Information Security
Side-Channel Attacks (Part 1)

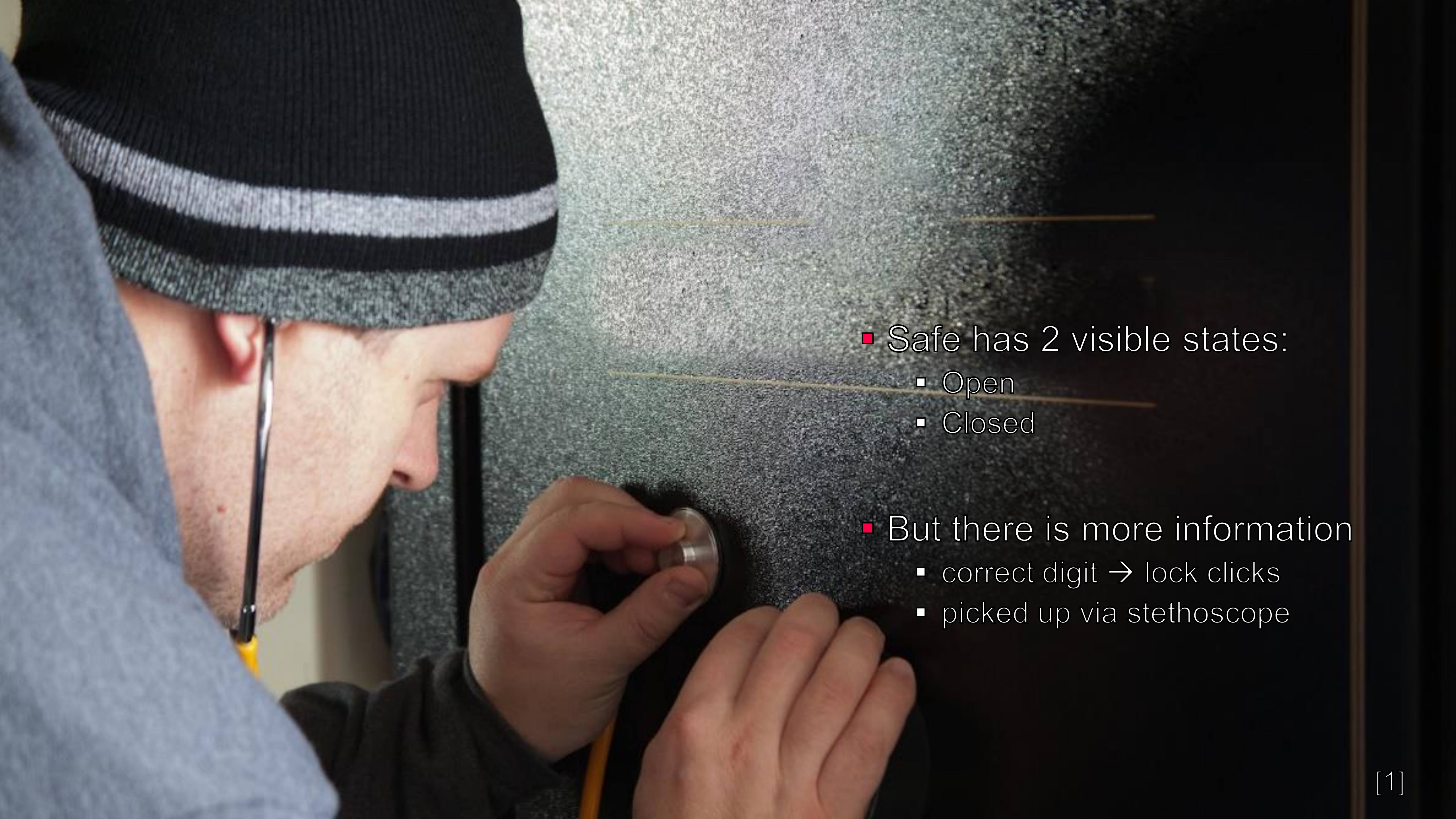
Peter Pessl
Graz University of Technology

What is a Side Channel?

- Systems want to tell us things
  - for that, they have an **intended communication channel**
    (Data sent over USB, Ethernet, Software API, …)
  - but often, they inadvertently tell us much more through **unintended side channels**

- Side channel
  - any source of information on some secret besides actual communication channel
  - side-channel analysis / attacks: making use of information to recover the secret

- Many real-life examples…



- Safe has 2 visible states:
  - Open
  - Closed
- But there is more information
  - correct digit → lock clicks
  - picked up via stethoscope

[1]

- Humans also have side channels
  - gestures
  - breathing
  - tone of voice
  - sweating
  - …
- We are conditioned to pick them up
  - „unconcious side-channel analysis¨

“Human Side-Channel Analysis”

Measuring human side-channels with a lie detector

Side-Channel Attacks on IT Systems

- Attacker uses as much information sources as possible
  - …no need to restrict yourself ot the main communication channel

- To retrieve secrets such as
  - passwords, PINs, keys
  - user activity (visited websites, running programs, …)
  - keystroke timings
  - …

Attack Settings

- There is loads of opportunity for side-channel attacks in IT
  - computers are complex
  - huge attack surface
- We cover 2 (very broad) attack settings
  - Microarchitectural side-channel attacks
  - Physical side-channel attacks
- But there are many more
  - Sensor-based attacks: use smart phone sensors to detect taps etc.
  - …

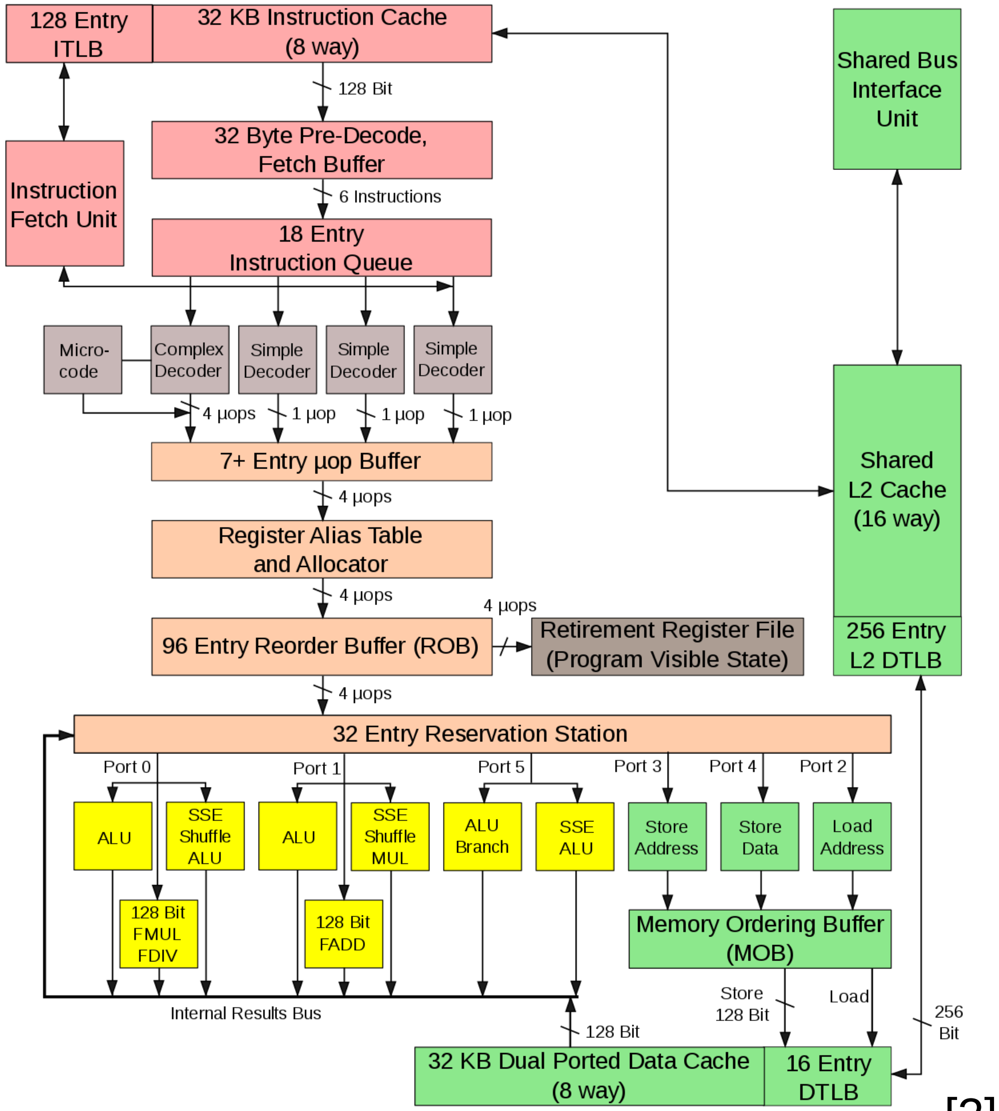
Microarchitecture

- Specified interface of processors: Instruction Set Architecture (ISA)
  - instructions and their encoding (opcodes)
  - register set
  - addressing modes, etc.
  - examples: x86, x86_64, ARMv8

- How ISA is actually implemented: Microarchitecture
  - execution units (ALU etc.), decoders, pipelines, caches, …

Example: Intel Core 2 Microarch.

- Implementation of x86_64
- Many different microarch. for single ISA

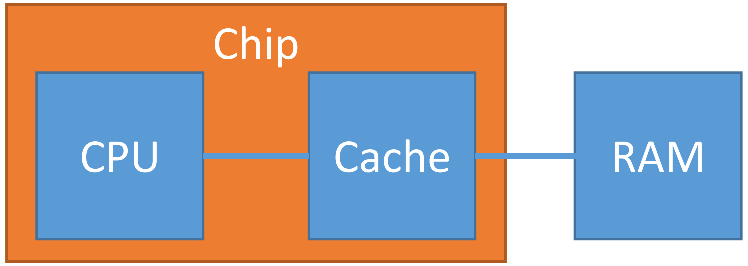


[2]

Intel Core 2 Architecture

Microarchitectural Side-Channel Attacks

- Problem: many building blocks…
  - are shared between multiple cores
  - have an internal state (storage)
    (usually hidden, but side-channels…)



- CPU-cache Attacks
  - cache: fast data buffer between CPU and RAM
  - Alice: may access address *0xabcd* in shared memory
  - Bob: access *0xabcd*, measure time
    - access is fast: *0xabcd* was cached → Alice accessed it before
    - access is slow: *0xabcd* was not cached → Alice did not access it before

Features and Limitations

- Attacks purely in software
  - no physical access to machine needed

- But code execution on attacked device required
  - multi-user machine (cloud)
  - Javascript in browser

More details on Microarchitectural Attacks: next week!

But for many devices…

If attacker can execute code

…he has already won ☠

Physical (Side-Channel) Attacks are still a threat

The Setting of Physical (Side-Channel) Attacks

Device with Assets

can communicate and send commands, but no code exeuction

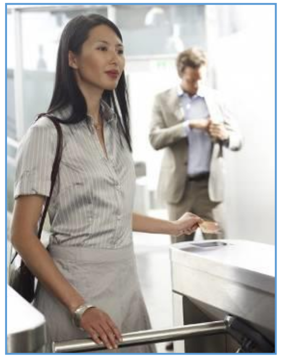
Hands of the Attacker

possesion or close vicinity

Classic Applications Exposed to Physical Attacks

- Payment
- Government IDs
- Transportation
- Brand protection: printer cartridges, batteries, …
- IP protection: source code, netlists, …
- Digital rights management

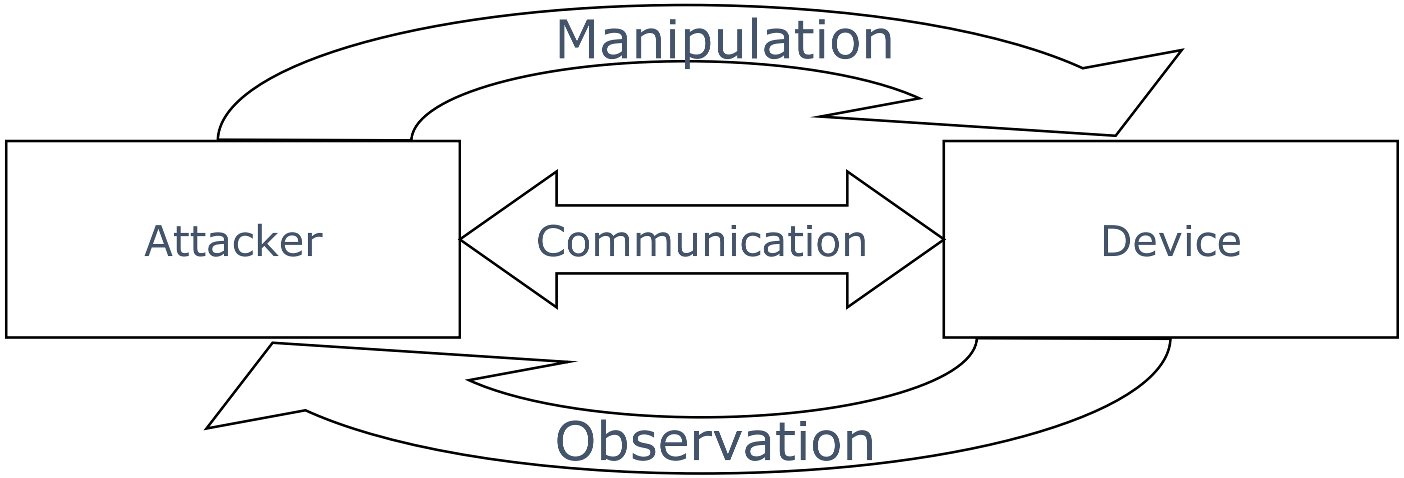

```c
typedef struct malloc_chunk* mbinptr;

/* 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;         \
```



Summary: Many scenarios, where attacker is in possession or vicinity of the device

Attacker can be a regular / legitimate user

Physical Attacks



observe or manipulate physical properties of the device or its environment

Classic example: Tempest/Van-Eck Phreaking

- Each cable is also an antenna…
  - antennas emit EM radiation
  - can we exploit this?

- Read display content!
  - Van-Eck Phreaking (1985)
  - Relatively cheap equipment
  - Up to 100m

350 MHz, 50 MHz BW, 12 frames (160 ms) averaged

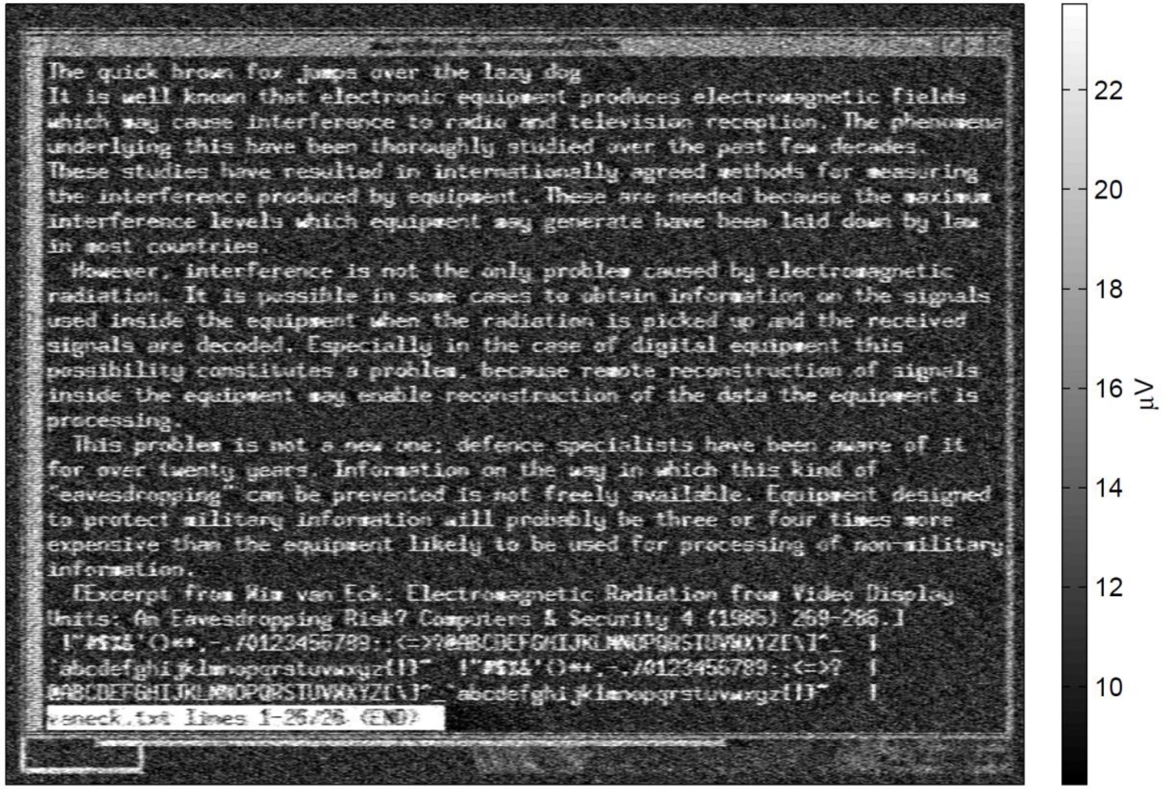


Figure 4.3: Text signal received from a 440CDX laptop at 10 m distance through two intermediate offices (3 plasterboard walls).

Classic example: Tempest/Van-Eck Phreaking

~~SECRET~~

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

Approved for Release by NSA on 09-27-2007, FOIA Case # 51633

TEMPEST: A Signal Problem

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

In 1962, an officer assigned to a very small intelligence detachment in Japan was performing the routine duty of found with microphones for? Why was there a large metal grid carefully buried in the cement of the ceiling over the

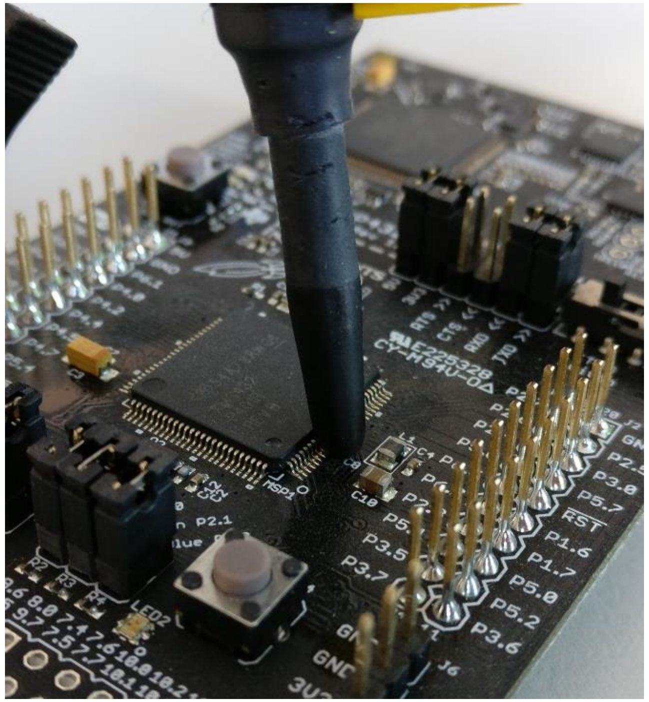
- Earliest techniques: 1943
- TEMPEST
  - spying
  - shielding

Physical Attacks - Categorization

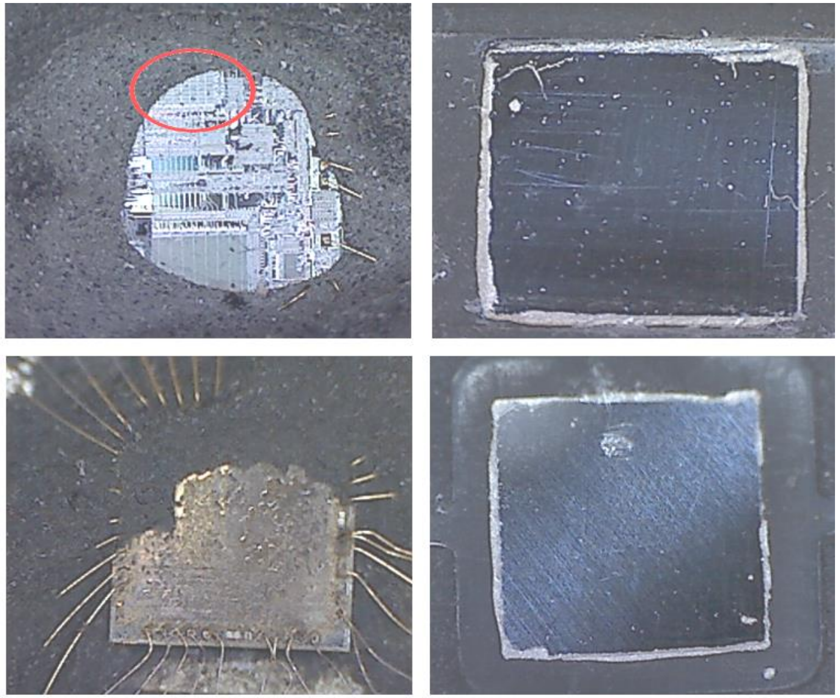
- Behavior of the attacker
  - Passive: only observes certain physical properties
  - Active: manipulate device to induce faults

- Degree of invasiveness
  - Non-invasive: Device is not altered physically
  - Semi-invasive: De-packaging, no electrical contact to internal signals
  - Invasive: No limits

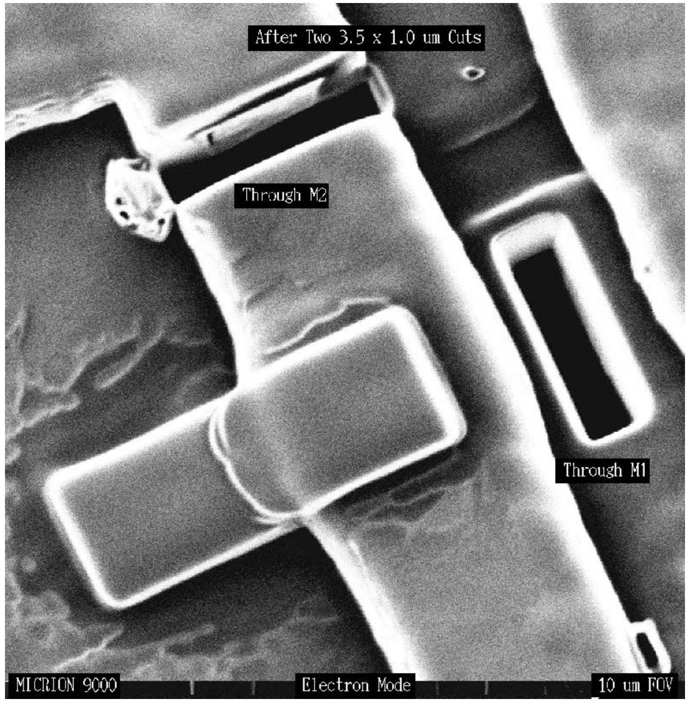
Degree of Invasiveness



Non-Invasive



Semi-Invasive



Invasive

Passive Attacks

Attacking by observing physical properties of the device

Basic Idea of Passive Attacks

Any computation influences physical properties

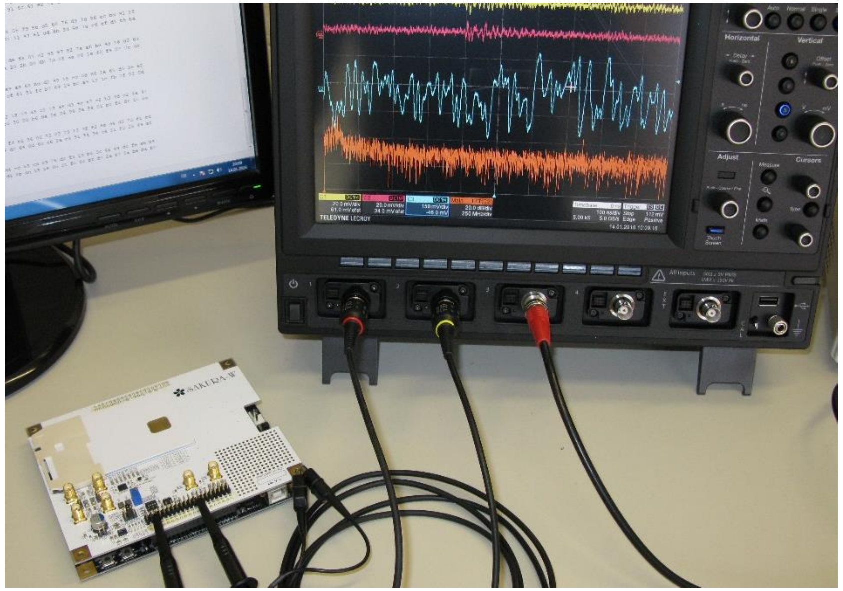
…computations depend on secret

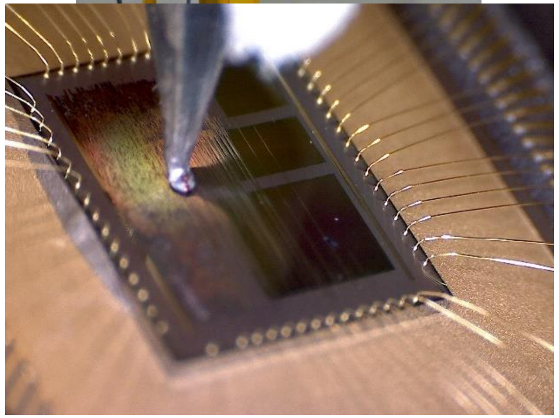
…→observe properties to reveal secret

Physical properties such as:

Passive Side-Channels

- Timing
- Power consumption
- EM emanations
  - long range: meters
  - short range: on-chip emanations
- Sound
  - Keyboards
  - Feeping of PC
- …





But…

- same ideas behind many attack paths!

- Different side-channels, but often same exploitation techniques
  - just setup and measurement different
  - attack strategies and analysis techniques very similar

Passive Attacks: Timing

Implementation of a PIN Check

```
r = strcmp(secret_pin, typed_pin);
if(r==0){
  /* grant access */
  access_secret_data();
} else {
  /* deny access */
  incorrect_password();
}
```

← let's have a closer look

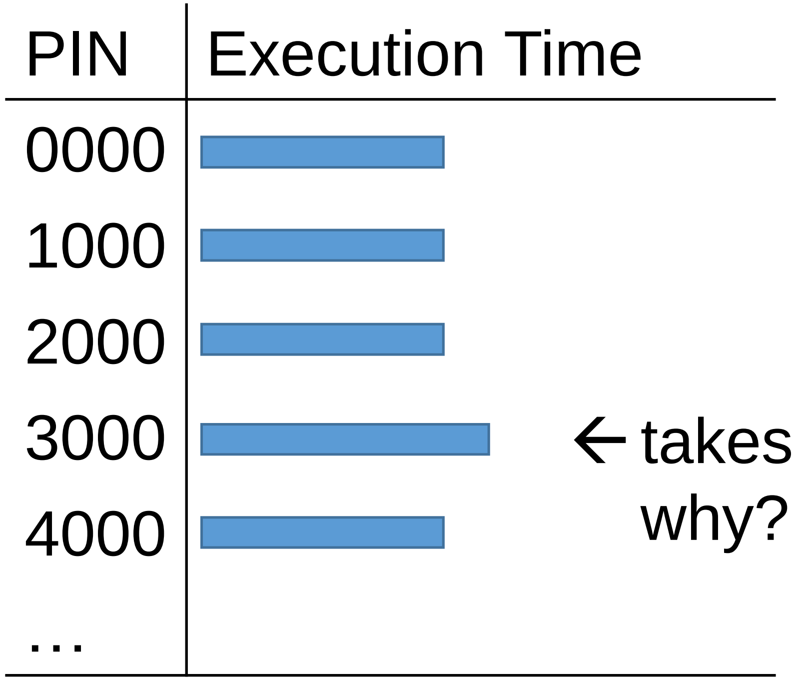
glibcs strcmp

```
int strcmp (const char *p1, const char *p2) {
  const unsigned char *s1 = (const unsigned char *) p1;
  const unsigned char *s2 = (const unsigned char *) p2;
  unsigned char c1, c2;
  do {
    c1 = (unsigned char) *s1++;
    c2 = (unsigned char) *s2++;
    if (c1 == '\0')
      return c1 - c2;
  } while (c1 == c2);
  return c1 - c2;
}
```

← strcmp terminates after first difference!

A Timing Attack on the PIN Check

- Try all possibilities for first digit and measure time



← takes slightly longer
why? 3 is correct, so the second digit is checked

Full Attack on the PIN Check

- Try all possibilities for first digit and measure time
  - set other digits to some fixed value
  - pick digit with highest execution time
- Repeat for other digits
  - set first digits to recovered values

- Comparison (4 decimal digits)
  - brute force: 10^4 = 10000 combinations
  - timing attack: 10x4 = 40 combinations

Divide-and-Conquer in Side-Channel Attacks

- Separate large secrets into smaller pieces (subkeys)
  - depending on attack target, pieces might be: bits, decimal digits, bytes, 32-bit words, …
  - but always: small enough to try all possibilities (2, 10, 256, 2^{32}, …)

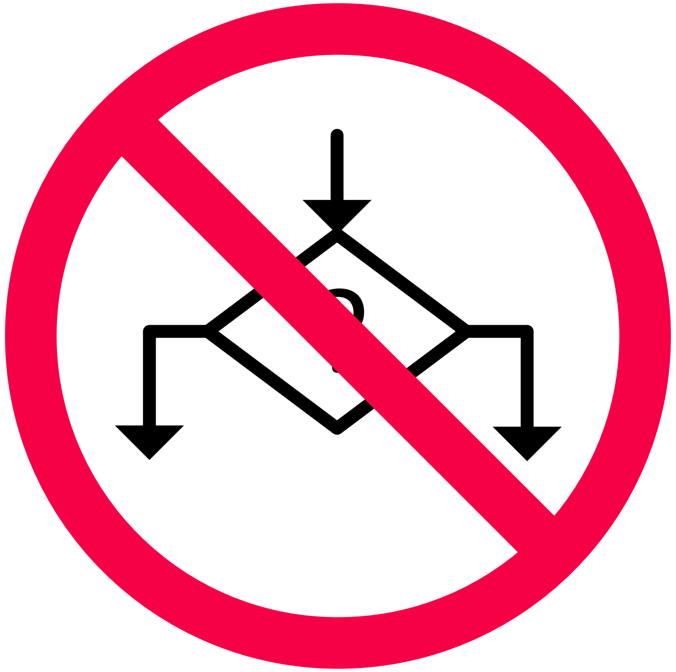
- Recover subkeys individually with side channels
  - enumerate possibilities (test all of them, not necessarily as input to device)
  - pick value that best fits to side-channel information

- Drastic reduction of attack complexity (compared to brute force)
  - PIN check: $10^4 \rightarrow 10 \times 4$
  - AES with 128-bit key (16 bytes): $2^{128} = (2^8)^{16} \rightarrow (2^8) \times 16$ how? later!

Protection against Timing Attacks

- No branching on secret data: **Constant Runtime & Control Flow**
  - always exactly same instruction sequence, but different data
  - branches depending on entire secret of course OK, such as:
    ```
    if(r==0) access_secret_data();
    ```

- Mind your hardware!
  - table lookups depending on secret data
    → cache attacks! hardware inserts „implicit“ branch!



Protecting the PIN Check

```
int pincmp (const char *p1, const char *p2, int pinlen) {
  char diff = 0;
  for(int i = 0; i < pinlen; i++){
    char c1 = *p1++;
    char c2 = *p2++;
    diff |= (c1 ^ c2);
  }
  return (diff != 0);
}
```

1. Always run through all digits
2. Constant-time comparison using bitwise logic operations

Timing Attacks on Cryptographic Implementations

- Also many timing attacks for cryptographic code
  - there do exist constant-time algorithms (not much slower)
  - but…

- Threat still **often overlooked / ignored**
  - no protection at all: “outside of threat model”
  - or not all parts of algorithm properly protected

- Two very recent timing attacks on ECDSA running on **certified devices**
  - Minerva attack on smart cards (a month ago)
  - TPM-Fail: Timing Attack on TPM devices (this week)

Are we secure?

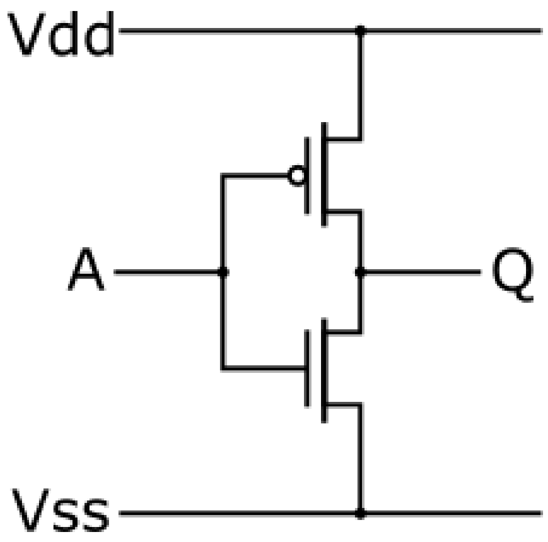
- Timing: maybe
- Other channels: no!

Passive Attacks: Power Analysis

CMOS Circuits

- Vast majority of today’s digital circuits use CMOS
  - CMOS = “Complementary metal oxide semiconductor”
- CMOS offers two nice properties
  - high noise immunity
  - low power consumption
- A main reason for low power consumption:

Only switching draws power (*)



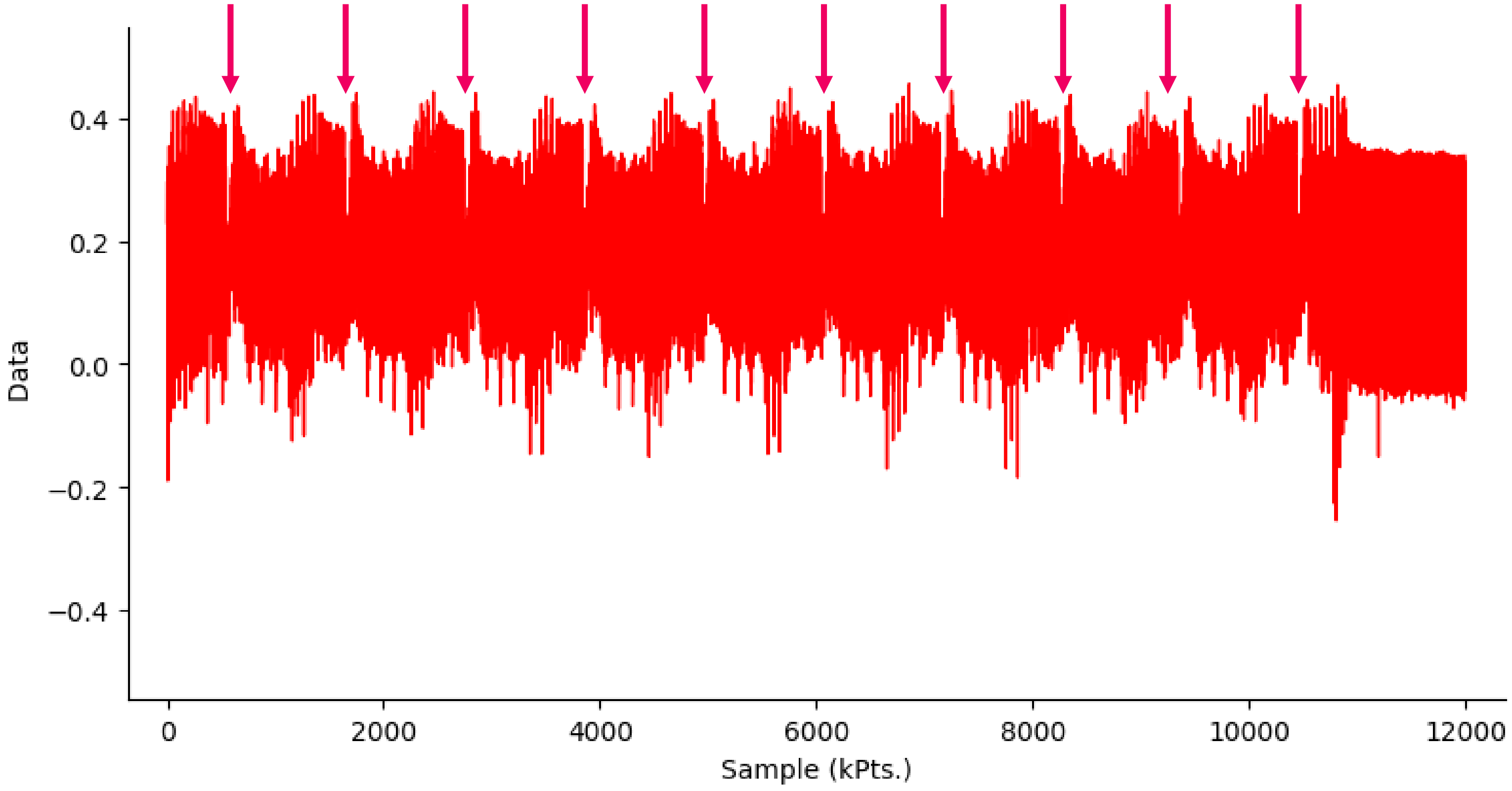
CMOS Circuits – Power Consumption

- Different instructions and different data → different switching
- CMOS instantaneous power consumption depends on
  - the instruction that is executed
  - the data that is being processed

- We measure instantaneous power consumption during operation
  - sampling rate up to gigasamples (10^9 values per second)
  - we call a measure power-consumption curve a **trace**

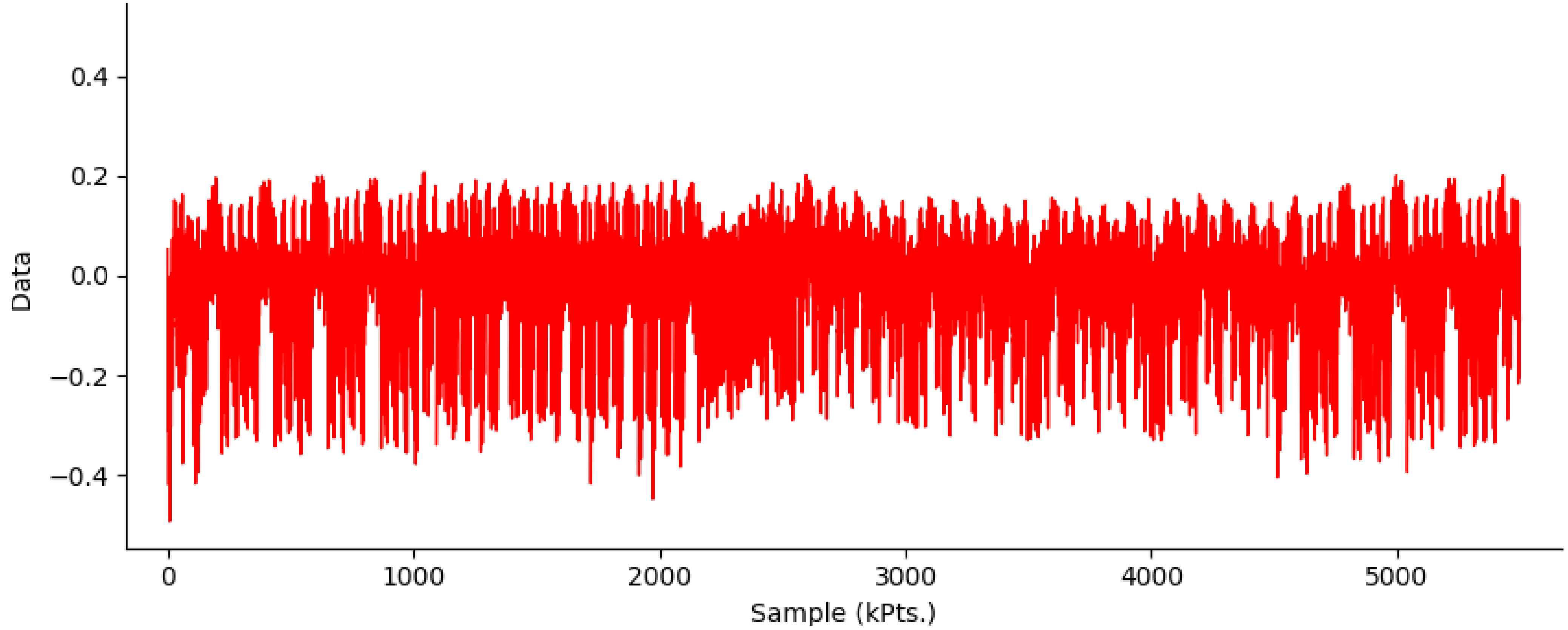
- First signal-processing step: „looking at it“

Power Consumption – An Example Trace



- 10 rounds of AES

Power Consumption – Zoom on Single Round



Typical Learning from a Visual Inspection

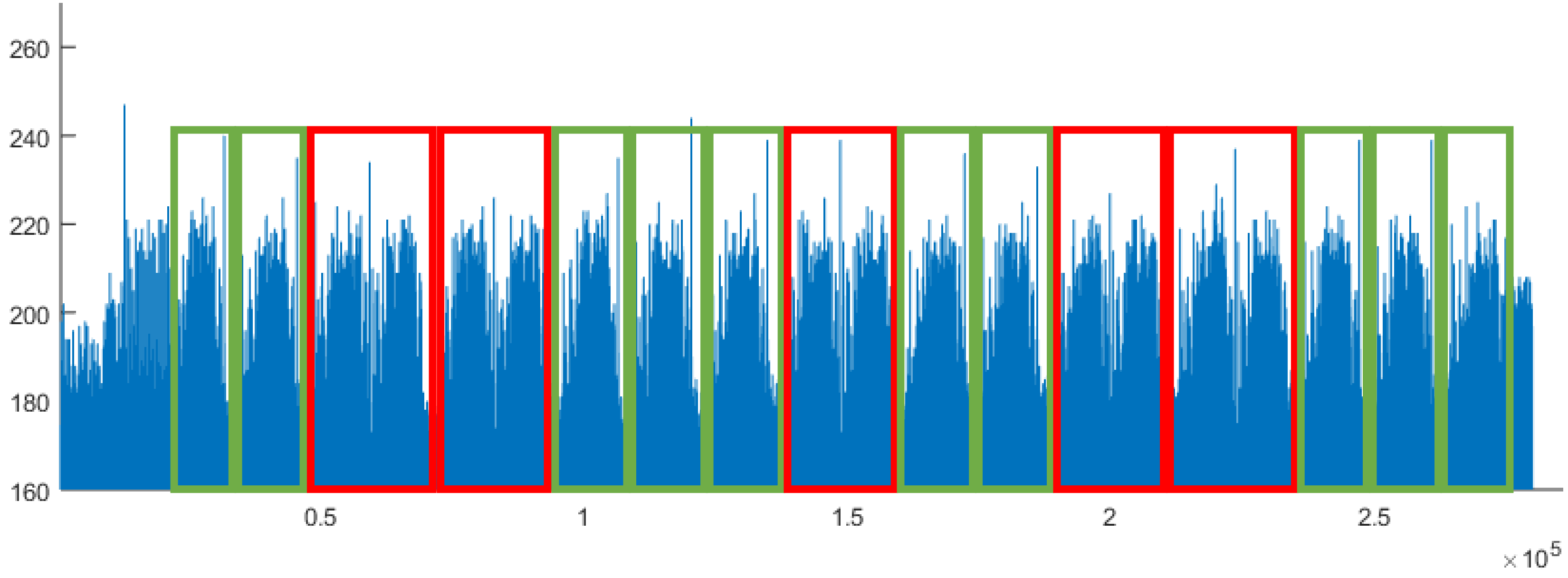
Visual inspection: learn about operations / instructions

- Detection of repeated patterns and variations of patterns
  - detection of loop length, repeated operations, taken branches
  - …learn control flow / instruction sequence

- Detection of gross changes in the power consumption profile
  - memory accesses in general (especially EEPROM or flash programming)
  - access to peripherals (e.g. coprocessors, IO)
  - …

Can we already exploit that?

Our First Power-Analysis Attack



Power consumption of device performing RSA decryption: $m = c^d \bmod n$

we can identify patterns, but need a some more information for an attack

Excursion: Efficient Implementation of Modular Exp.

- RSA decryption: $m = c^d \bmod n$
  - $n \geq 2048$ bits
- Efficient implementation? Compute $((c^d) \bmod n)$?
  - c,d are also ≈ 2048 bits → c^d has more than 2^{2048} bits
  - we have to find something better…
- 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 Exp.

- Bit indizes: d_i = i-th bit of d (d_0 is LSB)
- Recursive decomposition of exponentiation
  - we can write: $d = 2\lfloor d/2 \rfloor + (d \bmod 2) = 2(d >> 1) + d_0$
  - in the exponent: $c^d = (c^{\lfloor d/2 \rfloor})^2 \cdot c^{d_0}$
  - $\lfloor d/2 \rfloor$ is still too large, so we 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
  - we start at $\lfloor d/(2^x) \rfloor = 1$ and make our way up

Excursion: Efficient Implementation of Modular Exp.

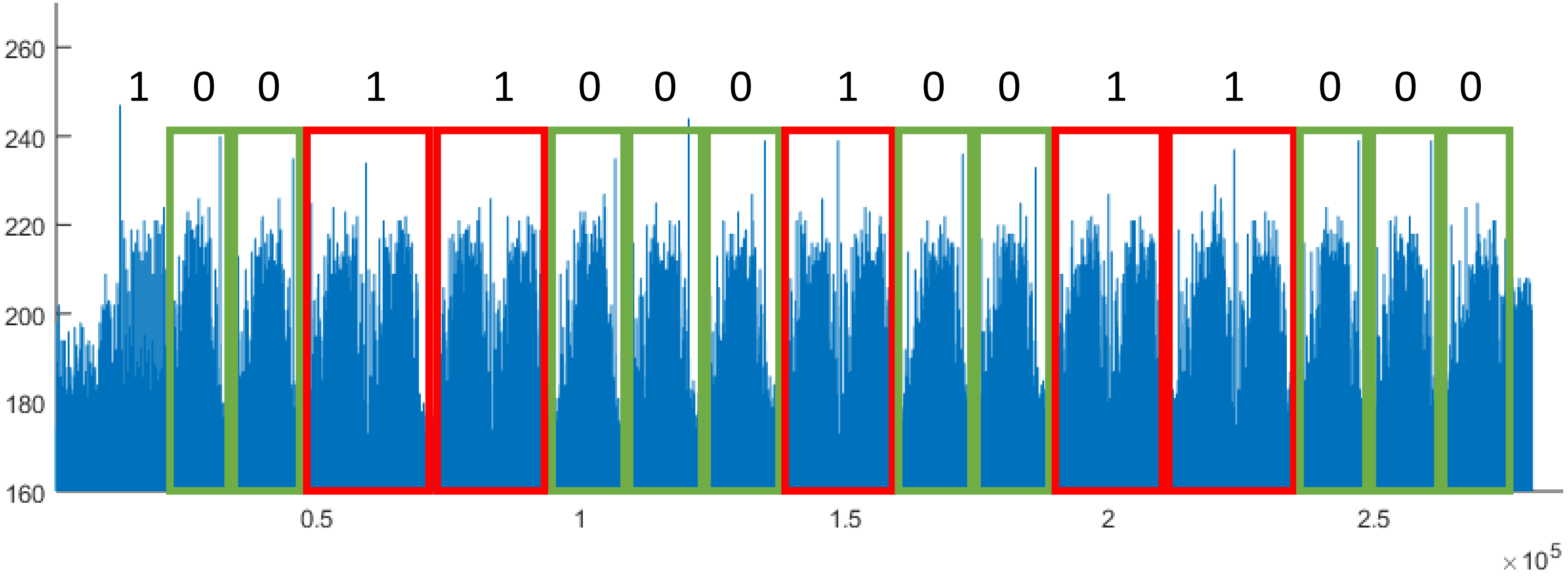
- Left-to-right square-and-multiply exponentiation

```
m = 1                        //init
for i = 2047 downto 0        //scan bits from MSB to LSB
   m = m^2 mod n             //squaring: c^x = (c^⌊x/2⌋)^2 · c^x0
   if d_i = 1 then           //if bit is set (x_0 = 0 → c^x0 = 1, mult. can be skipped)
      m = m · c mod n        //  then multiply: c^x = (c^⌊x/2⌋)^2 · c^x0
```

- Example: $d = 26 = 11010_b$
  - $c^{26} = ((((1^2 \cdot c)^2 \cdot c)^2)^2 \cdot c)^2$

- …but what does that mean for our side-channel attack?

Our First Power-Analysis Attack – Key Recovery

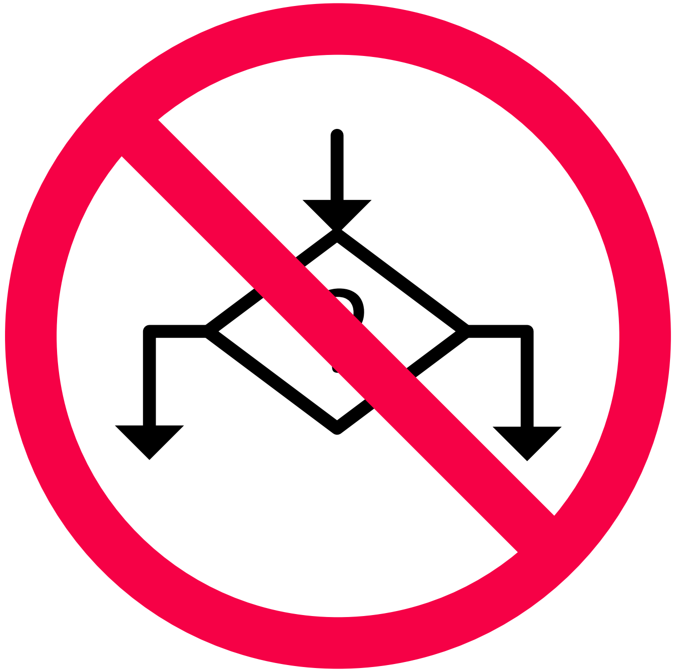


Key recovery by just „looking at the thing“

Countermeasures

- No branching on secret data: **Constant Runtime & Control Flow**
  - always exactly same instruction sequence, but different data

- There do exist more secure alternatives
  - exponentiation algorithms that run in constant time
  - constant time modular reduction
  - …

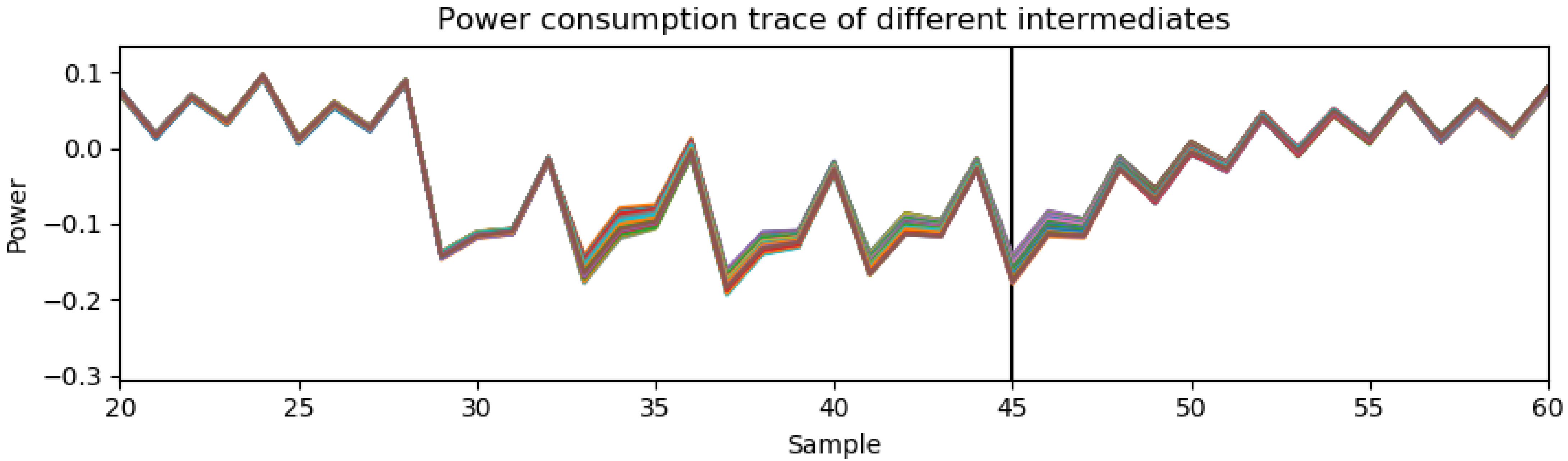


…two attacks, both defeated by constant time?
Is this the answer to all our problems?

CMOS Circuits – Power Consumption

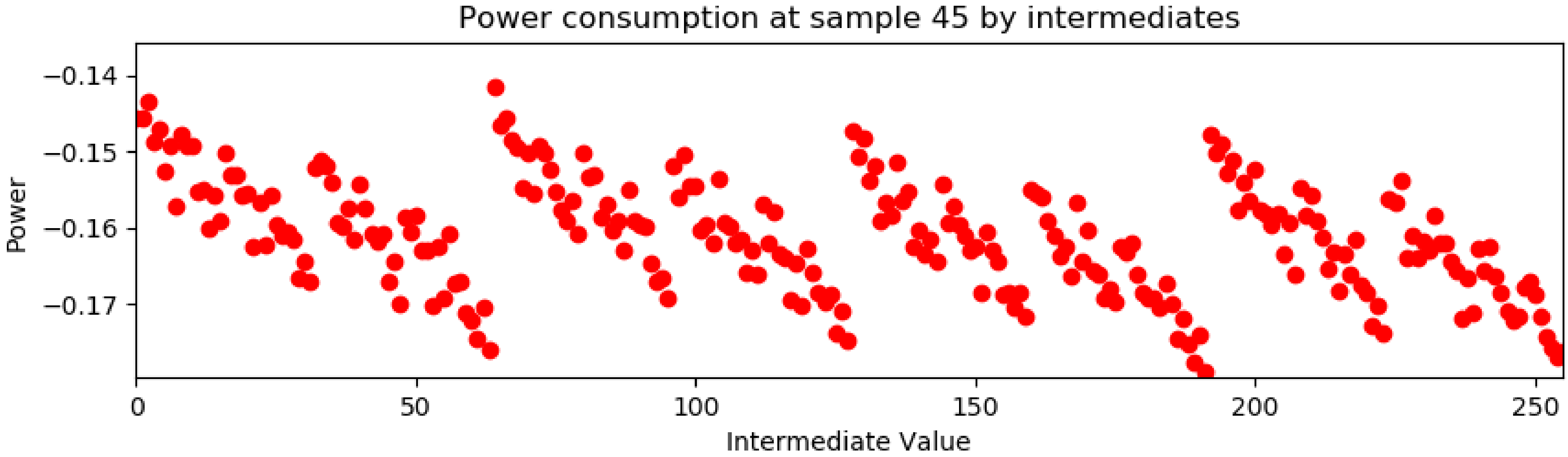
- Different instructions and different data → different switching
- CMOS instantaneous power consumption depends on
  - the instruction that is executed
  - the data that is being processed
- We measure instantaneous power consumption during operation
  - sampling rate up to gigasamples (10^9 values per second)
- First signal-processing step: „looking at it“

CMOS Power Consumption: Data Dependency



- Averaged power consumption of a load instruction for values {0, 255}
- High variance in point 45…

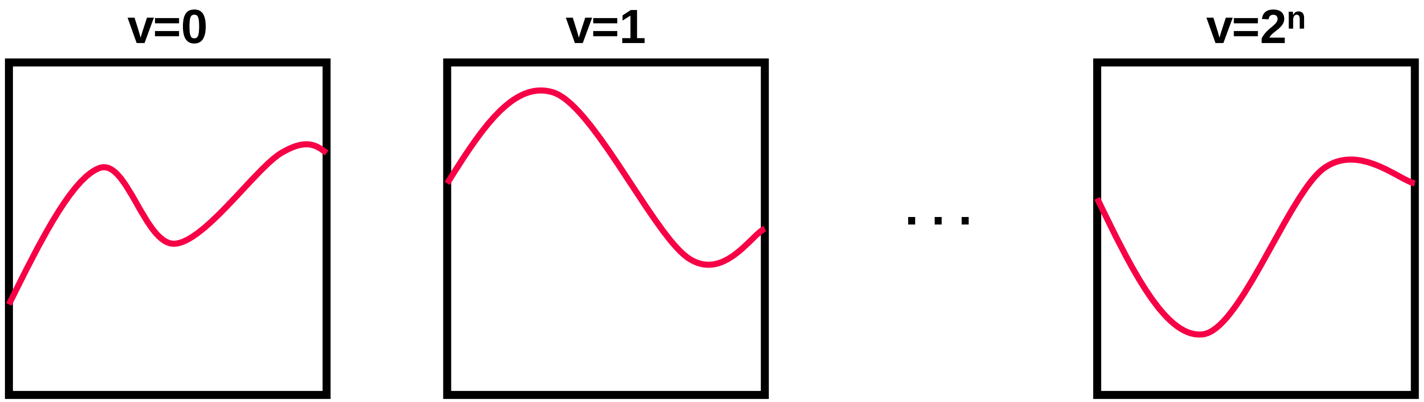
Data Dependency – A Closer Look



- different intermediates cause different power consumption
- use recorded values for an attack → Template Attack

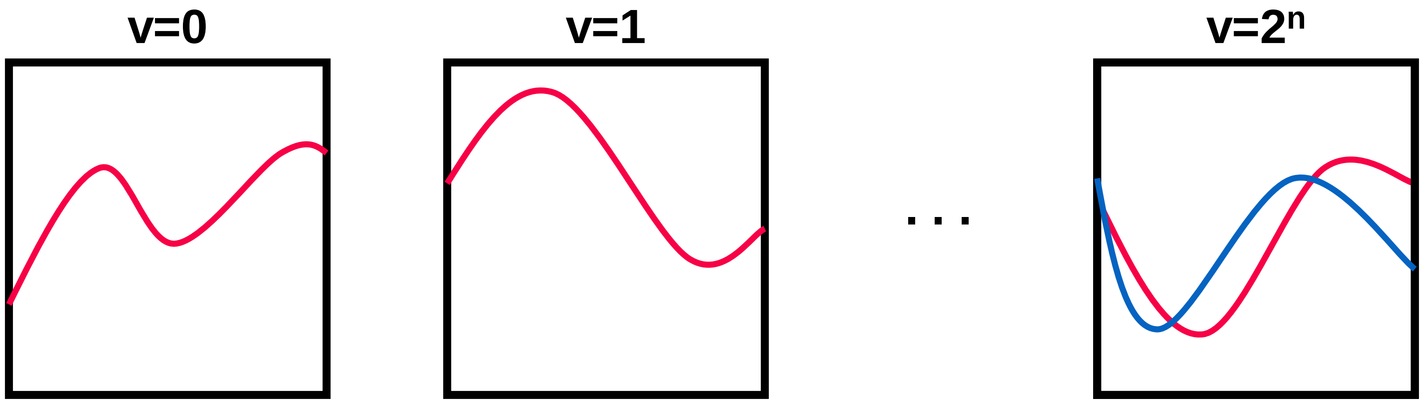
Basic Steps of a Template Attack

1. Characterization phase
   - profile power consumption for each possible value of intermediate v
   - record traces with all inputs known, group according to v
   - we call profile a „template“



Basic Steps of a Template Attack

2. Attack phase
   - compare (match) measured traces to all templates
   - use v which best fits, process probabilities…

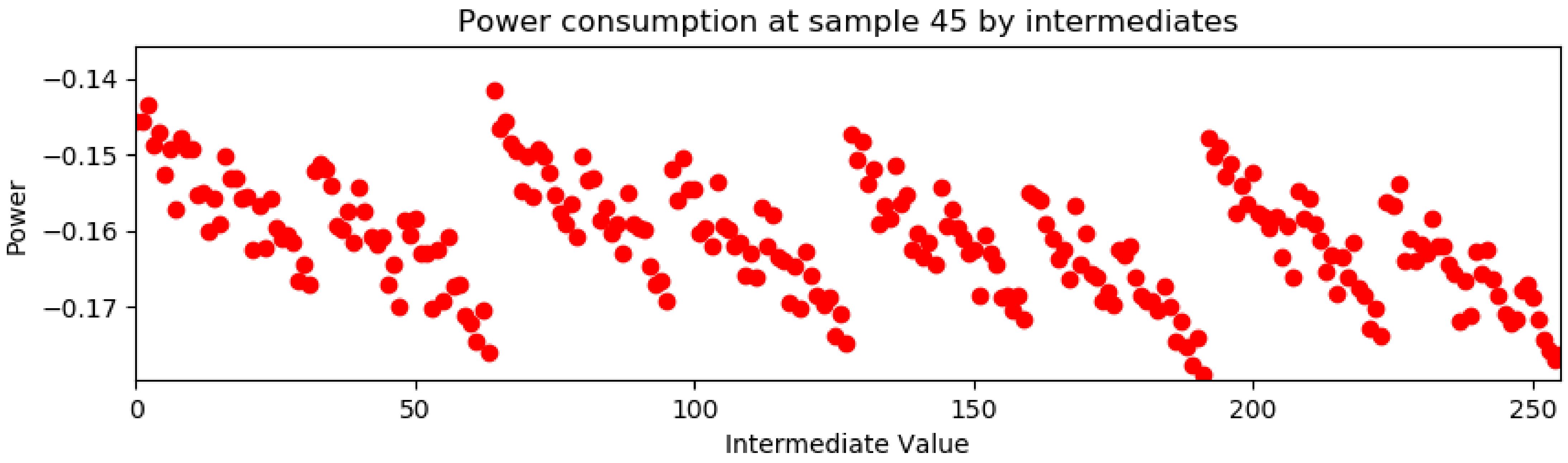


Template Attacks

- Very powerful attack
  - sometimes key-recovery with single observation
  - sometimes also only option (just have a single trace)

- Downside: many prerequisites and detailed knowledge on device needed
  - When is secret processed? What is the concrete algorithm?
  - You need access to an identical device where you can control **all** inputs (but even then not trivial)

Data Dependency – Another Look



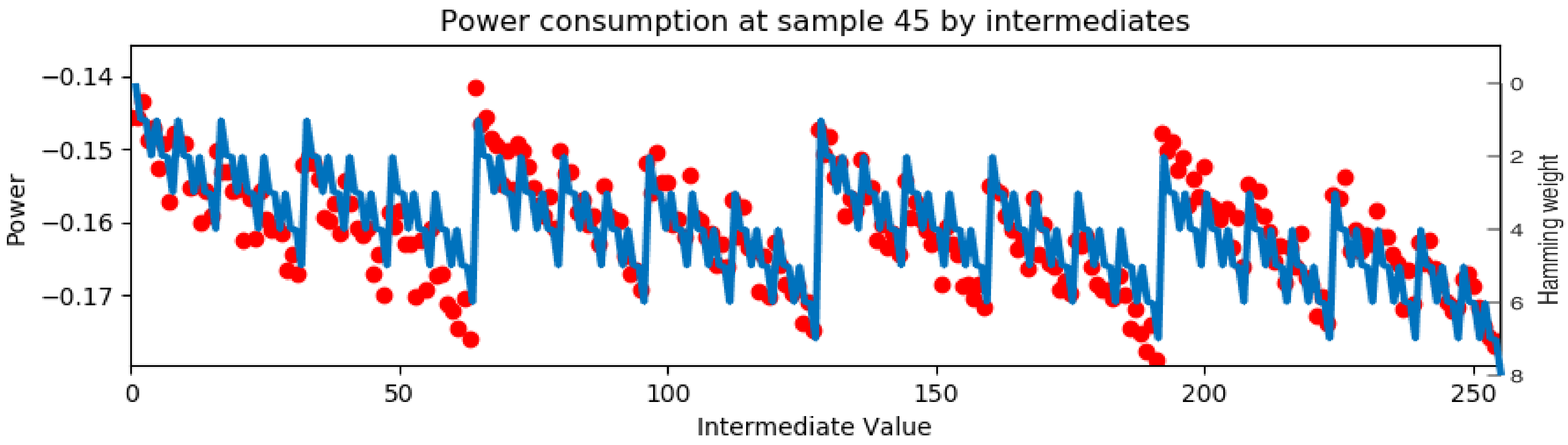
- There is clearly some pattern…
- We can model the power consumption

Hamming Weight Power Model

- Recall: power depends on switching!

- Microcontrollers perform a „precharging“ for memory accesses
  - simple analogy: all bits are set to 0, then new value comes in
  - each 1 bit draws power → power is proportional to number of set bits
  - Number of 1 bits = **Hamming weight HW(a)**

Indeed…



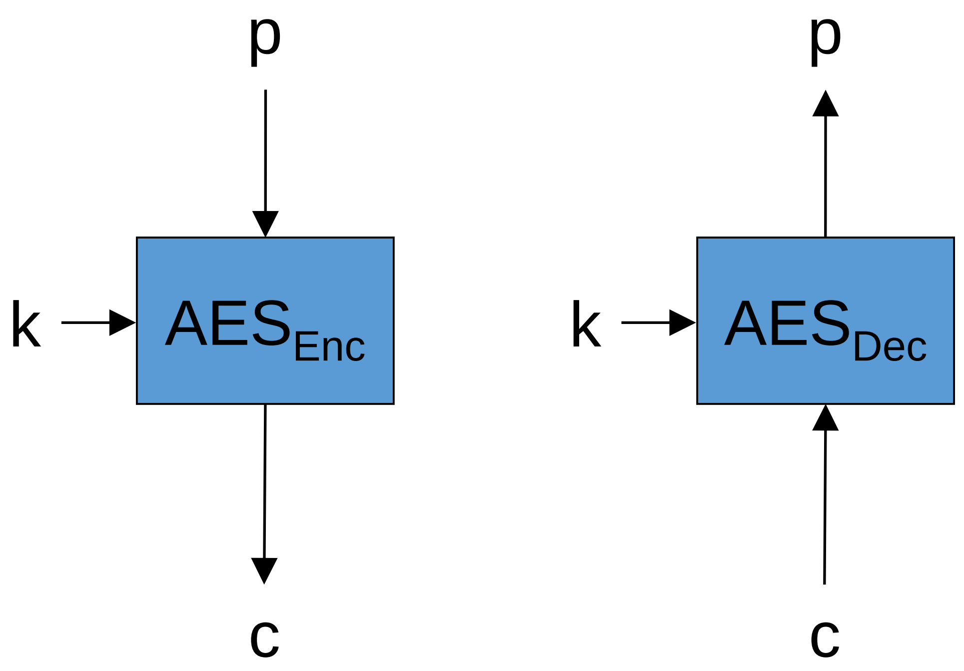
- Many devices have similar power behavior
- We can reuse power models for a large number of devices

Attack without detailed knowledge of device and concrete implementation!

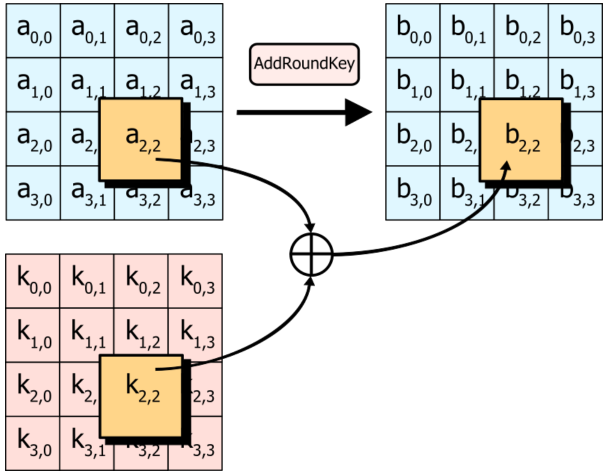
Quick Refresher: AES

- Advanced Encryption Standard
- Block cipher with block size: 128 bit
- Key size: 128/192/256 bit

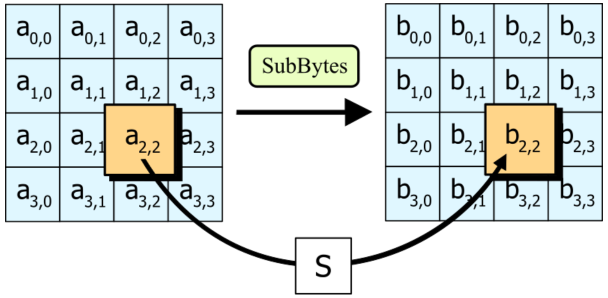
- Byte oriented
  - State is 4x4 matrix of bytes
- 10 rounds of 4 steps



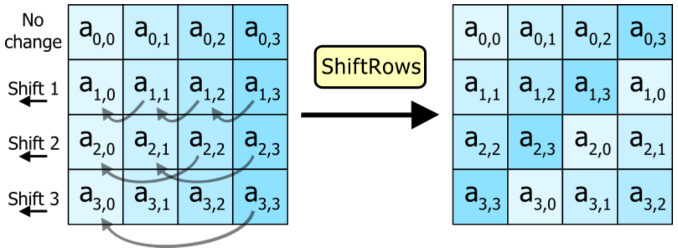
Quick Refresher: AES



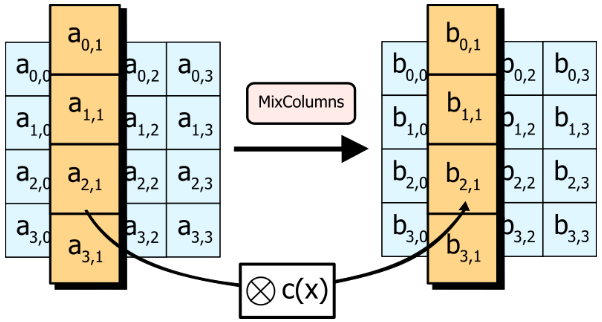
1. AddRoundKey



2. SubBytes



3. ShiftRows

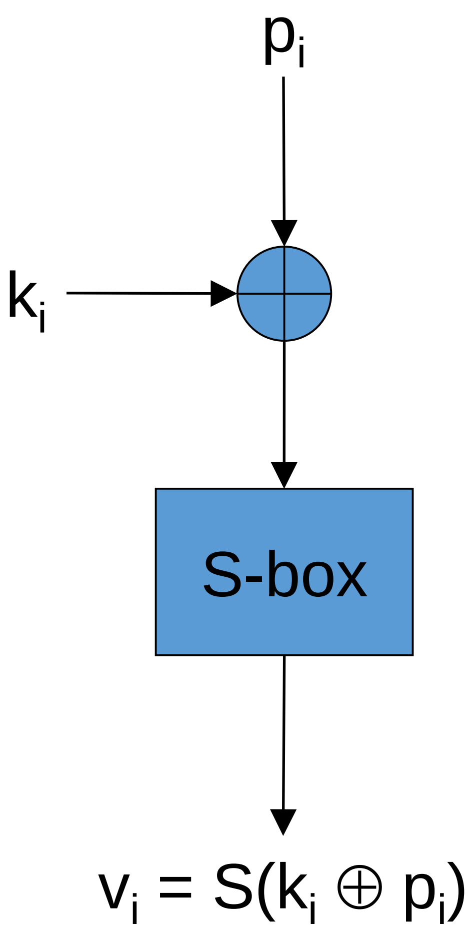


4. MixColumns

http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

Quick Refresher: AES

- First round
  - roundkey = key
- Other roundkeys: key schedule
  - key schedule is invertible



Differential Power Analyis (DPA)

A concrete example with AES on a microcontroller

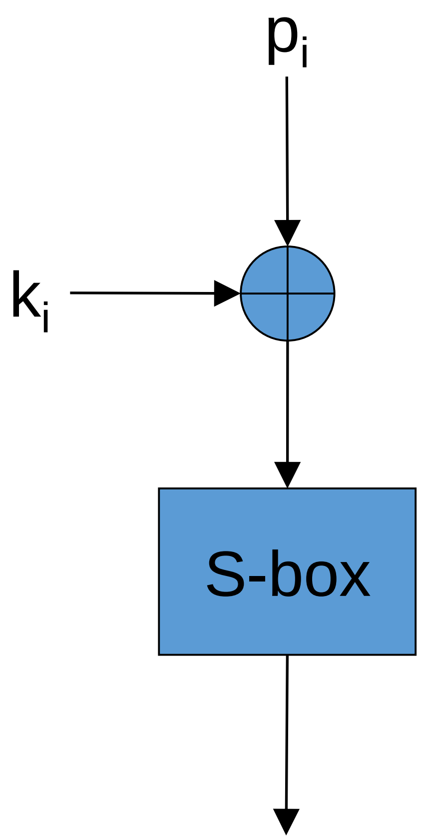
Main Steps

1. Select an intermediate value
   - depends on small number of key bits (subkey)
2. Query device and measure power
3. Enumerate all possible subkey values
   - 2^8 key hypothesis
   - for each PT/CT: 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

Step 1: Select an Intermediate Value

- Should depend on:
  - small number of key bits (enumerable, e.g. 8)
  - known and varying data (pt/ct)

- Common choice:
  - SubBytes output of first round (1 byte)

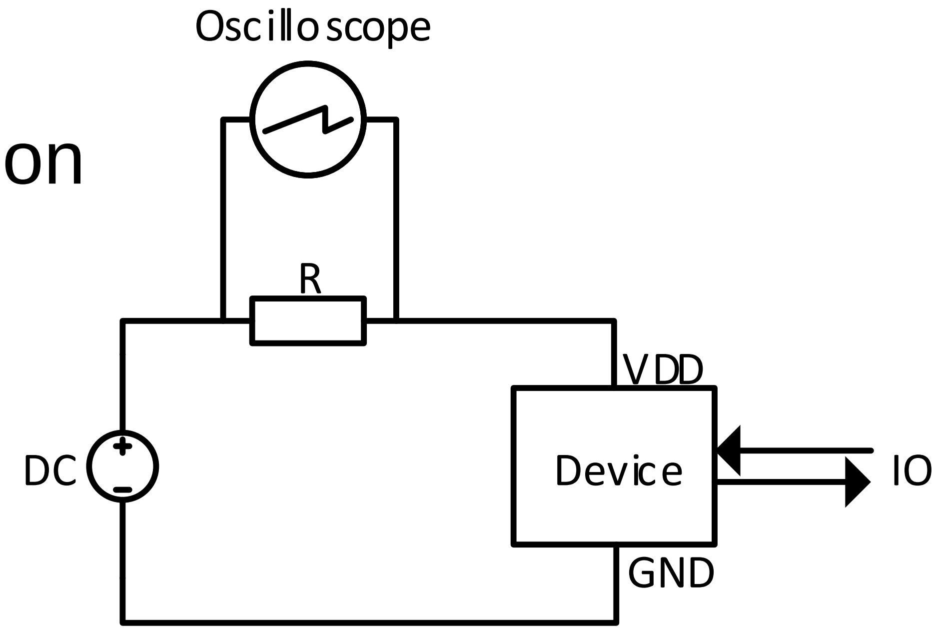


$v_i = S(k_i \oplus p_i)$

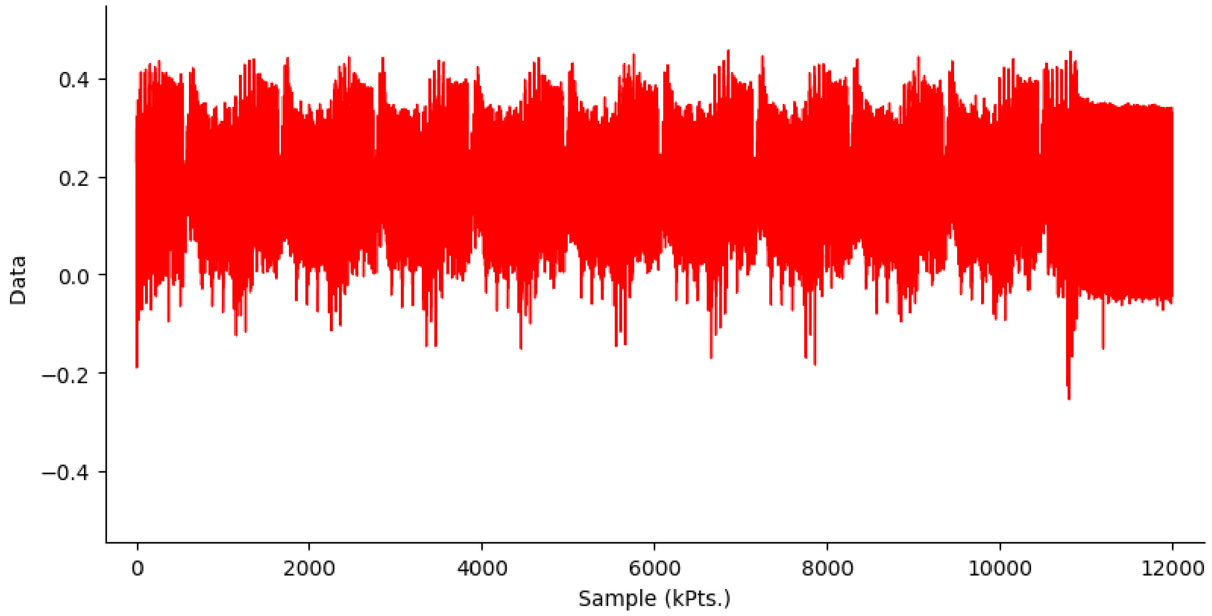
Step 2: Measure Power Consumption

- Query device
- Gather IO (plaintext/ciphertext)
  - for next slides: assume we get plaintext of encryption (attack also works for ciphertext, decryption, etc)

- Measure instantaneous power consumption
  - measurement must include time where v is processed

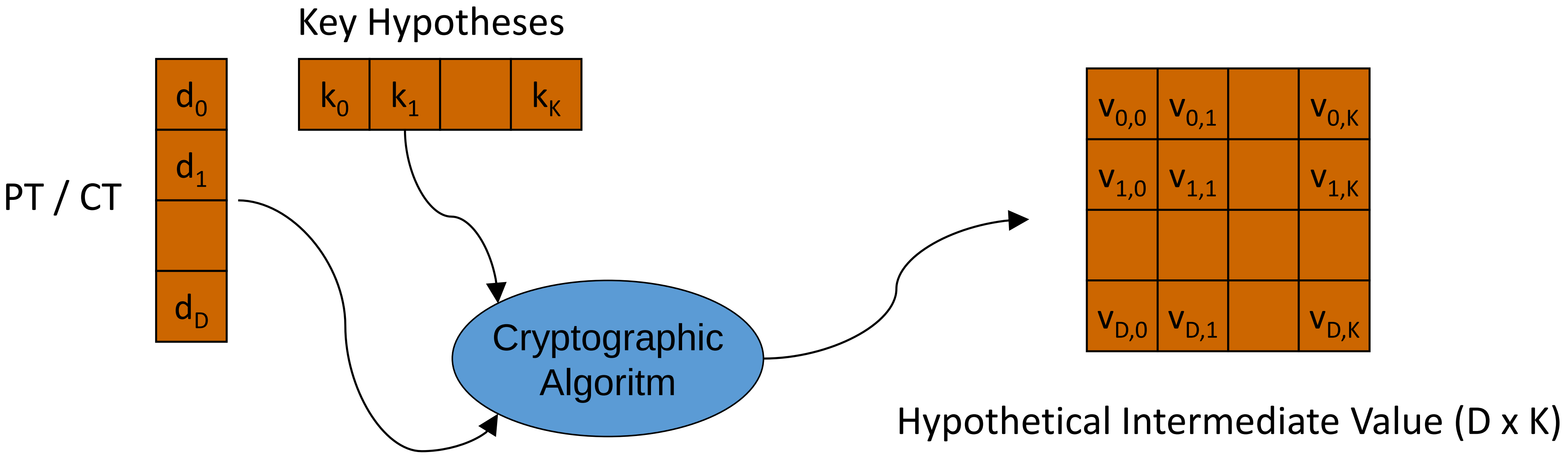


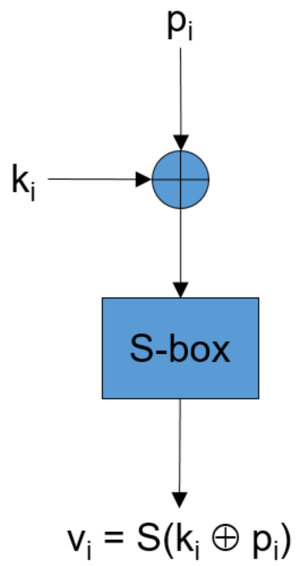
Step 2: Measure Power Consumption



- How to know what part is measured?
  - visual inspection, trial&error, experience,…

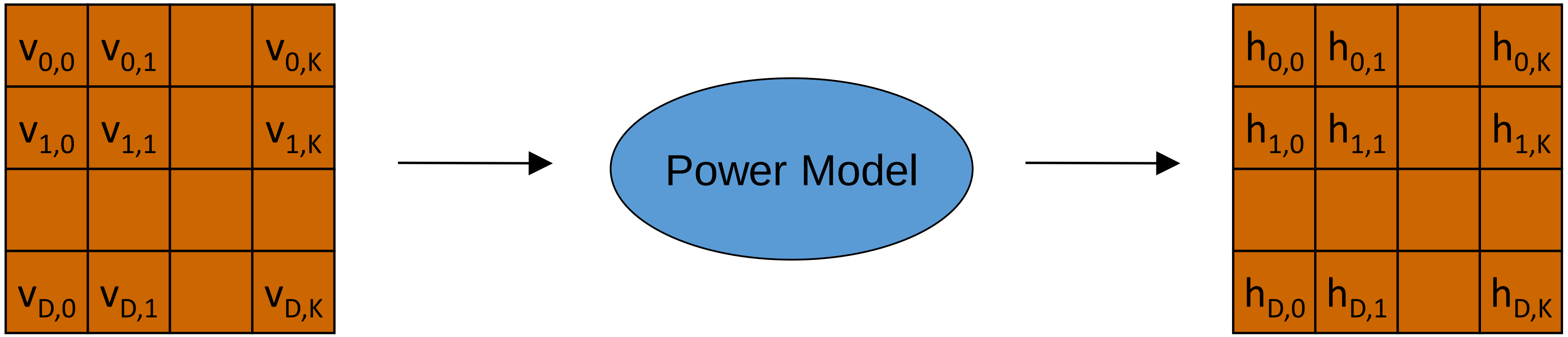
Step 3: Enumerate Subkeys and Intermediate Values





- D observations (measurements)
- K hypotheses ($K = 2^8$)
- D x K hypothetical intermediate values

Step 4: Predict Power Consumption

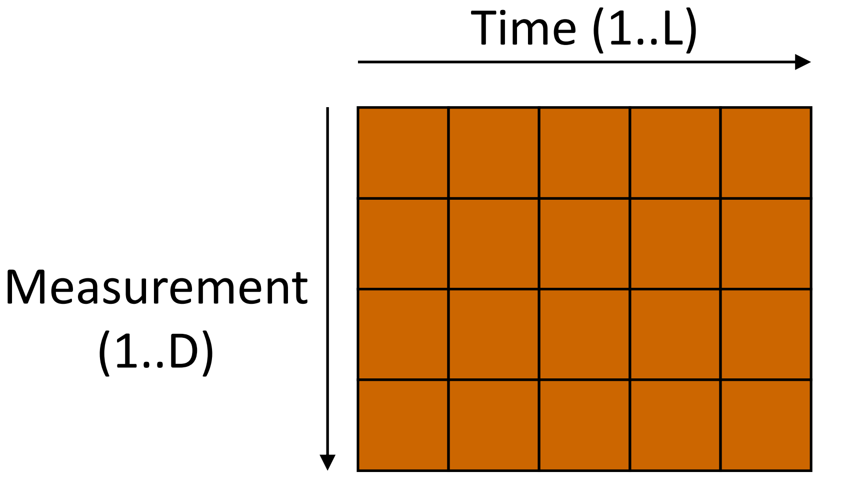


Hypothetical Intermediate Value

Hypothetical Power Consumption

- We attack a microcontroller, so we use Hamming weight (# of set bits)

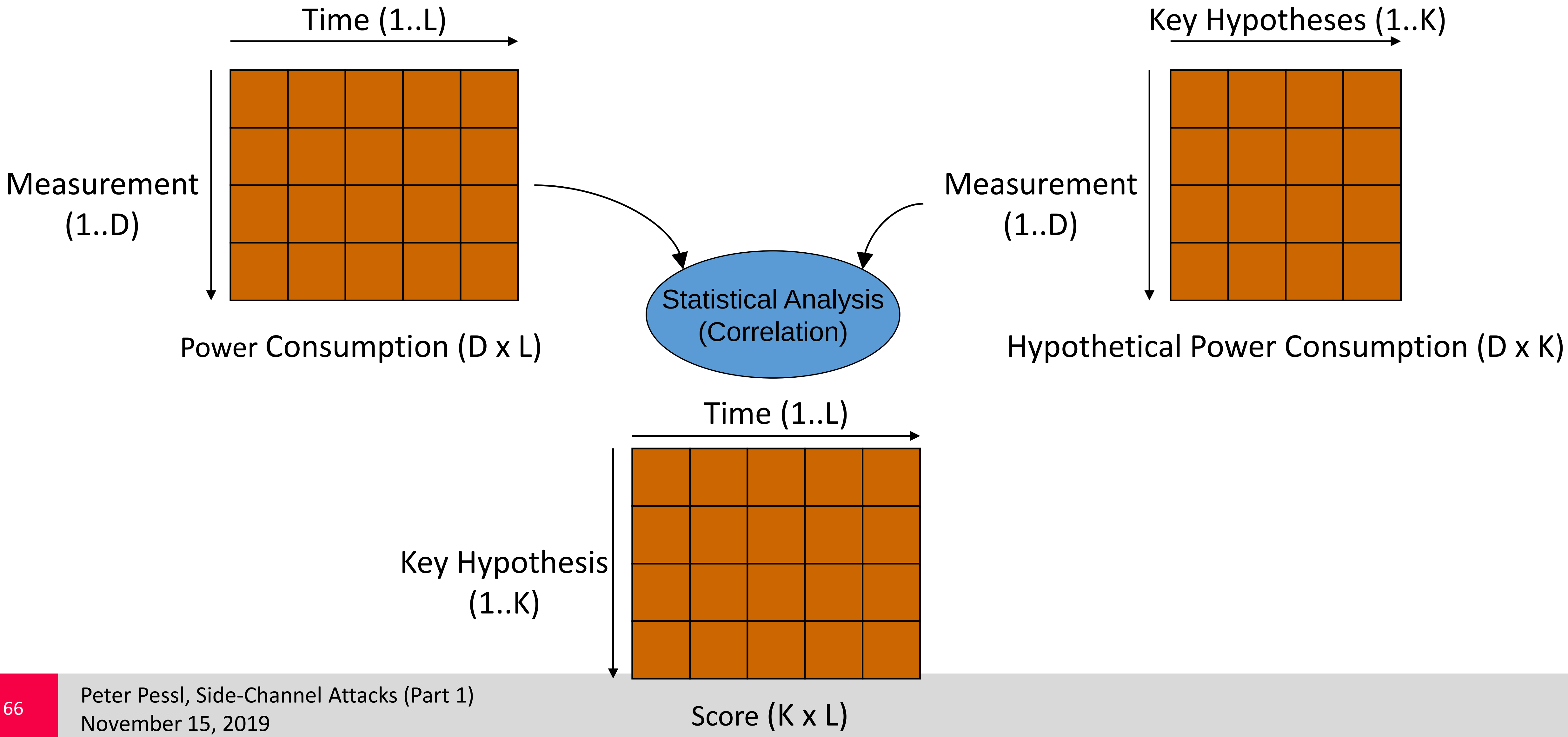
Step 5: Comparison



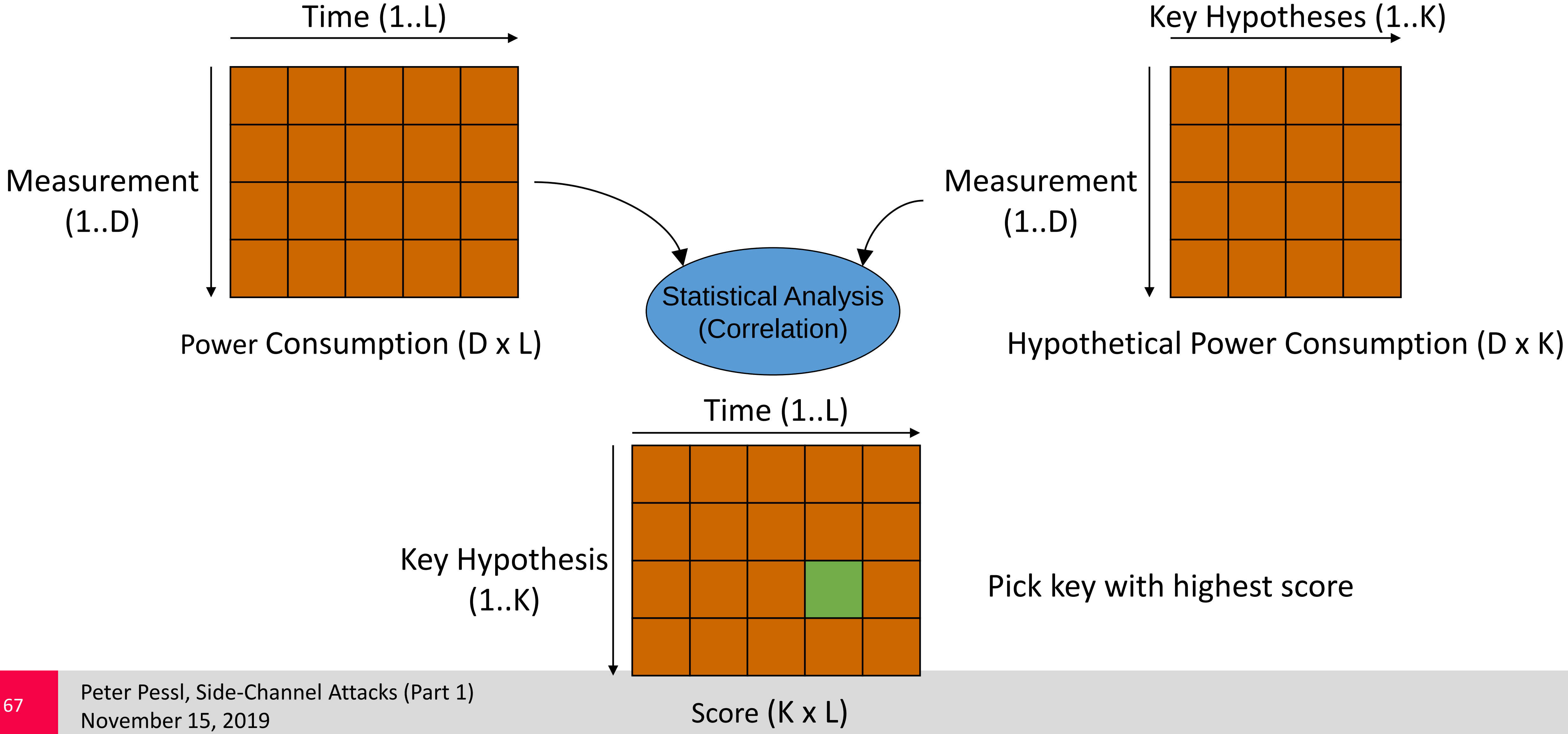
Power Consumption (D x L)

- Trace matrix
  - for each measurement we get L samples
- Problems
  - L can be large
  - we have no idea when targeted intermediate is processed
- Simply test all locations!

Step 5: Comparison

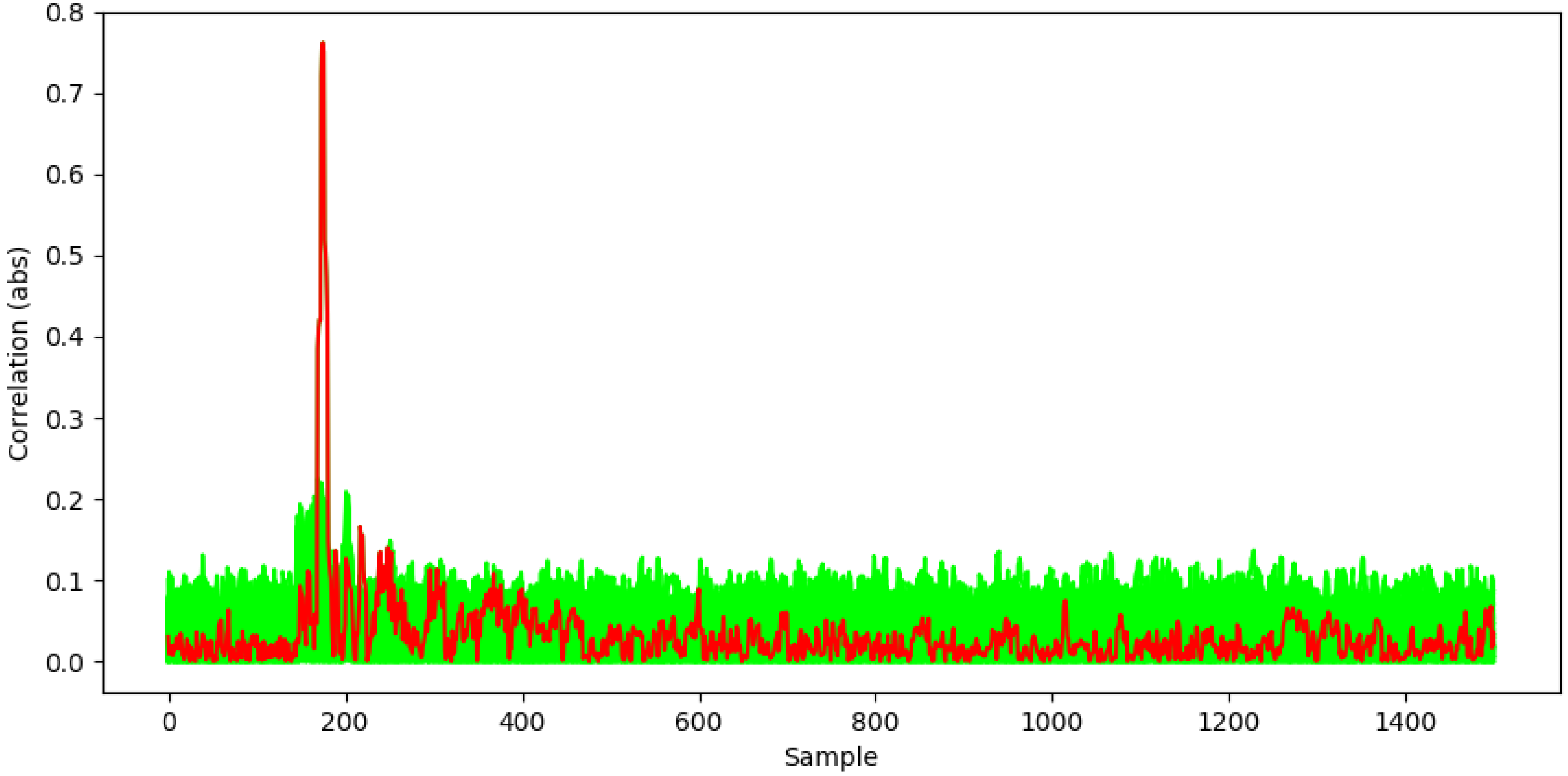


Step 5: Comparison



Pick key with highest score

Exemplary Outcome



- red: correct key
- green: other 255 keys

DPA: Recap

- Requires little assumptions…
  - on device (power models)
  - on concrete implementation (when does it leak?)
  - yet still effective
- But there are also downsides
  - simplifications that affect performance
  - not applicable to single traces
  - …

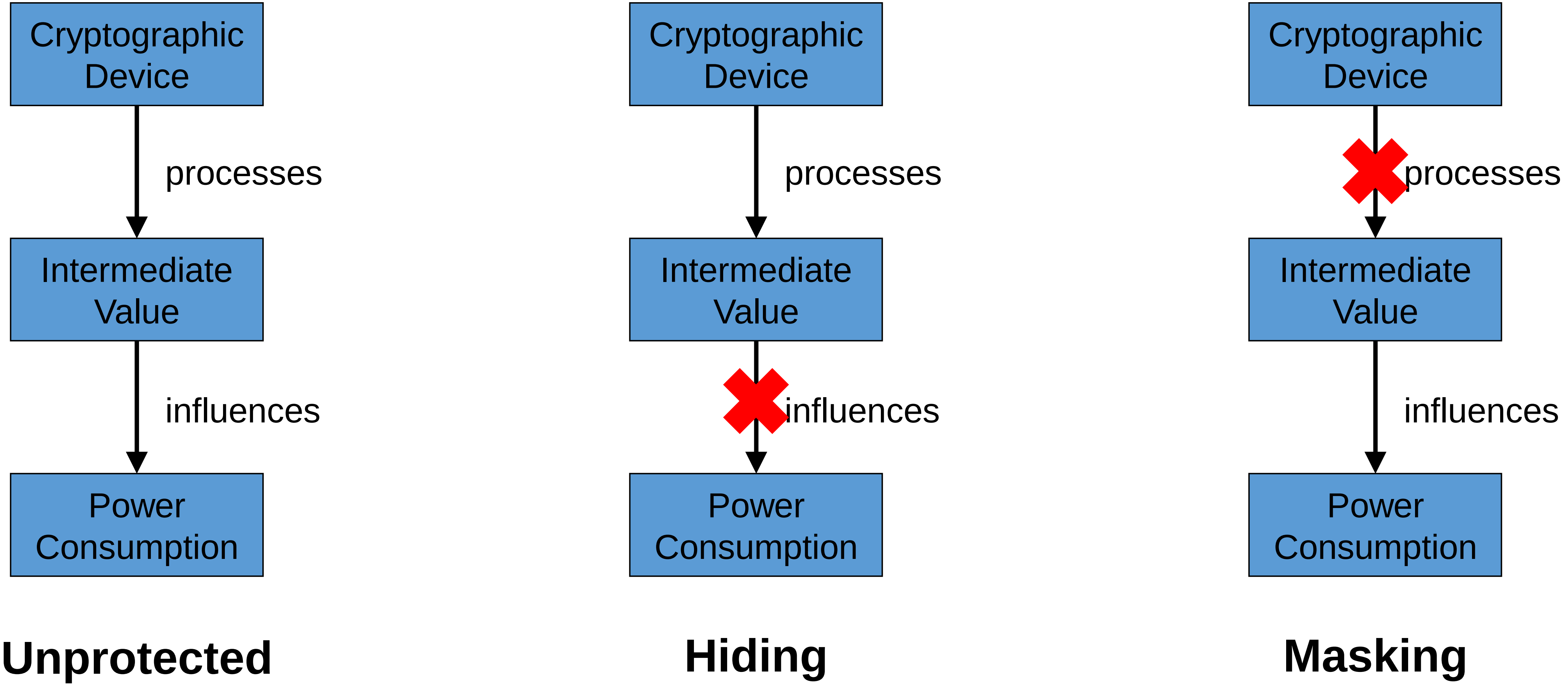
Power Analysis: Countermeasures

Key Updates

- Attacks require some number of traces
  - side channels are noisy
  - statistics requires enough data
- Simplest countermeasure: restrict number of operations per key
  - only a certain number of encryptions, then change key
- Problems
  - key update anything but trivial
  - attacking unprotected implementations very easy (less than 50 traces)

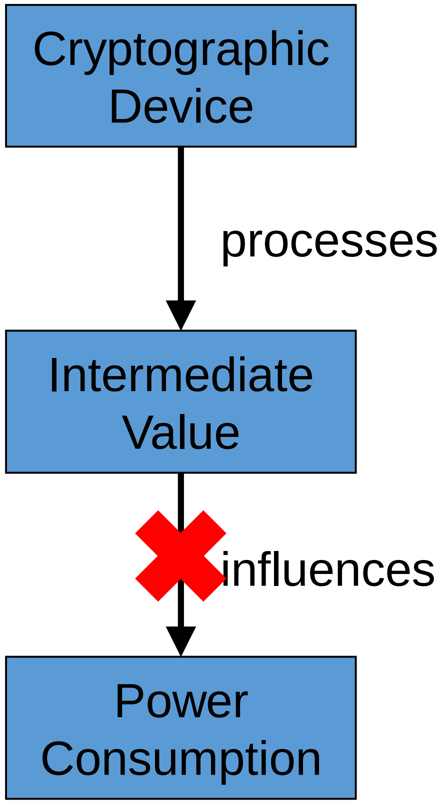
Need to protect AES itself!

Scenarios


Cryptographic Device
processes
Intermediate Value
influences
Power Consumption
Unprotected
Cryptographic Device
processes
Intermediate Value
influences
Power Consumption
Hiding
Cryptographic Device
processes
Intermediate Value
influences
Power Consumption
Masking

Hiding

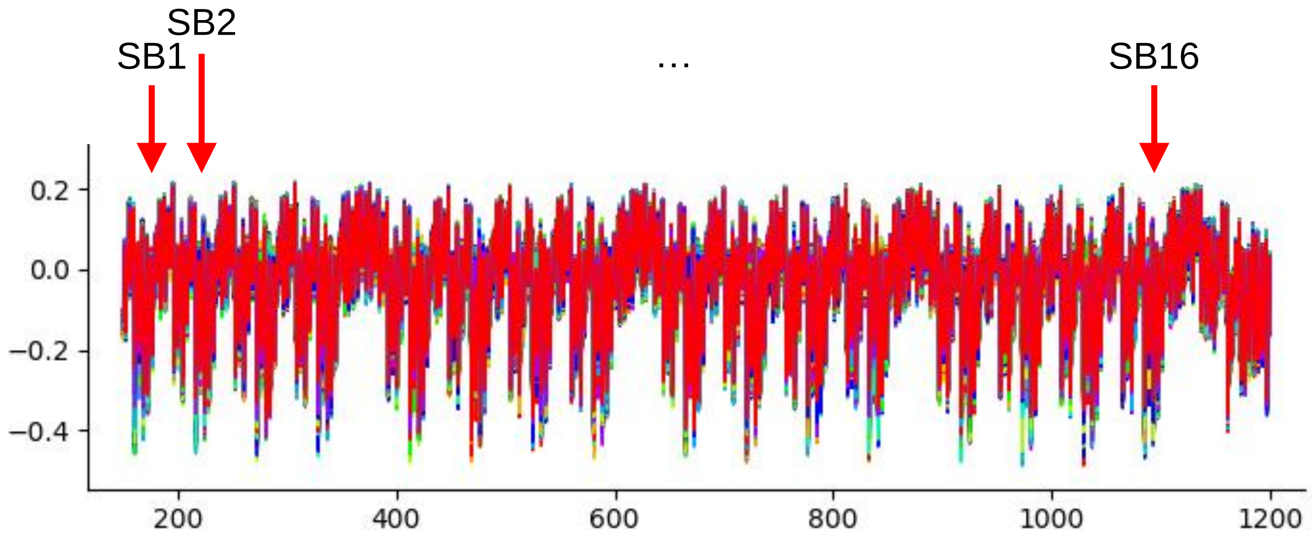
- Hide (reduce) the data-dependent power consumption



Example: Hiding in Time Dimension

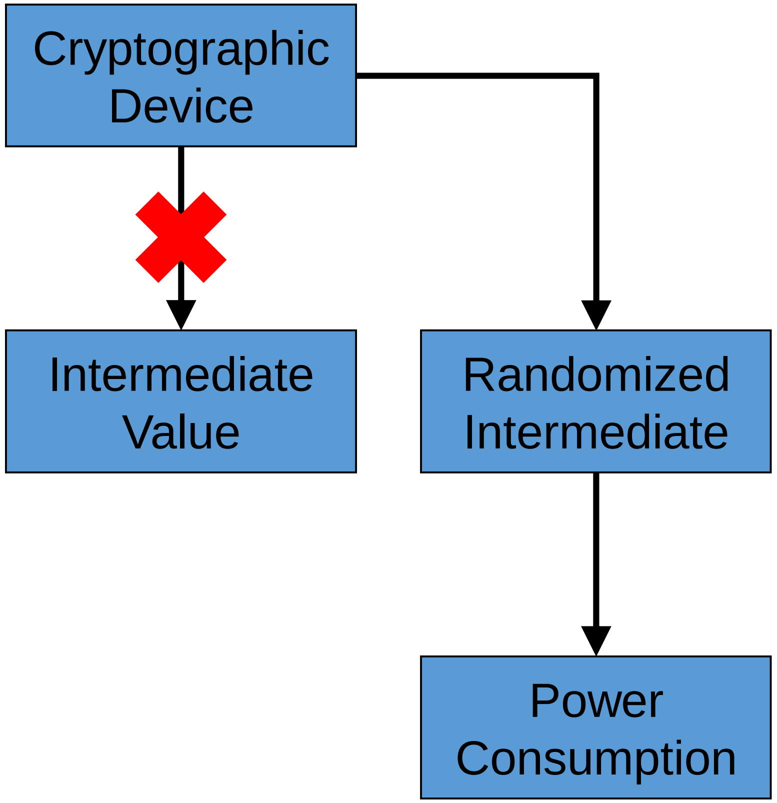
- Assumption of DPA:
  same operation at same instant in time

- Break assumption!
  - e.g., randomly shuffle order of operations



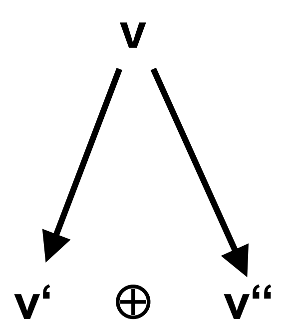
Masking

- Operate on randomized intermediates
  - side-channel information on randomized intermediate does not help attacker
  - but still require correct algorithm output



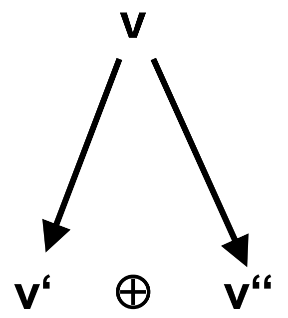
Masking - Idea

- XOR-Sharing (Boolean masking)
  - split v = v‘ ⊕ v’’
  - initial sharing (start of algorithm):
    1) sample random v’’
    2) compute v‘ = v ⊕ v’’
  - during computation:
    process v’ and v’’ individually
- Observation:
  v’ and v’’ are (mutually) independent of v
  power consumption of just one value doesn’t reveal anything!

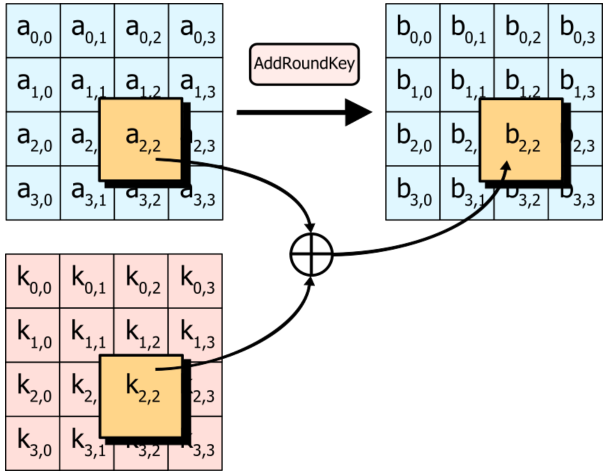


Computations on a Masked State

- Process shares individually, but still need same result
  - $f(v) = f(v' \oplus v'') = f(v') \oplus f(v'')$
- Only true for linear functions (with respect to $\oplus$)
- What about the AES?
  1. AddRoundKey
  2. SubBytes
  3. ShiftRows
  4. MixColumns

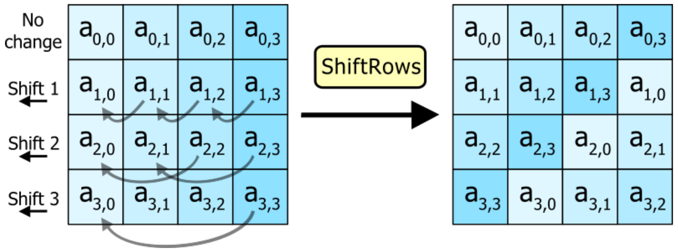


Masking on AES



AddRoundKey

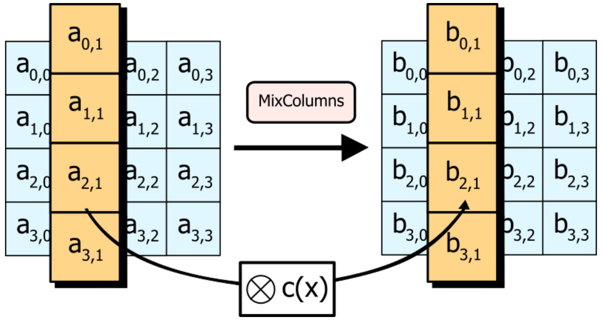
both a, k are shared: $a \oplus k = (a' \oplus k') \oplus (a' \oplus k'')$



ShiftRows

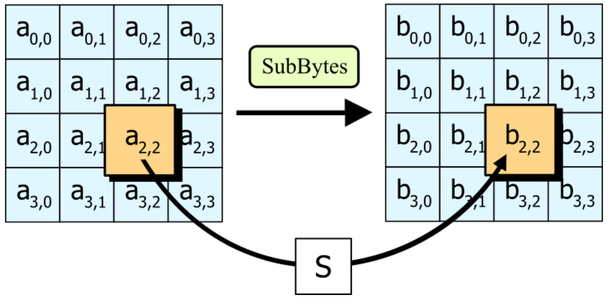
shift both shares individually:
$\text{ShiftRows}(a) = \text{ShiftRows}(a') \oplus \text{ShiftRows}(a'')$

Masking on AES



MixColumns

apply MixColumns (MC) on shares individually:
$MC([a_1, a_2, a_3, a_4]) =$
$MC([a_1', a_2', a_3', a_4']) \oplus MC[a_1'', a_2'', a_3'', a_4''])$



SubBytes

SubBytes is nonlinear!
SubBytes(a) $\neq$ SubBytes(a') $\oplus$ SubBytes(a'')

What now?

- There are still ways to protect nonlinear functions

- „Masking Scheme“ = how we deal with nonlinearities
  - many different schemes
  - shares need to „communicate“, gets tricky very fast

Countermeasures - Recap

- Ideal: mixture of countermeasures
  - masking + hiding + key updates + …

- Important: all countermeasures can again be broken!
  - more sophisticated attacks, more measurements, …
  - but ideally, effort gets much larger

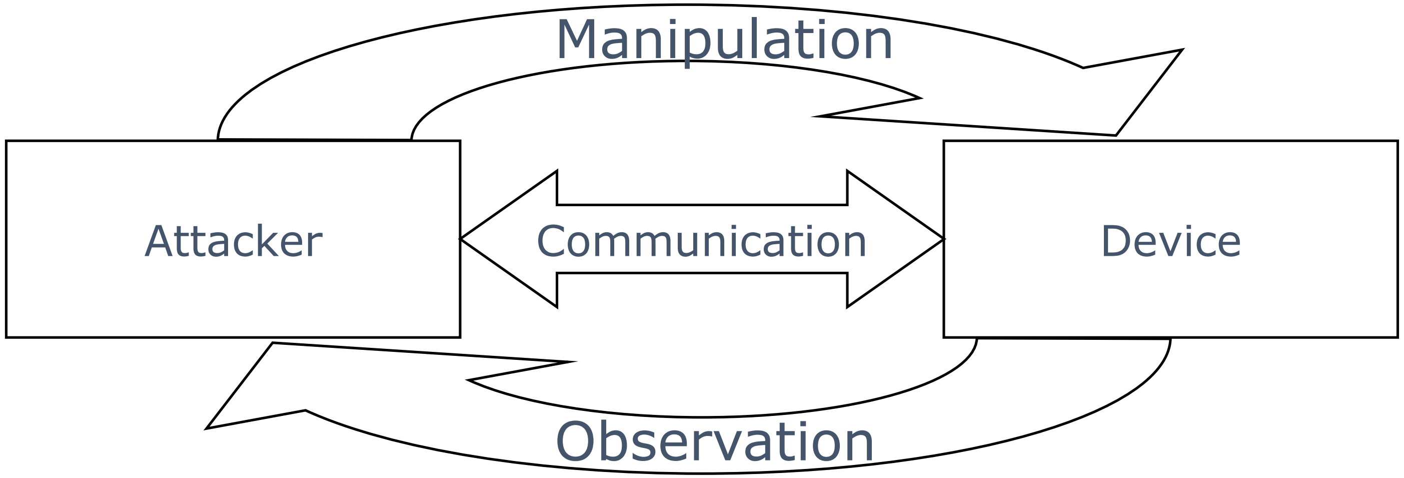
  - note: only explained most basic versions of the countermeasures
    there exist much better (and complex) variants

Active Attacks

Attacking by manipulating the device

Physical-Attack Scenario

- Attacker has (legitimate) access to device
- Thus far: passive attacks (and countermeasures)
  - attacker just listens
- But the attacker can do much more…



Basic Idea of Active Attacks

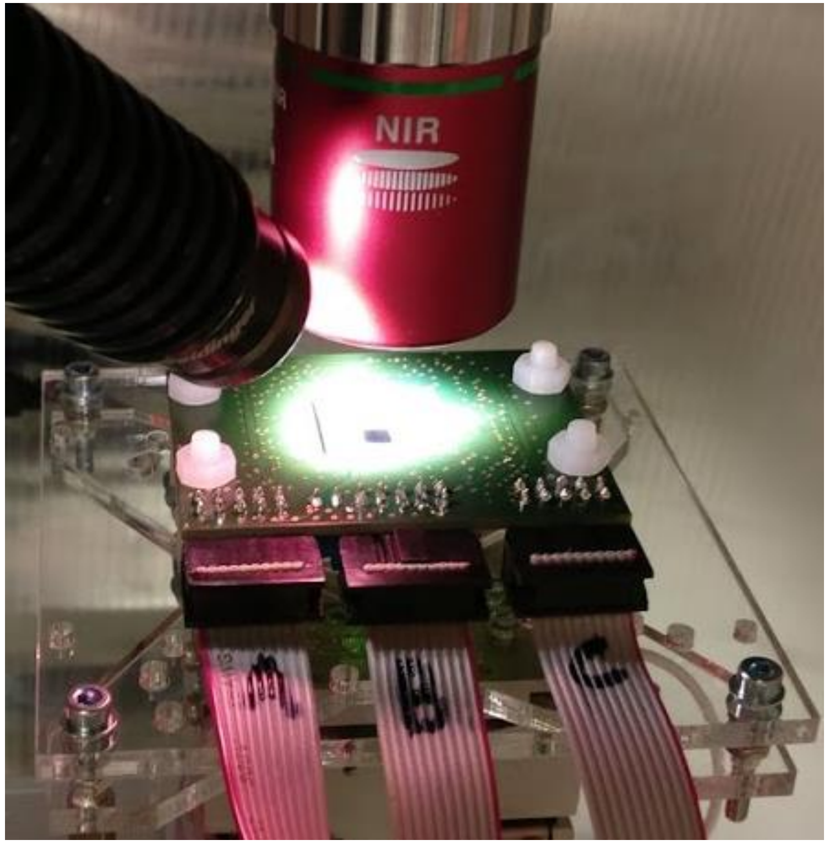
Goal: manipulate the device in order to compromise its security

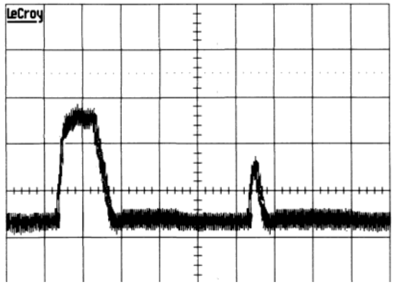
- Changing the general behavior of the device
  - Deactivation of countermeasures or sensors
  - Change of program code (e.g. skip PIN check, …)
  - …
- Faults in a cryptographic algorithm
  - Device calculates faulty ciphertexts and actual ciphertexts
  - Use difference to reveal the key

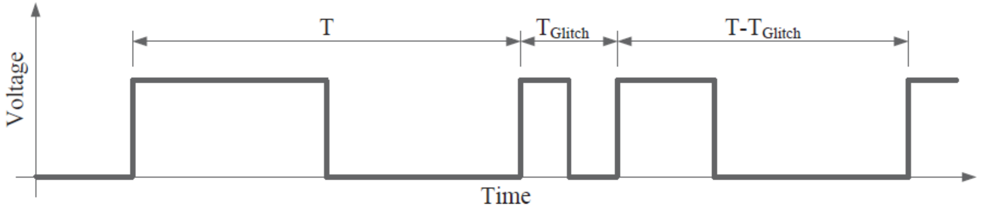
Fault Attacks: Techniques

- Fault injection techniques
  - spike / glitch attacks (clock, VDD, IO, …)
  - Laser
  - …

- Effects
  - instructions skipped
  - data corrupted
  - …







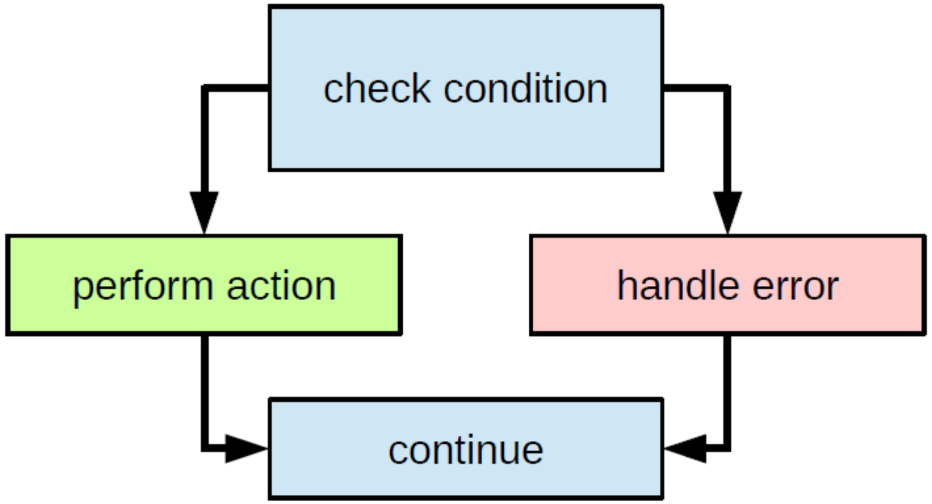
But…

- same ideas behind many attack paths!

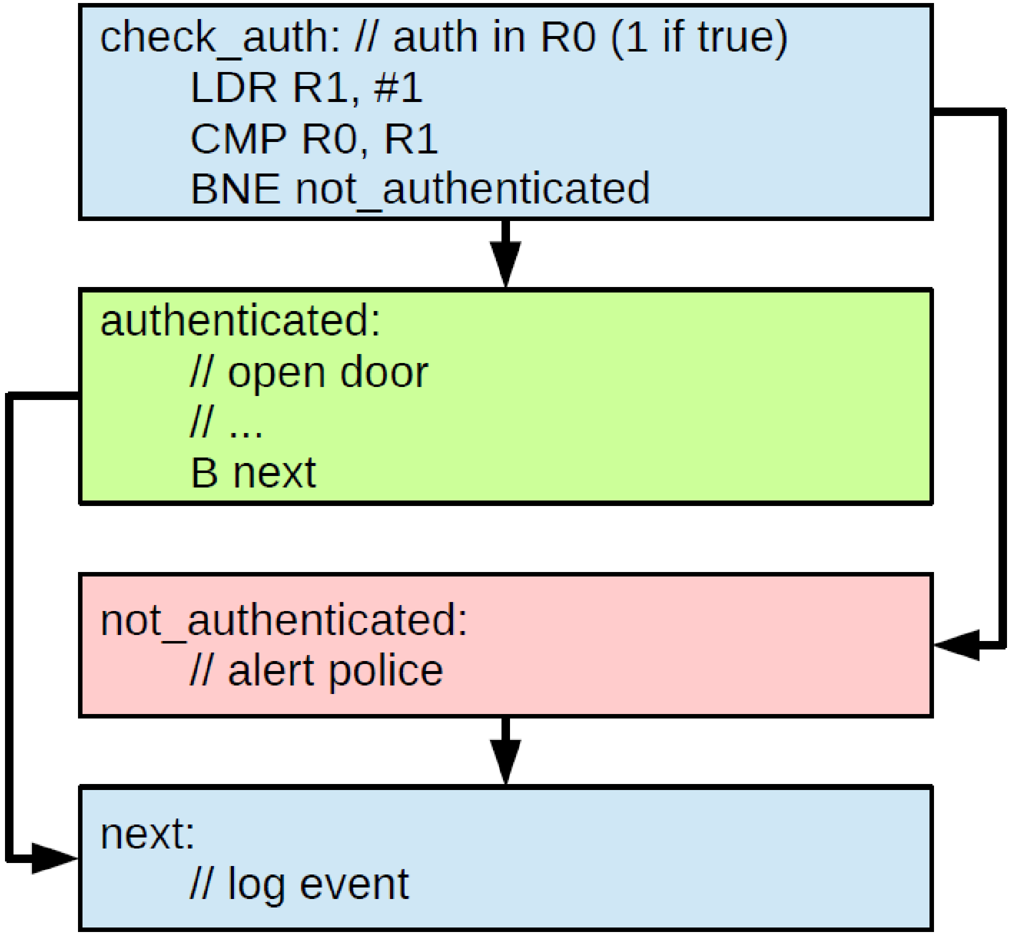
- Different fault-injection techniques, but same exploitation
  - different techniques can result in different faults
  - but once there is a fault, it doesn‘t matter how it was injected

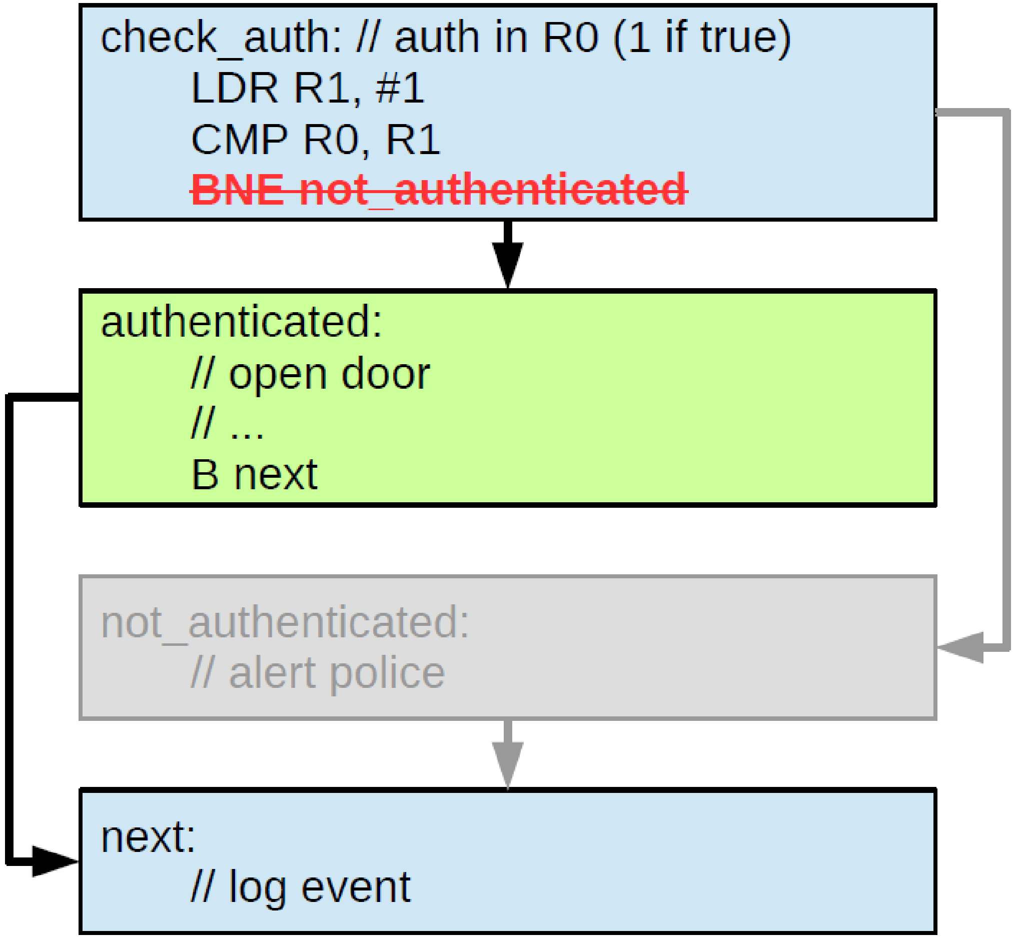
Example: PIN Check

```
unsigned pin = read_pin();
bool auth = tpm_check(pin);
if( auth ) {
        open_door();
} else {
        alert_police();
}
log_event();
```



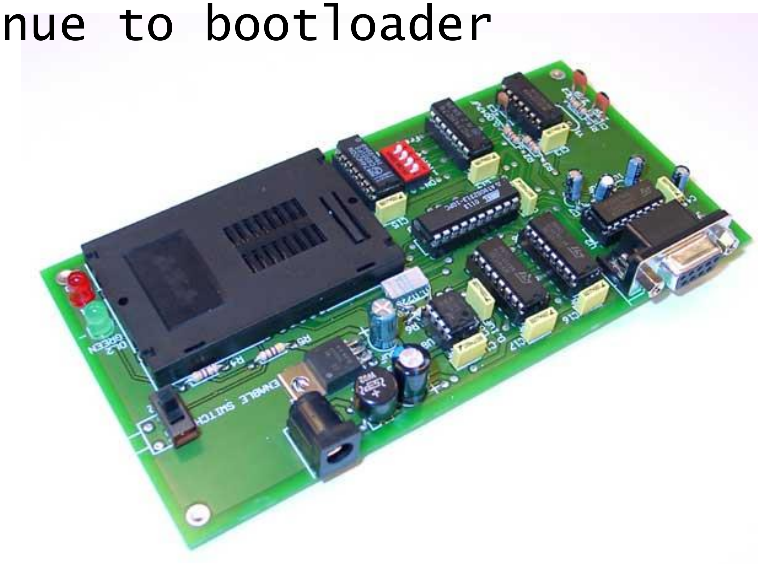
Example: PIN Check – Skipping Attack





Real-World Example: Piracy

```
// startup
loop: jmp loop;
// continue to bootloader
```



- PayTV (early 2000s)
  - pirated cards bricked via remote firmware update
  - inserted infinite loop, otherwise unchanged
  - solution: glitching to increment IP, but no jmp
  - „Unlooper“ device

- Gaming devices
  - Xbox360 reset hack
  - voltage glitching on reset line
  - execute untrusted code (modified firmware)

A Fault Attack on RSA

aka the „textbook“ fault attack

Discrete Maths: Chinese Remainder Theorem (CRT)

- For an unknown x, you are given its remainder with multiple moduli

$$x \equiv a_1 \bmod n_1$$
$$x \equiv a_2 \bmod n_2$$
$$x \equiv a_3 \bmod n_3$$
…

- CRT states: given all a_i, n_i, you can find x
- Solution for two equations
  - compute m_1, m_2 such that $m_1 n_1 + m_2 n_2 = 1$ (extended Euclid)
  - $x = (a_1 m_2 n_2 + a_2 m_1 n_1) \bmod n_1 n_2$

Why am I telling you this?

- RSA signatures: compute $S = \mathrm{m}^{\mathrm{d}} \bmod \mathrm{n}$
  - remember: $\mathrm{n} = \mathrm{pq}$
  - also remember Fermat‘s little theorem: $\mathrm{a}^{\mathrm{x}} \bmod \mathrm{p} \equiv \mathrm{a}^{\mathrm{x} \bmod \mathrm{p}\text{-}1} \bmod \mathrm{p}$

- CRT + RSA
  - 2 exponentiations with half the bit-length and smaller exponents

$$\mathrm{S_p} = \mathrm{m}^{\mathrm{d} \bmod (\mathrm{p}\text{-}1)} \bmod \mathrm{p}$$
$$\mathrm{S_q} = \mathrm{m}^{\mathrm{d} \bmod (\mathrm{q}\text{-}1)} \bmod \mathrm{q}$$

$$\begin{aligned}\mathrm{S} &= (\mathrm{a} \cdot \mathrm{S_p} + \mathrm{b} \cdot \mathrm{S_q}) \bmod \mathrm{n} \\ &= (\mathrm{q} \cdot (\mathrm{q}\text{-}1 \bmod \mathrm{p}) \cdot \mathrm{S_p} + \mathrm{p} \cdot (\mathrm{p}\text{-}1 \bmod \mathrm{q}) \cdot \mathrm{S_q}) \bmod \mathrm{n}\end{aligned}$$

"Bellcore" Attack against RSA CRT

- Run signing algorithm twice, inject fault in either S_p or S_q (here: S_p)
  - faulty value: S', S_p'
- Key recovery: $q = \gcd(S\text{-}S', n)$

- Why does this work?

$$
\begin{aligned}
S'\text{-}S &\equiv (a \cdot S_p' + b \cdot S_q) - (a \cdot S_p + b \cdot S_q) \\
&\equiv a \cdot (S_p'\text{-}S_p) \bmod n \equiv q \cdot (q^{-1} \bmod p) \cdot (S_p'\text{-}S_p) \bmod n \\
&= q \cdot y
\end{aligned}
$$

$$\gcd(S'\text{-}S, n) = \gcd(q{\cdot}y, q{\cdot}p) = q$$

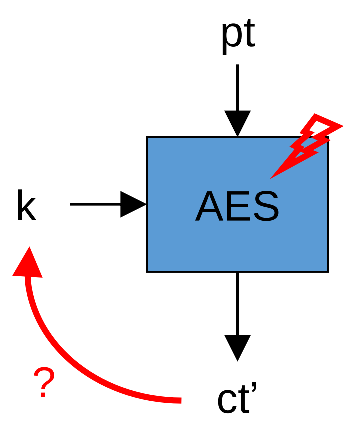
Fault Attacks on the AES

Idea…

- Inject fault during AES encryption
  - get: faulty ciphertext ct’
  - want: key
  - → faulting alone is only half the game!

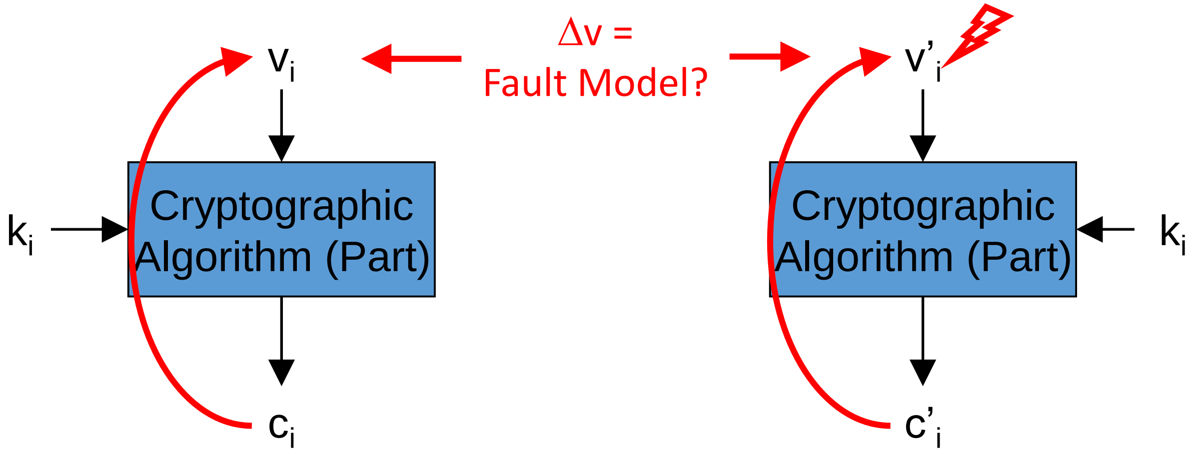
- Idea: compare correct and faulty ciphertext
  - encrypt same plaintext twice, once with a fault
  - use difference in ciphertext to recover the key

  - Differential Fault Attack



Differential Fault Attacks – Basic Principle

- Pick an intermediate v
  - intermediate that will be combined with a small part of the last round key
- 2 invocations with same pt, 1 with a fault in v
  - usually don‘t know exact fault, but often fault model (e.g., flip 1 bit)
- Enumerate possible subkey values
  - compute backwards for each guess
  - check if XOR-difference = fault model
  - wrong guess: „randomize“ v and Δv
- Helpful: AES key schedule is invertible
  - if it were not: attack decryption or attack round after round

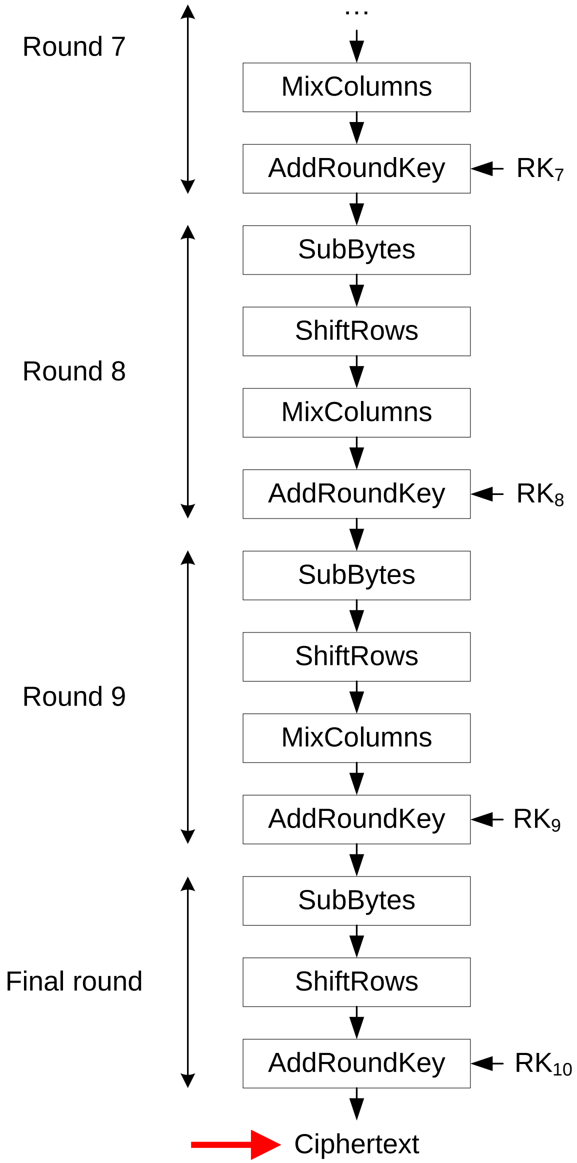


Differential Fault Attacks on AES

Faulting Ciphertext?

NO

Ciphertext difference does not depend on key!



Differential Fault Attacks on AES

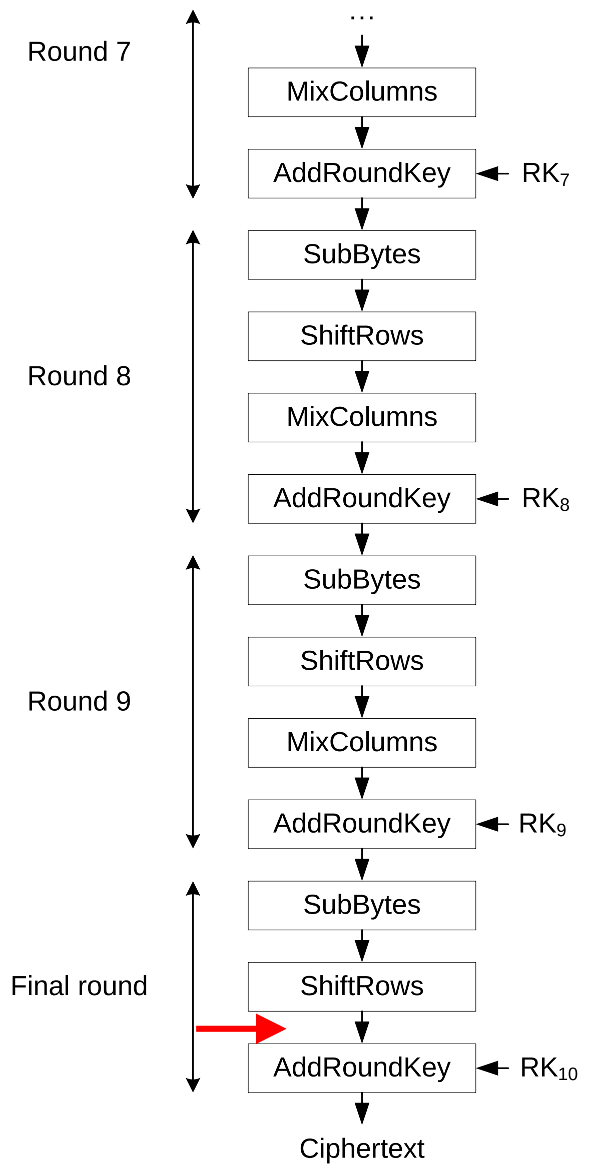
Faulting before AddRoundKey10?

…depends on faults

with bit flips (random or known)

- no attack possible
- fault propagates through XOR→ ciphertext difference does not depend on key

$$c = v \oplus k$$
$$c' = (v \oplus \Delta v) \oplus k = c \oplus \Delta v$$

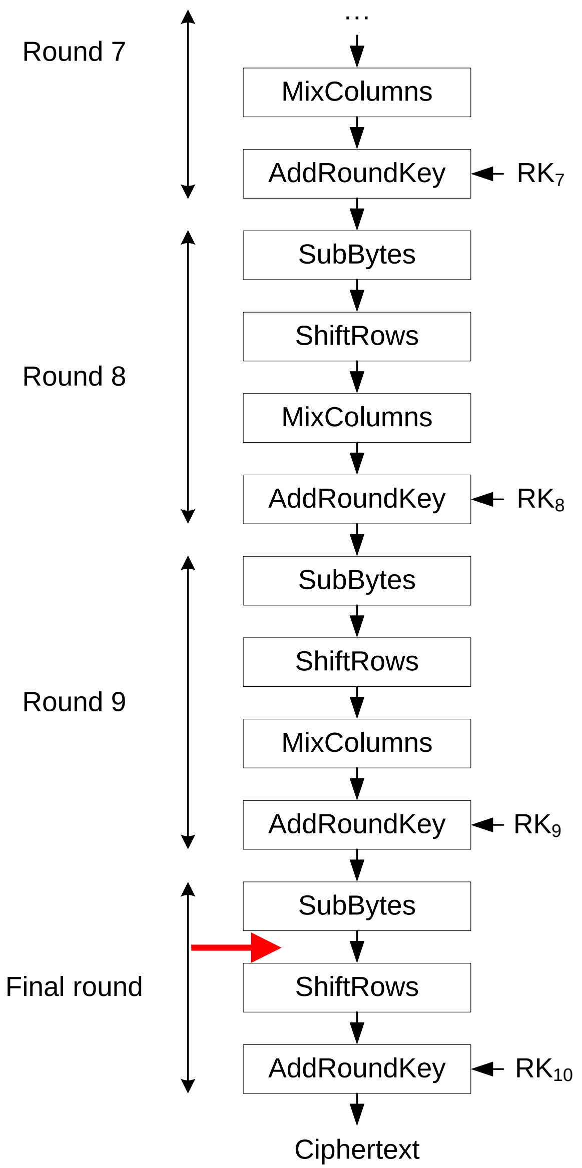


Differential Fault Attacks on AES

Faulting before ShiftRows10?

same situtation as for AddRoundKey

- ShiftRows just rearranges bytes



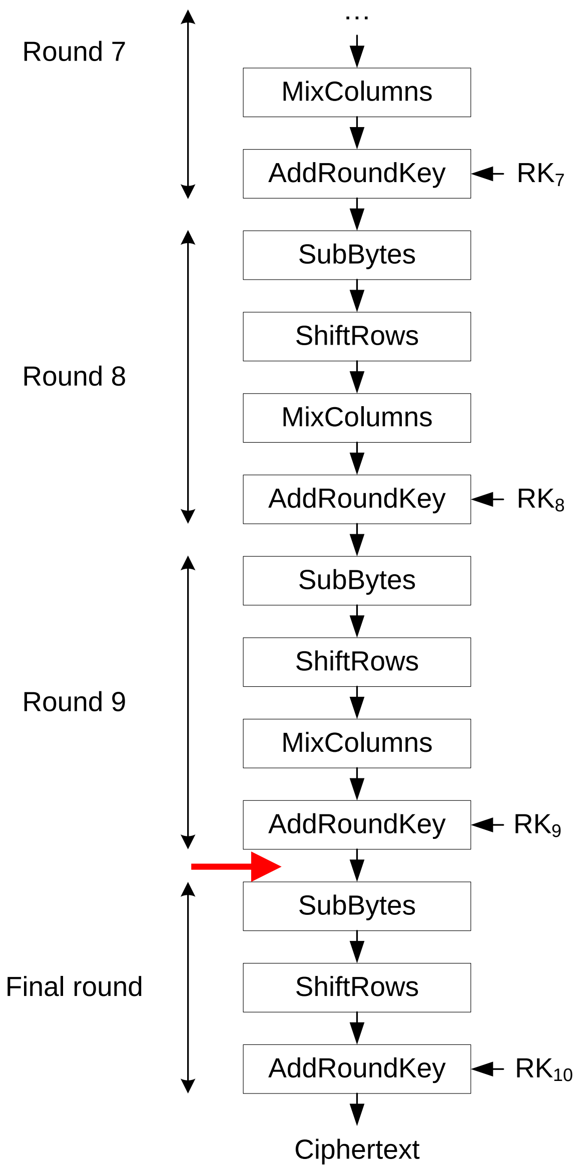
Differential Fault Attacks on AES

Faulting before SubBytes10?

…depends on faults

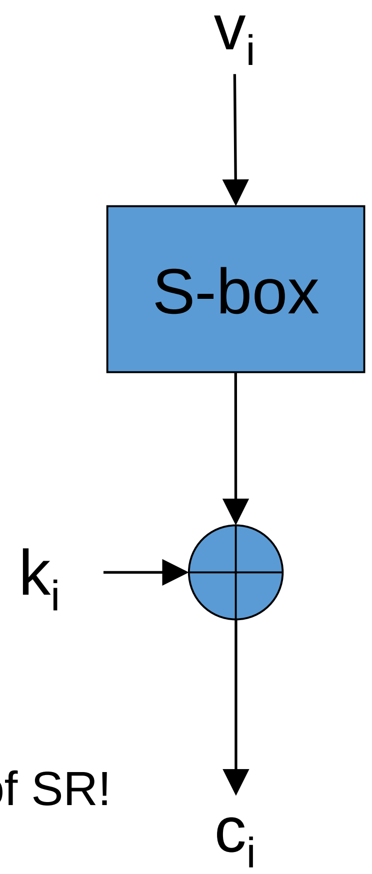
Flip 1 bit?

- attack possible



Single-Bit Fault before SubBytes

- Receive correct and faulty ct
- Enumerate all 2^8 values for k_i
  - compute back to v (for correct and fault, for all possible k_i)
  - compute Δv for all k_i
  - check if Δv follows fault model (1 bit fault)

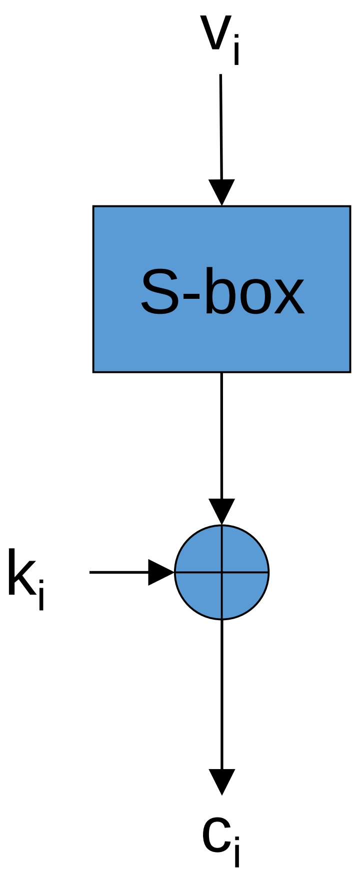


ShiftRows is omitted here.
But remember: indizes can be different because of SR!

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):
```



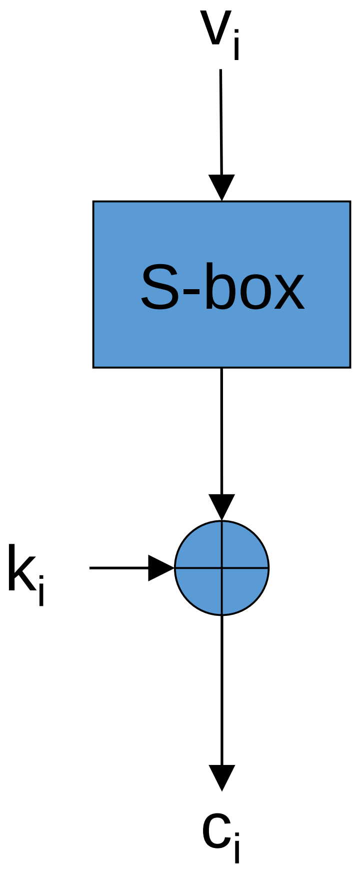
Single Bit Fault before SubBytes - Example

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 …
```

Only few keys have this property: filter them!

Use another c/c’ pair to get down to 1 key (w.h.p.)



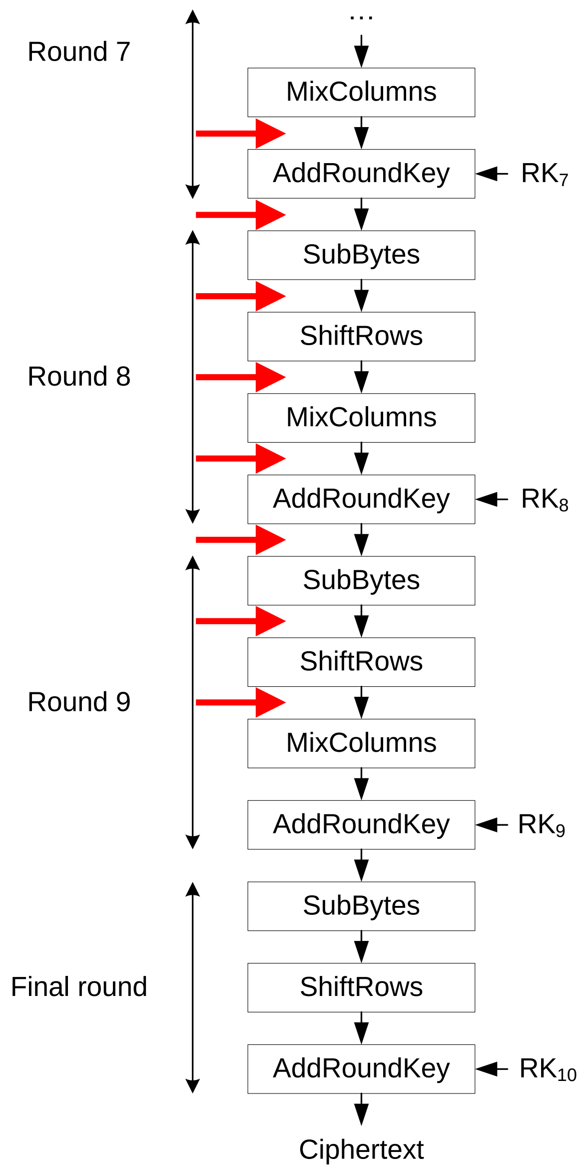
Differential Fault Attacks on AES

Problem: Precise bit flips hard to achieve

More sophisticated attacks:

random faults on bytes or 32 bits in earlier stages

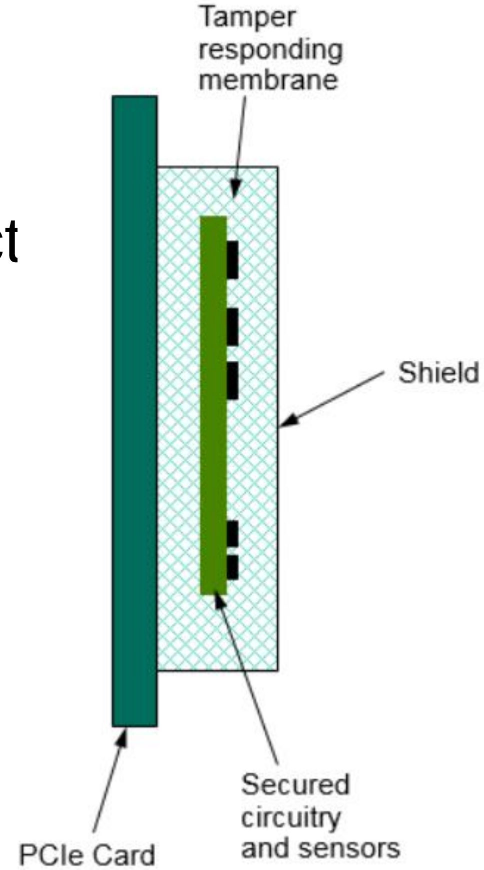
(much easier to achieve)



Active Attacks: Countermeasures

Analog Countermeasures

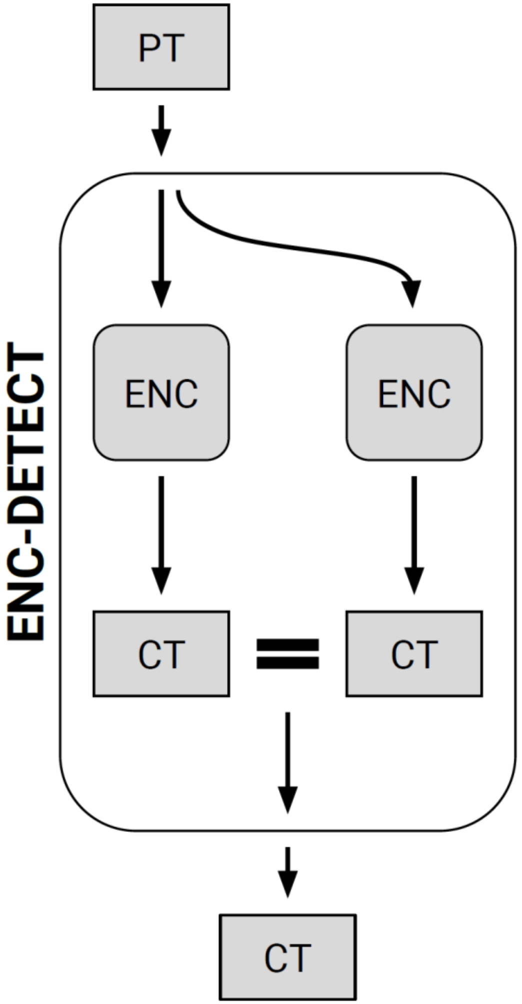
- Sensors to detect anomalies
  - active meshes: fine wire mesh across IC, disruption is detect
  - power surge sensors
  - temperature sensors
  - light sensors



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, …
- Attack
  - attacker injects same fault twice (difficult…)
  - or use more sophisticated methods (statistical attacks)



An Important Takeaway

- Attack, Countermeasure, another Attack, next Countermeasure, …
  - different side channels, more refined attacks, etc.
  - never-ending game of cat and mouse
- In the physical setting, there is no absolute security!
  - …each device can be broken by a determined attacker
- Our goal: ensure that attack effort >> value of secret
  - Would you do an attack costing millions to get some free tram rides?

Found that interesting?

We discuss more details in the Master-level course **Embedded Security,** where students also perform experiments on real hardware.

Next Week

VO: Microarchitectural Side-Channel Attacks

from cache attacks to Meltdown

KU: Physical Side Channels: Demos and Tutorials

demos for power analysis and faults, short tutorials for KU

Images

- [1] Cracking a Safe - Breaking into a Vault: By Blue Coat Photos [CC BY-SA 2.0 (http://creativecommons.org/licenses/by-sa/2.0)], via Flickr
- [2] Intel Core microarchitecture: By „Appaloosa“ [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0), via Wikimedia Commons