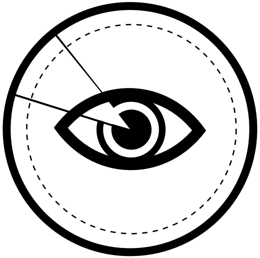
What do we see here?

10 rounds of AES

Power Consumption - Zoom in on Single Round



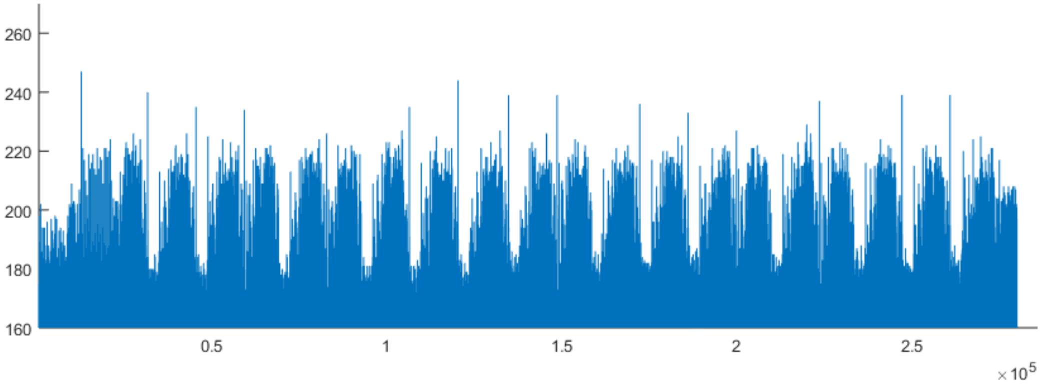
What do we see?



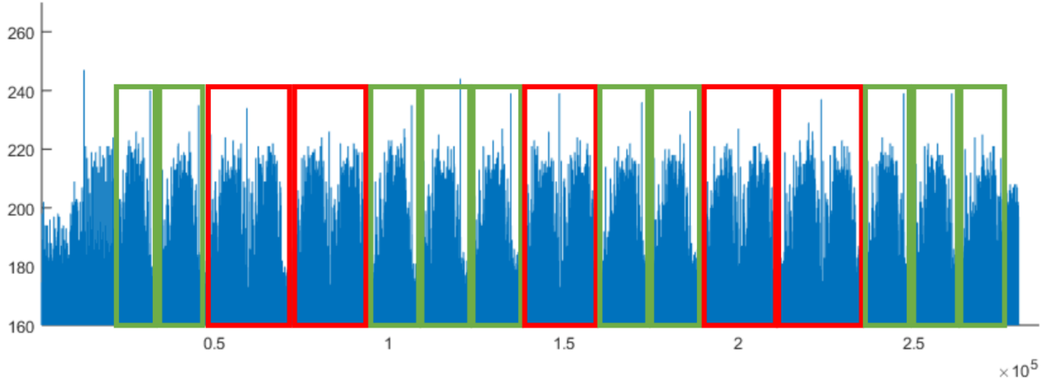
- Operations / Instructions
- Repeated patterns and variations of patterns
  - Loops, repeated operations, taken branches
  - Learn control flow / instruction sequence
- Jumps in power consumption profile
  - Memory accesses (especially EEPROM or flash programming)
  - Access to peripherals (e.g., co-processors, I/O)

→ Can we exploit that?

Our First Power-Analysis Attack



Our First Power-Analysis Attack



We see a power trace of an RSA exponentation ($m = c^d \mod n$)

How to get the key from that?

Excursion: Efficient Implementation of Modular Exponention

RSA decryption: $m = c^d \bmod n$, where n has ≥ 2048 bits

Efficient implementation?

- Compute $(c^d) \bmod n$?
- c and d are also ≥ 2048 bits
- c^d has more than 2^{2048} bits!

Some reminders for modular arithmetic:

- $a \cdot b \bmod n = (a \bmod n) \cdot (b \bmod n) \bmod n$
- $c^{a+b} \bmod n = (c^a \bmod n) \cdot (c^b \bmod n) \bmod n$
- $c^{a \cdot b} \bmod n = (c^a \bmod n)^b \bmod n$

Excursion: Efficient Implementation of Modular Exponention

- Look at exponent d in binary: $d_i = i$th bit of d, where $d_0 = \text{LSB}$
- Recursive decomposition of exponentiation:
  - We can write $d = 2 \cdot \lfloor d/2 \rfloor + (d \bmod 2) = 2 \cdot (d \gg 1) + d_0$
    In the exponent, we get: $c^d = (c^{\lfloor d/2 \rfloor})^2 \cdot c^{d_0}$
  - But $c^{\lfloor d/2 \rfloor}$ is still way too large, so repeat:
    $c^{\lfloor d/2 \rfloor} = (c^{\lfloor d/4 \rfloor})^2 \cdot c^{\lfloor d/2 \rfloor \bmod 2} = (c^{\lfloor d/4 \rfloor})^2 \cdot c^{d_1}$
  - ...until $\lfloor d/2^x \rfloor = 1 \rightarrow c^{\lfloor d/2^x \rfloor} = c$
- Iterative version: Start at $\lfloor d/2^x \rfloor = 1$ and work our way up

Excursion: Efficient Implementation of Modular Exponention

- Algorithm: Left-to-right *Square-and-Multiply* exponentiation

$m \leftarrow 1$ // init

for $i = 2047$ **downto** 0 : // scan bits from MSB to LSB

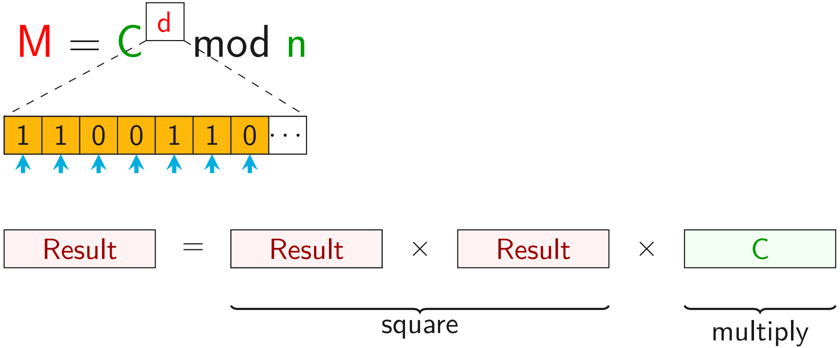
$m \leftarrow m^2 \bmod n$ // squaring: $c^x = (\mathbf{c}^{\lfloor \mathbf{x}/2 \rfloor})^{\mathbf{2}} \cdot c^{x_0}$

if $d_i = 1$ **then:** // if bit is set (else $x_0 = 0 \rightarrow c^{x_0} = 1$, can skip mult.)

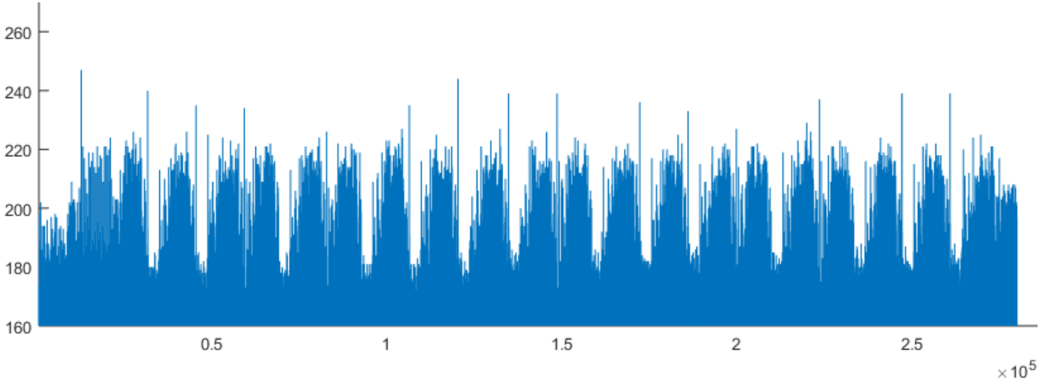
$m \leftarrow m \cdot c \bmod n$ // then multiply: $c^x = (c^{\lfloor x/2 \rfloor})^2 \cdot \mathbf{c}^{\mathbf{x_0}}$

- Example: $d = 26 = 1\,1\,0\,1\,0_b \rightarrow c^{26} = ((((1^2 \cdot c)^2 \cdot c)^2)^2 \cdot c)^2$

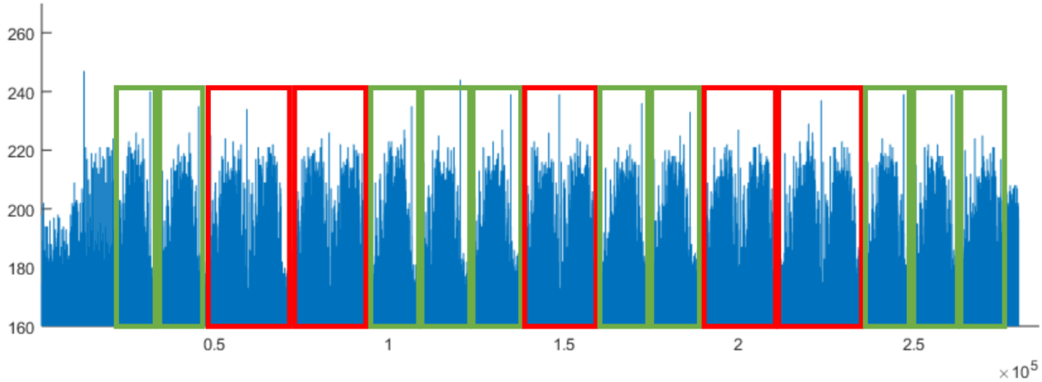
Excursion: Efficient Implementation of Modular Exponention



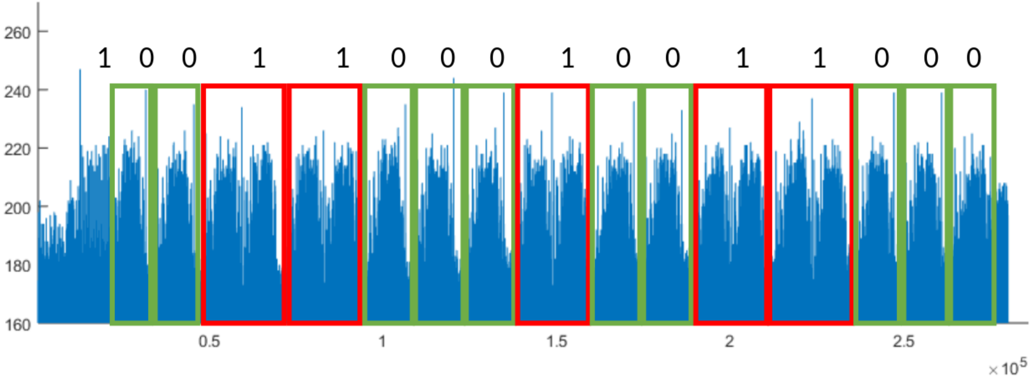
Our First Power-Analysis Attack - Key Recovery



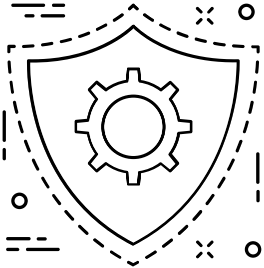
Our First Power-Analysis Attack - Key Recovery



Our First Power-Analysis Attack - Key Recovery



Power Side Channel Countermeasures



"Constant-time" means more than just *constant time*

- No branching on secret data: **constant runtime and control flow**

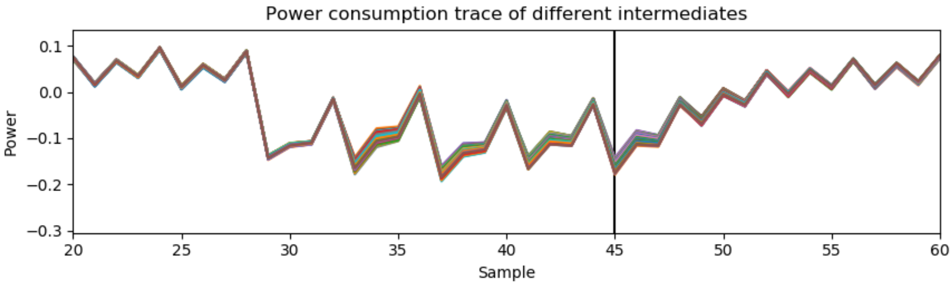
→ always same instruction sequence but different data

- More secure alternatives:
  - Constant-time exponentiation algorithms
  - Constant-time modular reduction
  - ...

And now: "Constant-time" → all problems solved?

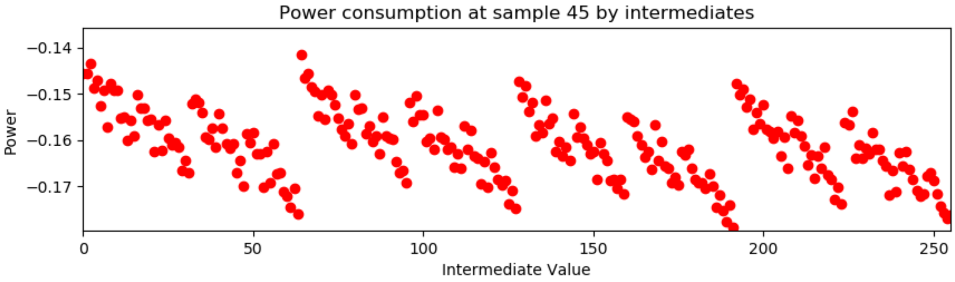
What about data differences?

CMOS Power Consumption: Data Dependency



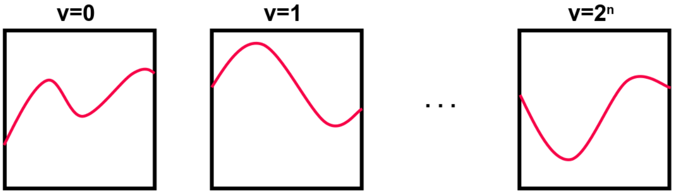
Averaged power traces of a load instruction for values $\{0, 255\}$

CMOS Power Consumption: A Closer Look



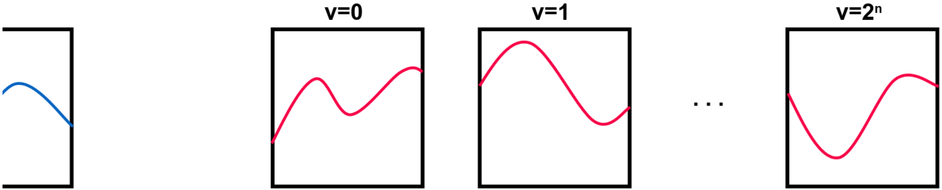
Different intermediate values → different power consumption

Record + match values = *Template Attack*



- Profile power consumption for each possible value of intermediate v
- Record traces with all inputs known, group by v
- Profile == "Template"

Template Attack - Phase 2 - Attack / Exploitation

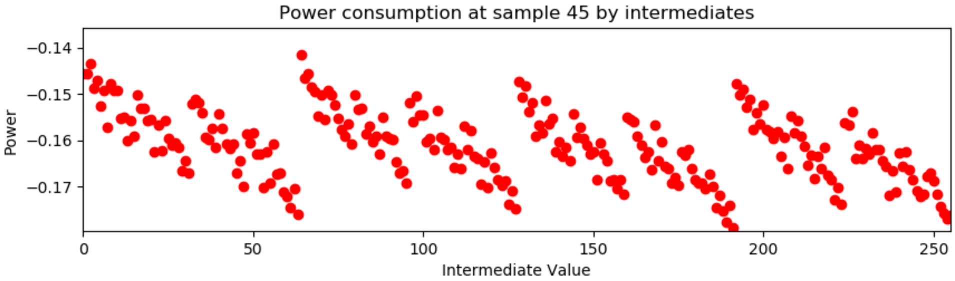


- Compare (match) measured traces to all templates
- Use v which fits best (compute probabilities)

Template Attacks: Conclusion

- Pro: Very powerful
  - Key recovery with single trace
  - Sometimes the only option (“we only have a single trace”)
- Contra: many prerequisites and detailed knowledge needed
  - When is secret processed?
  - What is the concrete algorithm?
  - Identical device / setup needed where you can control **all** inputs

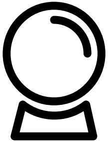
Another Look at Intermediate Values



There is some kind of pattern...

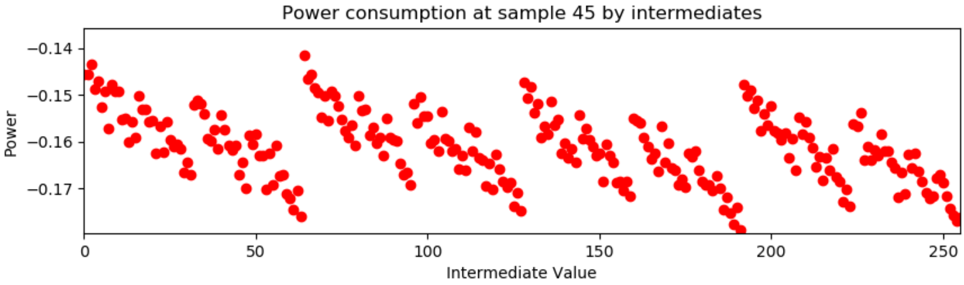
We can *model* the power consumption

Hamming Weight Power Model



- "Power consumption depends on switching"
- What's stored before a value is stored? Assume 0
- Now the new value: each '1'-bit draws power $\rightarrow$ power is proportional to number of bits set
- number of bits set == Hamming weight

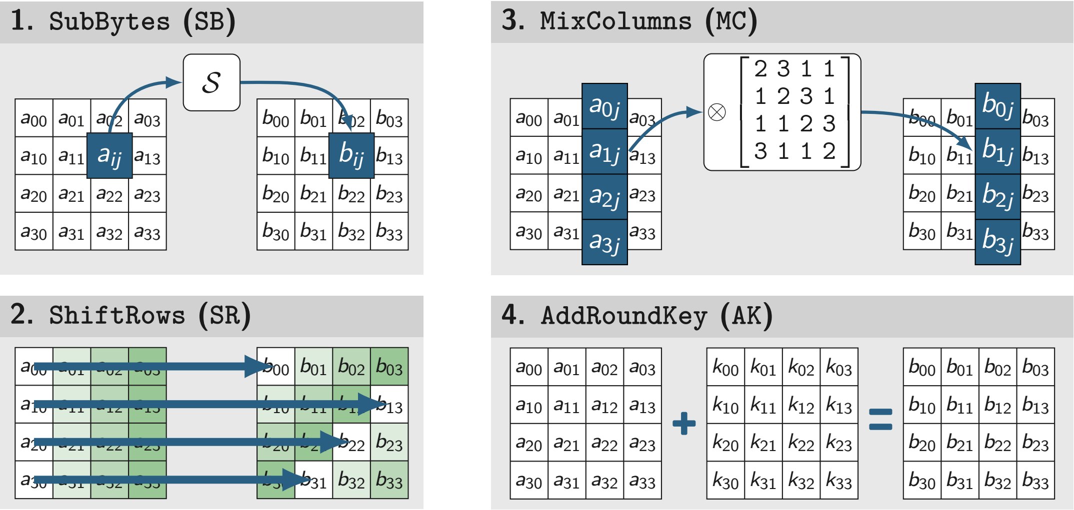
And Another Look at Intermediate Values



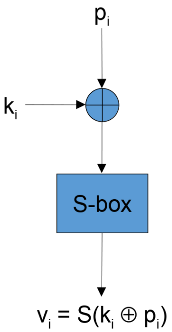
Many devices have similar power behavior → reuse power models

→ an attack without detailed knowledge of device an concrete implementation!

Reminder: AES-128 Block Cipher (10 Rounds)



Excursion: AES

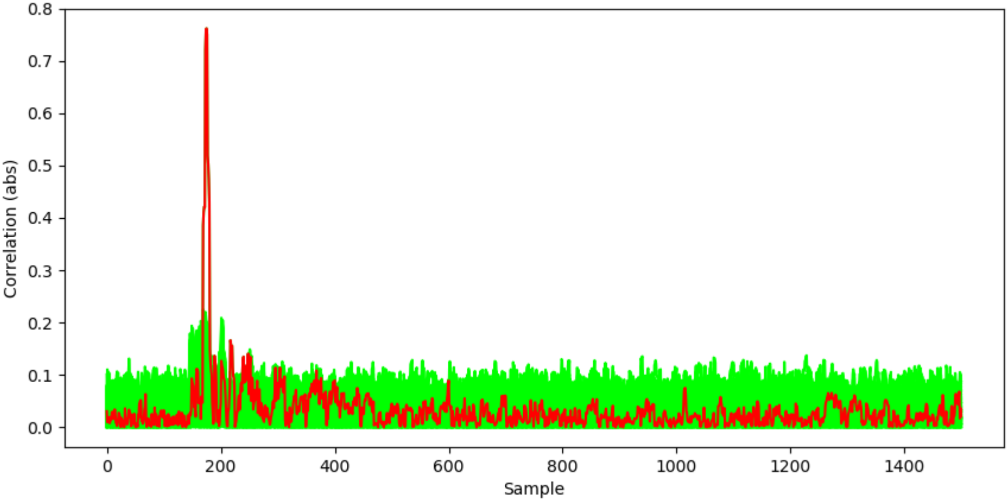


- First round: round key = key
- Other rounds: key schedule
  - key schedule is invertible

Differential Power Analysis (DPA)

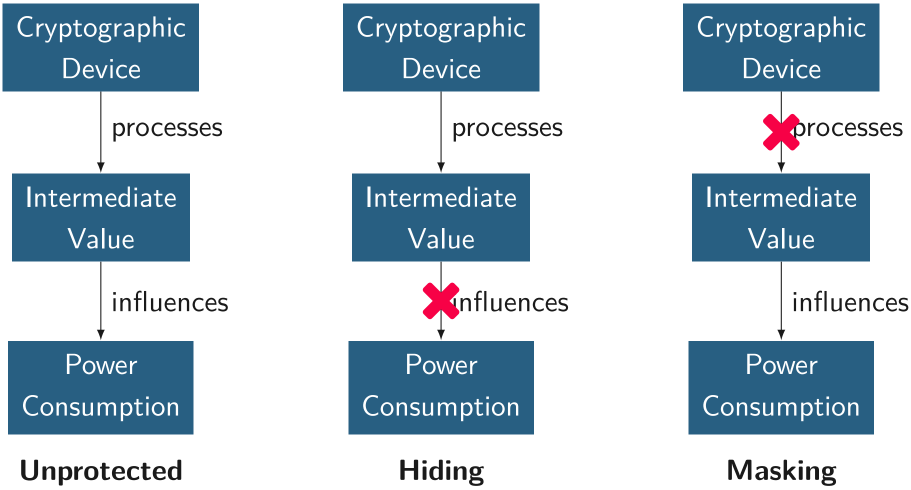
Differential Power Analysis (DPA) in 5 Steps

1. **Select intermediate** value that depends on a small number of key bits (subkey)
2. **Measure power** while querying device
3. **Enumerate** all **possible** subkey **values**
   - 2^8 key hypotheses
   - for each plaintext/ciphertext: predict intermediate for each key hypothesis
4. **Predict power** consumption of intermediate (power model, e.g., Hamming weight)
5. **Compare** prediction with measurement
   - pick key hypothesis that fits best
   - statistical hypothesis tests



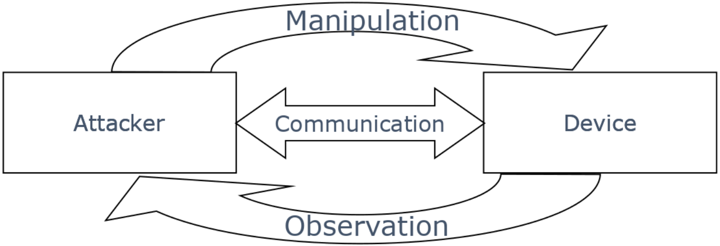
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Correlation (abs)
0
200
400
600
800
1000
1200
1400
Sample

Countermeasures against Power Analysis



Fault Attacks

Fault Attacks



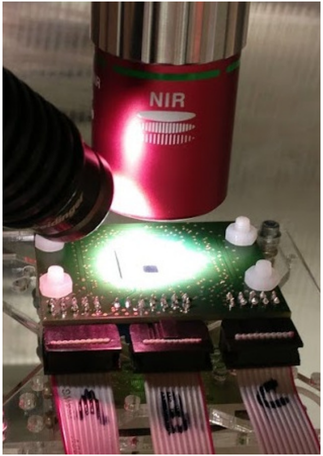
Just listening is boring ...

$\rightarrow$ let's manipulate things more actively

Basic Idea of Fault Attacks

- Goal: manipulate device to compromise security
- Change behavior
  - Deactivate countermeasures / sensors
  - Skip PIN check
- Fault crypto algorithms
  - Compute faulty and correct ciphertexts
  - Use difference to reveal key

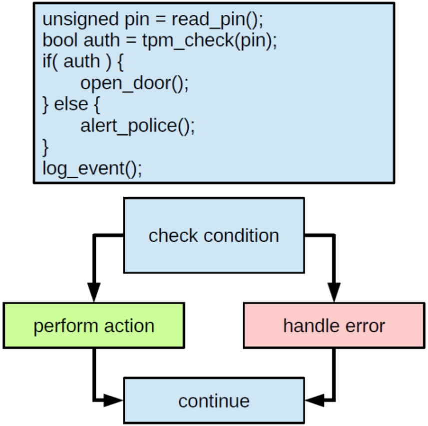
Fault Attack Techniques



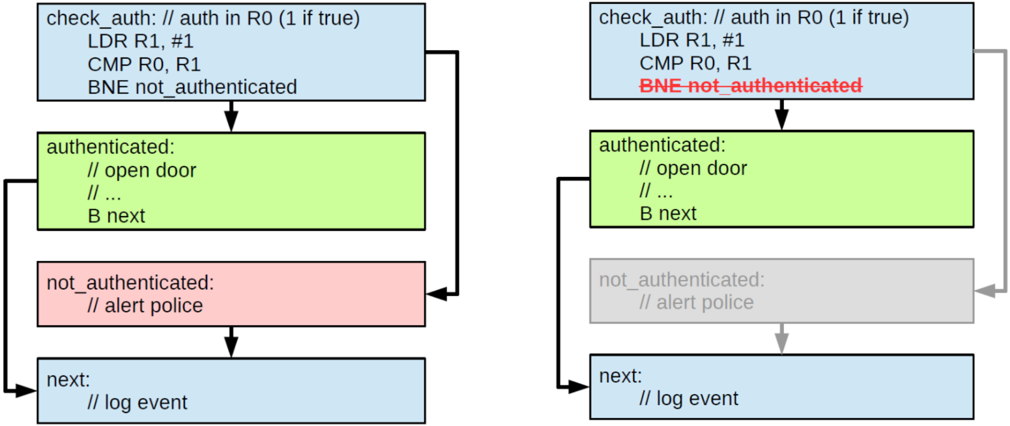
- Spike / glitch: clock, voltage, etc.
- Heat, Radiation, Laser

- Effects:
  - Instructions skipped
  - Data corrupted
  - ...

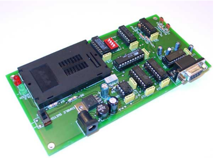
Example: PIN Check



Example: PIN Check - Skipping Attack



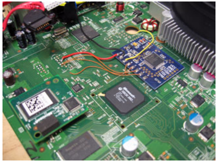
Real-World Example: Piracy



PayTV (early 2000s)

- vendors bricked pirated cards via firmware update
- insert endless loop in startup
- solution: glitch to escape loop (“unlooper device”)

Real-World Example: Piracy



Gaming devices

- Xbox360 reset hack
- voltage glitching on reset line
- execute untrusted modified firmware

Fault Attack on RSA

Excursion: Efficient RSA Signature with Chinese Remainder Theorem

- RSA signatures: $S = M^d \bmod n$, where $n = p \cdot q$

- Efficient implementation trick: Chinese Remainder Theorem (CRT)
  - Compute signature result $\bmod p$ and $\bmod q$

$$S_p = S \pmod{p} = M^{d \bmod p-1} \pmod{p}$$
$$S_q = S \pmod{q} = M^{d \bmod q-1} \pmod{q}$$

  - . . . and merge the results with CRT:

$$S = S_p \cdot (q^{-1} \bmod p) \cdot q + S_q \cdot (p^{-1} \bmod q) \cdot p \pmod{p \cdot q}$$

  - 2 exponentiations with half the bit-length and smaller exponents

Fault Attack on RSA Signatures with CRT ("Bellcore attack")

- Compute signature twice and fault one computation $\lightning$

$$S = \boxed{\text{something}} \cdot q + \boxed{\text{some rest}} \pmod{n}$$

$$S^{\lightning} = \boxed{\text{faulty}} \cdot q + \boxed{\text{some rest}} \pmod{n}$$

$$S - S^{\lightning} = \boxed{\text{some garbage}} \cdot q + \boxed{0} \pmod{n}$$

- Get the secret q using

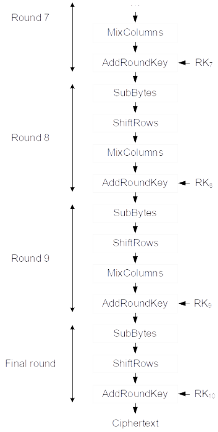
$$\gcd(S - S^{\lightning}, n) = \gcd(\boxed{\text{some garbage}} \cdot q,\ p \cdot q) = q$$

```
bagger> dog Enclave/encl
```


MUFARSA
My Undervolting
Fault Attack on RSA
imgflip.com

Fault Attack on AES

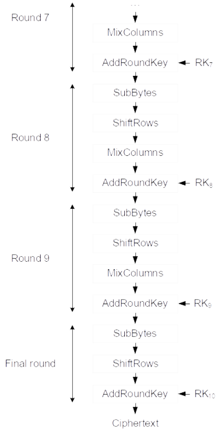
Differential Fault Attacks on AES



Faulting ciphertext?

- **No.**
- ciphertext difference does not depend on key

Differential Fault Attacks on AES



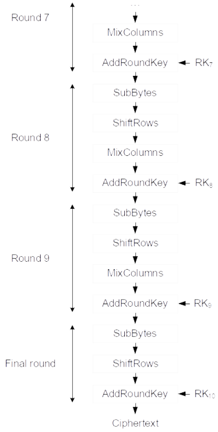
Faulting before `AddRoundKey10`?

- depends on faults
- not with bit flips (random or known)
- $\rightarrow$ fault propagates through $\oplus$:

  $c = v \oplus k$

  $c' = (v \oplus \Delta v) \oplus k = c \oplus \Delta v$
- $\rightarrow$ ciphertext difference still does not depend on key

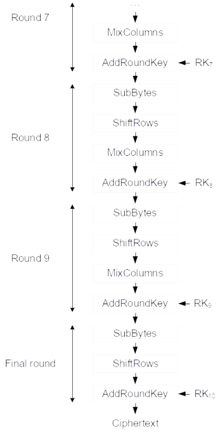
Differential Fault Attacks on AES



Faulting before `ShiftRows10`?

- same as before
- `ShiftRows` just rearranges bytes

Differential Fault Attacks on AES

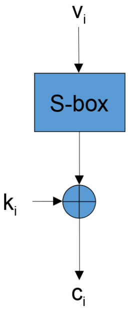


Faulting before `SubBytes10`?

- ...depends on faults
- Able to flip 1 bit?

→ Attacks possible

Single-Bit Fault before SubBytes

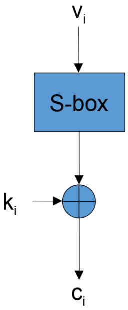


Receive correct and faulty ciphertext

Enumerate all 2^8 values for k_i

- compute back to v (for correct and fault, for all possible k_i)
- compute Δv for k_i
- check if Δv follows ault model (1 bit fault)
- indices can be different because of `ShiftRows`

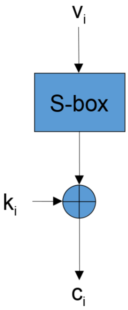
Single-Bit Fault before SubBytes - Example



Correct Output = 1A Faulty Output = 99

```
      k:  0  1  2  3  4  5  6  7  8 ...
------------------------------------------
C = 1a : S^-1(C  xor k):
C'= 99 : S^-1(C' xor k):
```

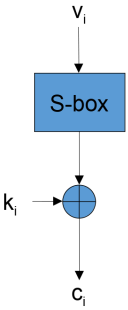
Single-Bit Fault before SubBytes - Example



Correct Output = 1A   Faulty Output = 99

```
                                   k:  0  1  2  3  4  5  6  7  8 ...
---------------------------------------------------------------------
C = 1a : S^-1(C  xor k): 43 44 34 8e e9 cb c4 de 39 ...
C'= 99 : S^-1(C' xor k): f9 e2 e8 37 75 1c 6e df ac ...
```

Single-Bit Fault before SubBytes - Example



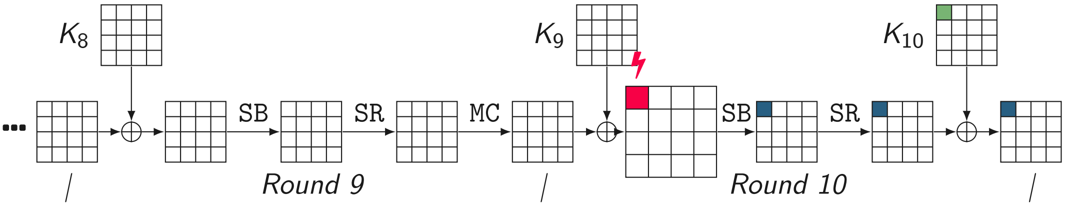
Correct Output = `1A`   Faulty Output = 99

```
                         k:  0  1  2  3  4  5  6  7  8 ...
--------------------------------------------------------
C = 1a : S^-1(C  xor k): 43 44 34 8e e9 cb c4 de 39 ...
C'= 99 : S^-1(C' xor k): f9 e2 e8 37 75 1c 6e df ac ...
                                                ^^
```

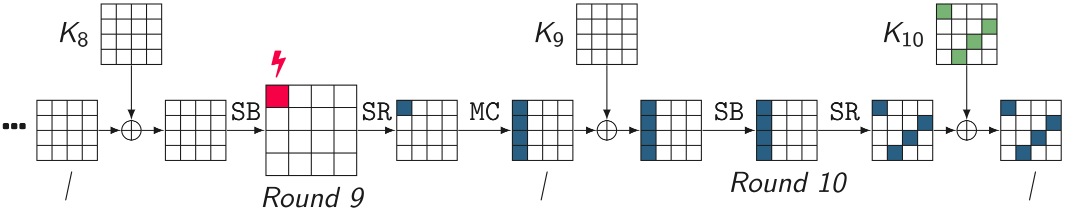
Only few keys have this property → filter them

Use further `C/C'` pairs to get down to 1 key

- Assume the attacker can cause precise 1-bit flips in Round 9 of AES, before S-box
- For each of 2^8 key guesses, Test if the *partial decryption* produces the expected 1-bit flip.



- Assume the attacker can cause imprecise 1-byte errors
- For each of 2^{32} key guesses,
  Test if the *partial decryption* produces the expected 1-byte error.
  (This can be optimized to require only 2 faulty encryptions to recover the full key)



AES-NI (New Instructions)

Instruction	Description
`AESENC`	Perform one round of an AES encryption flow
`AESENCLAST`	Perform the last round of an AES encryption flow
`AESDEC`	Perform one round of an AES decryption flow
`AESDECLAST`	Perform the last round of an AES decryption flow
`AESKEYGENASSIST`	Assist in AES round key generation
`AESIMC`	Assist in AES Inverse Mix Columns
`PCLMULQDQ`	Carryless multiply (CLMUL)

AES New Instructions - inside SGX

```
do
{
  i++;
  plaintext = <randomly generated>

  result1 = aes128_enc(plaintext);
  result2 = aes128_enc(plaintext);
} while (vec_equal_128(result1,result2) && i<iterations);
```

```
bagger> sudo ./aes-encrypt 100000 -262
```

CRYPTO VAULT SCREW HAMMER

Countermeasures for Fault Attacks

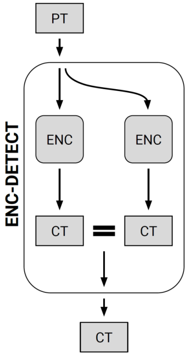
Analog Countermeasures

Detect anomalies[a]

- Active fine wire meshes across IC $\rightarrow$ disruption is detected
- Power surge sensors
- Temperature sensors
- Light sensors

[a] IBM 4767 Hardware Security Module battery-backed monitoring, meshes, light sensors, temperature sensors, etc. immediate deletion of keying material on tamper detection

Redundancy-based Countermeasure: Double Execution



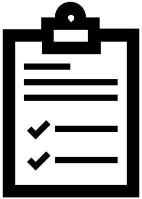
Encrypt multiple times, compare result

- comparison at different granularities possible:
- encryption, single round, each operation, ...

But the attacker might be able to

- inject the same fault twice (difficult ...)
- or use more sophisticated methods (statistical attacks)

Take Aways



Attack → defense → next attack → next defense → ...

- different side channels, more sophisticated attacks
- a never-ending cat-and-mouse game

There is no “100% secure”, especially in the physical setting

- any device can be broken by a determined attacker

Our goal:

- Ensure that attack effort is much greater than the value of the secret
- or: Would you do an attack that costs millions to get a free tram ride?

Thanks to Peter Pessl for some of the slides!