# Intro to Side-Channel Analysis: Concepts and Techniques

## (AI-assisted) side-channel attacks on real-world crypto implementations

## Hardwear.io 2024 (Amsterdam) training, Lecture 1, Day 1

Lejla Batina[1], Łukasz Chmielewski[2] & Péter Horváth[1]

[1] Digital Security group, Radboud University, The Netherlands
[2] Centre for Research on Cryptography and Security, Masaryk University, Czechia

October 21, 2024

# Training schedule

- Day 1: Intro to Side-channel attacks
  - Lecture: Side-channel attacks on crypto implementations
  - Two Assignments
- Day 2: Advanced attacks
  - Lecture: Leakage evaluation: TVLA and alternatives + Assignment
  - Lecture: Profiling attacks + Assignment
  - Lecture: AI basics
- Day 3: AI and SCA
  - Lecture: AI-assisted SCA
  - Assignments
  - Lecture: Leakage simulators
  - Lecture: Higher order attacks (if there is time)
  - Unfinished Excercises

# Day 1

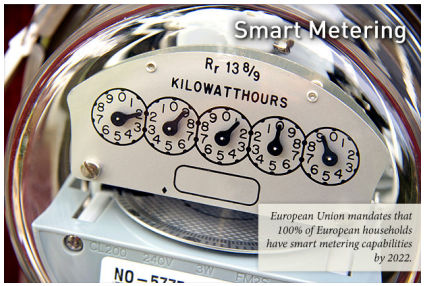
- Let's know each other!
- Basic concepts and principles – gray box, platforms, setups, side channels
- Side-channel leakage modeling
- Simple power analysis (SPA)
- Differential power analysis (DPA)
- Correlation power analysis (CPA)
- Assignments 1-2: Hands-on DPA attacks on software and hardware implementations
  - For now ignore the TVLA part
- Resources: `https://tinyurl.com/ysnznaka` [1]

[1] Full: `https://www.dropbox.com/scl/fo/bd9r3lvzbk7eilqgytroo/ADK3b5aUicY2hzshgYCCQ8g?rlkey=5uzcphdvbnhlhs4z6zal63tqc&st=j5al094h`

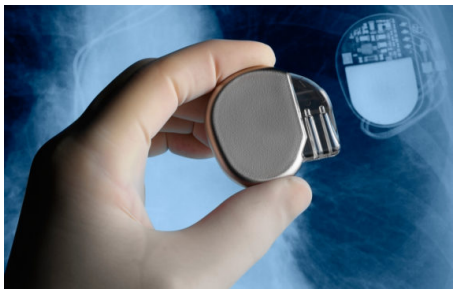
# Outline

# Introducing the side channel

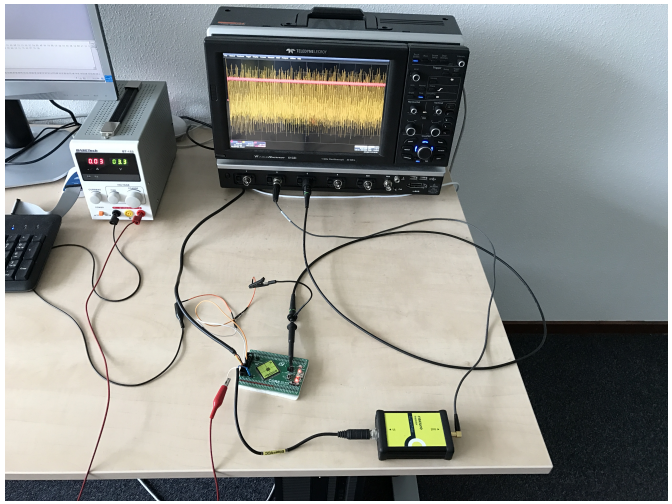
# Known challenge: embedded crypto devices






# Implementation attacks



# Relevance

**Minerva, October 3, 2019**

Researchers Discover ECDSA
Key Recovery Method

# Relevance

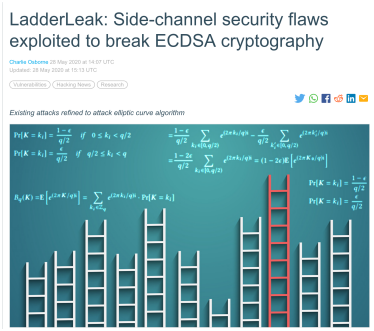
## TPM-FAIL, November 13, 2019


## Minerva, October 3, 2019

# Relevance

## TPM-FAIL, November 13, 2019


## LadderLeak, May 28, 2020

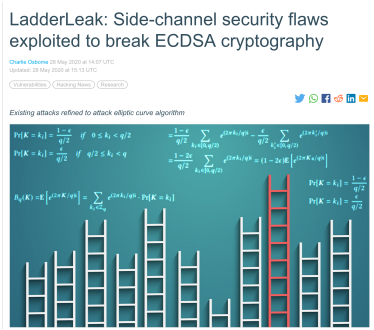



## Minerva, October 3, 2019

# Relevance

## TPM-FAIL, November 13, 2019


## LadderLeak, May 28, 2020



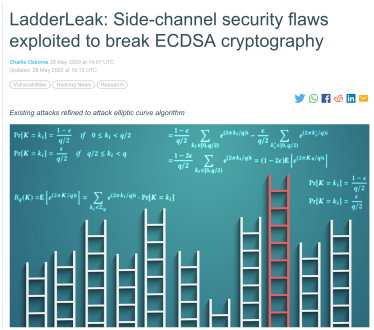

## SCA Titan: January 7, 2021

## Minerva, October 3, 2019

# Relevance

## TPM-FAIL, November 13, 2019


## LadderLeak, May 28, 2020




## SCA Titan: January 7, 2021

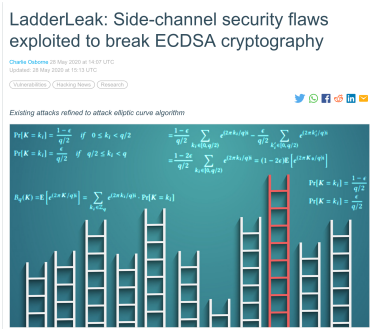
## Minerva, October 3, 2019


## TPMScan, March 12, 2024

# Relevance

## TPM-FAIL, November 13, 2019

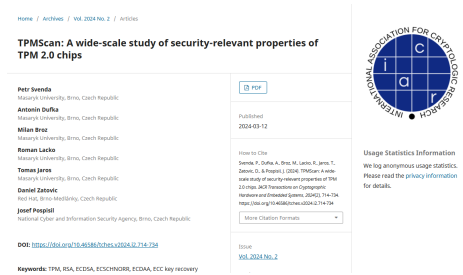

## LadderLeak, May 28, 2020




## SCA Titan: January 7, 2021

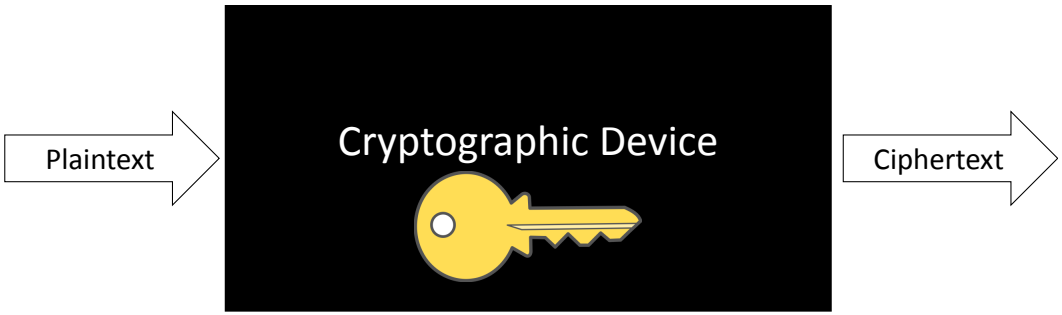
## Minerva, October 3, 2019


## TPMScan, March 12, 2024



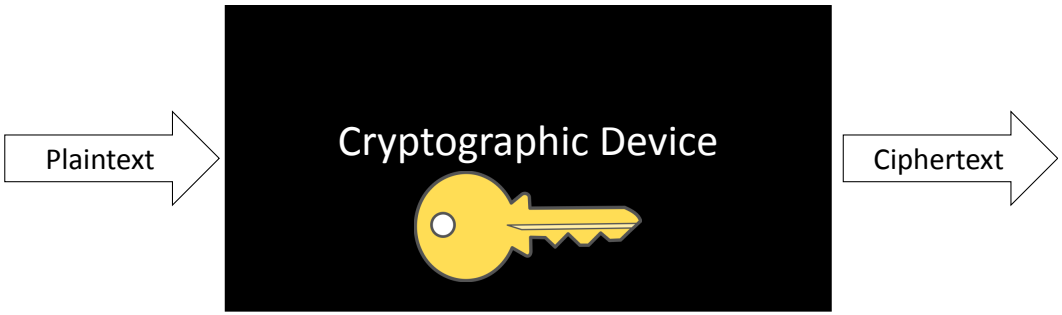

## EUCLEAK, Yesterday

# Blackbox scenario



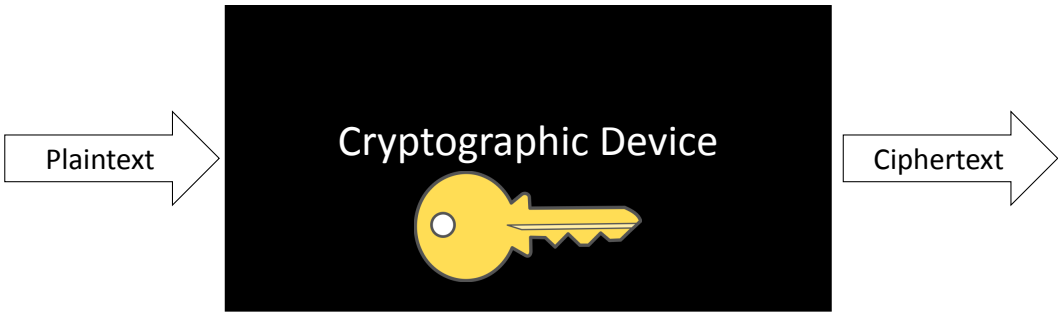

- Cryptographic function is a black box, parameterized with key, that maps plaintext into ciphertext

# Blackbox scenario




- Cryptographic function is a black box, parameterized with key, that maps plaintext into ciphertext
- Analyzing the security in the blackbox scenario relates to classical cryptanalysis

# Blackbox scenario




- Cryptographic function is a black box, parameterized with key, that maps plaintext into ciphertext
- Analyzing the security in the blackbox scenario relates to classical cryptanalysis
- Adversary's goal: secret key or plaintext recovery by observing plaintext/ciphertext pairs

# Greybox scenario

- Crypto is **implemented on a real device** such as a microcontroller, FPGA, ASIC etc.

# Greybox scenario

- Crypto is **implemented on a real device** such as a microcontroller, FPGA, ASIC etc.
- We can measure and process certain *physical quantities* in the device's vicinity

# Greybox scenario

- Crypto is **implemented on a real device** such as a microcontroller, FPGA, ASIC etc.
- We can measure and process certain *physical quantities* in the device's vicinity
- Adversary's goal: secret key or plaintext recovery by observing plaintext/ciphertext pairs **and a side channel**

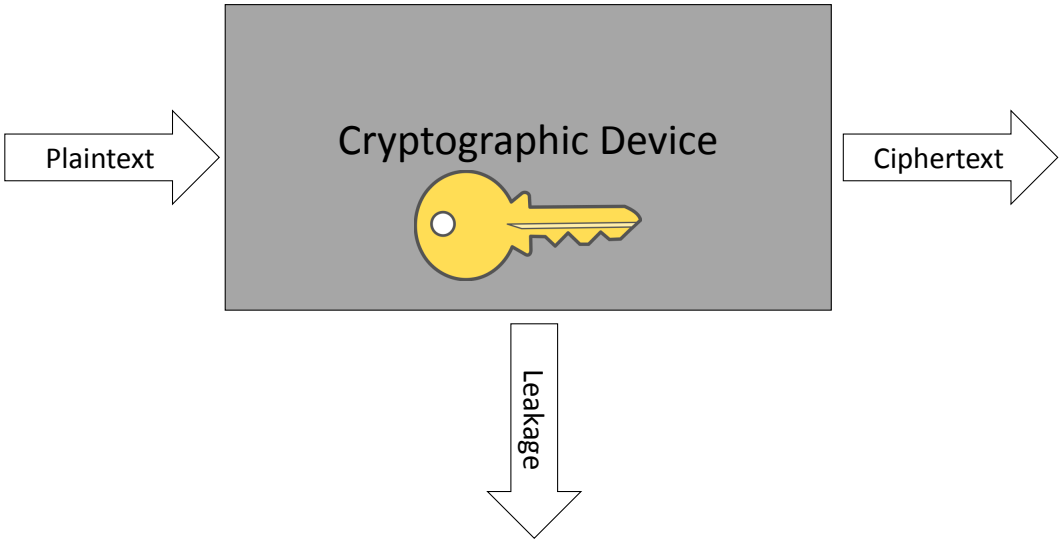
# Greybox scenario

- Crypto is **implemented on a real device** such as a microcontroller, FPGA, ASIC etc.
- We can measure and process certain *physical quantities* in the device's vicinity
- Adversary's goal: secret key or plaintext recovery by observing plaintext/ciphertext pairs **and a side channel**
- *Side channel* is any unintentional signal that can offer us a blurry view of the algorithm's internal computations

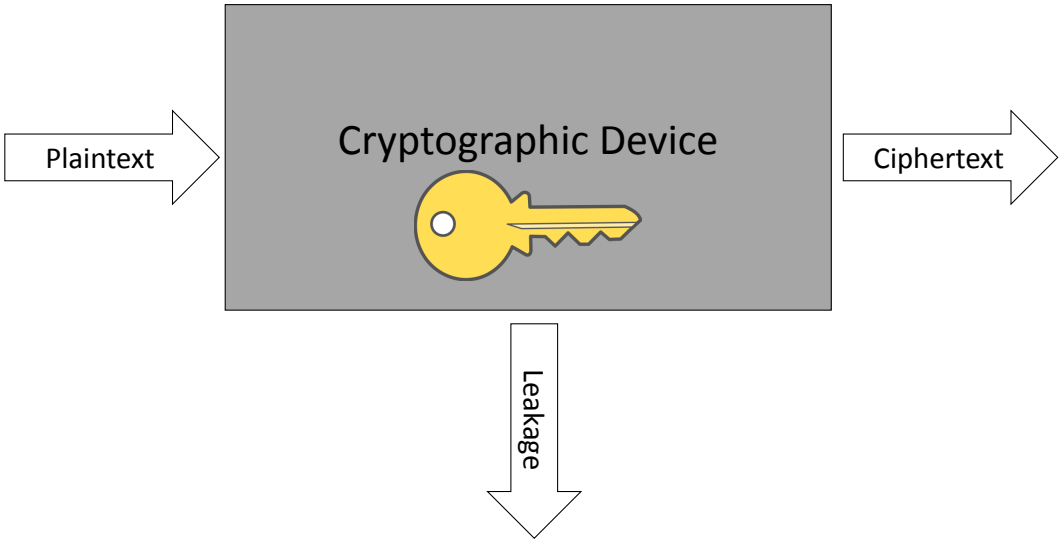
# Greybox scenario

- Crypto is **implemented on a real device** such as a microcontroller, FPGA, ASIC etc.
- We can measure and process certain *physical quantities* in the device's vicinity
- Adversary's goal: secret key or plaintext recovery by observing plaintext/ciphertext pairs **and a side channel**
- *Side channel* is any unintentional signal that can offer us a blurry view of the algorithm's internal computations
- Examples: execution/reaction time, power consumption, electromagnetic emission, sound

# Greybox scenario

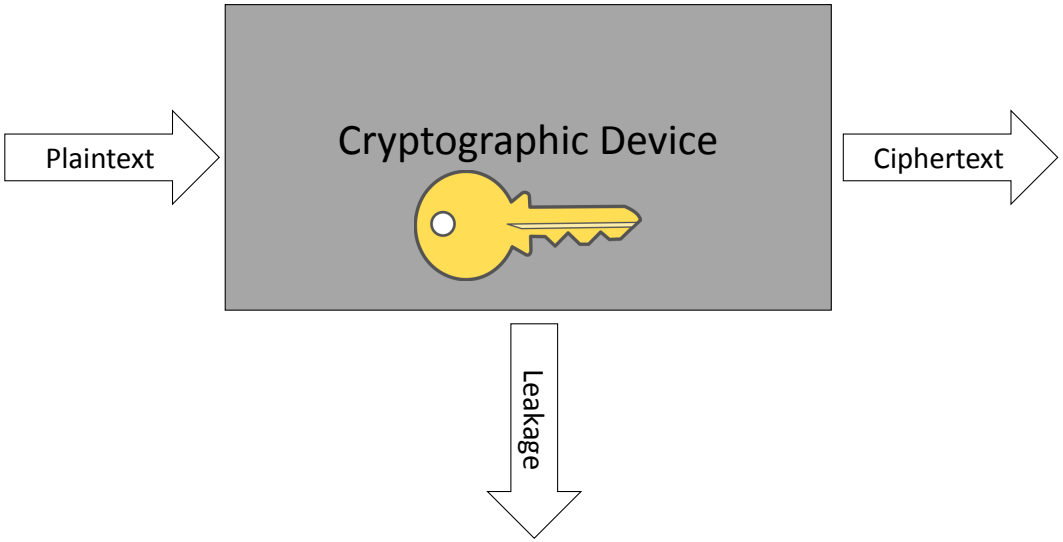


# Greybox scenario




- Assuming limited access to the internal computations through this side channel window → greybox scenario

# Greybox scenario




- Assuming limited access to the internal computations through this side channel window → greybox scenario
- Security in the blackbox scenario does not imply security under the greybox scenario

# Timing side-channel

# Timing side-channel: PIN verification

- Software for PIN code verification

```
Input: 4-digit PIN code
Output: PIN verified or rejected
Process CheckPIN (pin[4])
int pin_ok=0;
if (pin[0]==5)
   if (pin[1]==9)
      if (pin[2]==0)
         if (pin[3]==2)
            pin_ok=1;
         end
      end
   end
end
return pin_ok;
EndProcess
```

# Timing side-channel: PIN verification

- Software for PIN code verification

```
Input: 4-digit PIN code
Output: PIN verified or rejected
Process CheckPIN (pin[4])
int pin_ok=0;
if (pin[0]==5)
   if (pin[1]==9)
      if (pin[2]==0)
         if (pin[3]==2)
            pin_ok=1;
         end
      end
   end
end
return pin_ok;
EndProcess
```

- What are the execution times of the process for PIN inputs

```
[0,1,2,3], [5,3,0,2], [5,9,0,0]
```

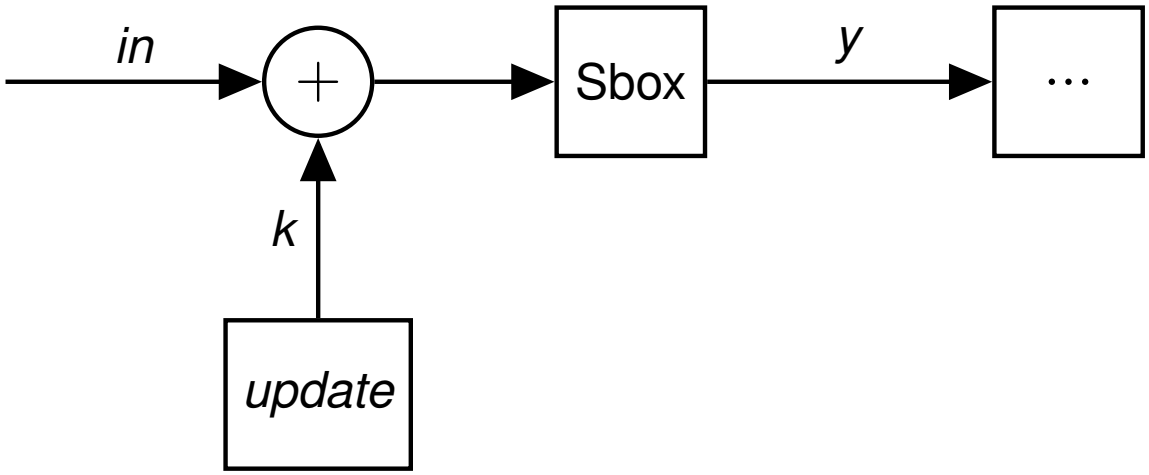
# Timing side-channel: PIN verification

- Software for PIN code verification

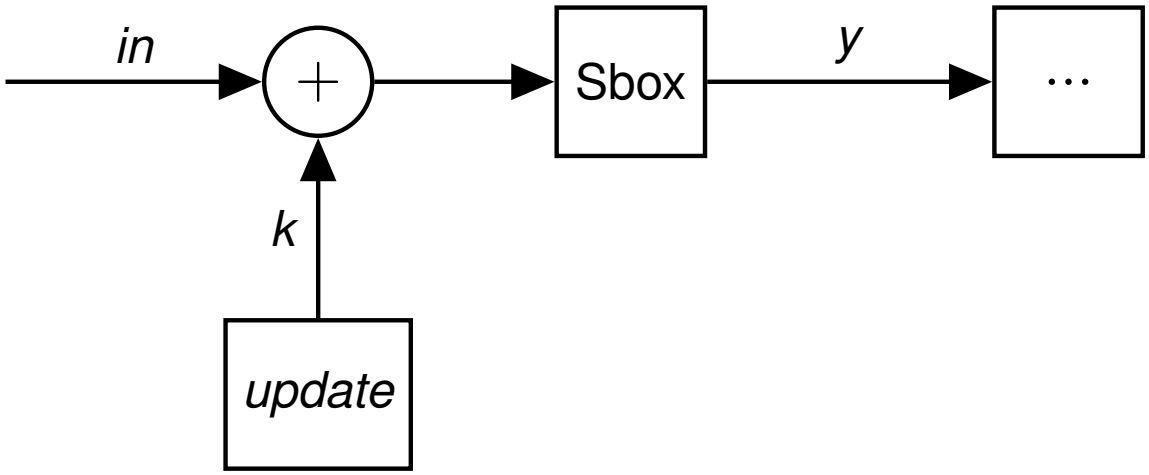
```
Input: 4-digit PIN code
Output: PIN verified or rejected
Process CheckPIN (pin[4])
int pin_ok=0;
if (pin[0]==5)
   if (pin[1]==9)
      if (pin[2]==0)
         if (pin[3]==2)
            pin_ok=1;
         end
      end
   end
end
return pin_ok;
EndProcess
```

- What are the execution times of the process for PIN inputs `[0,1,2,3]`, `[5,3,0,2]`, `[5,9,0,0]`
- The execution time increases as we get closer to `[5,9,0,2]`

# Timing side-channel: Cache attacks

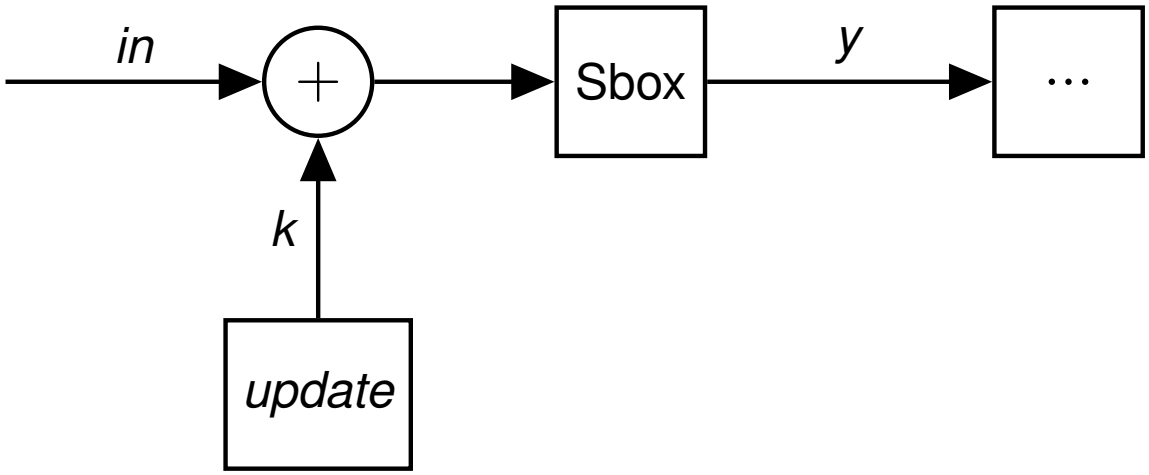


# Timing side-channel: Cache attacks



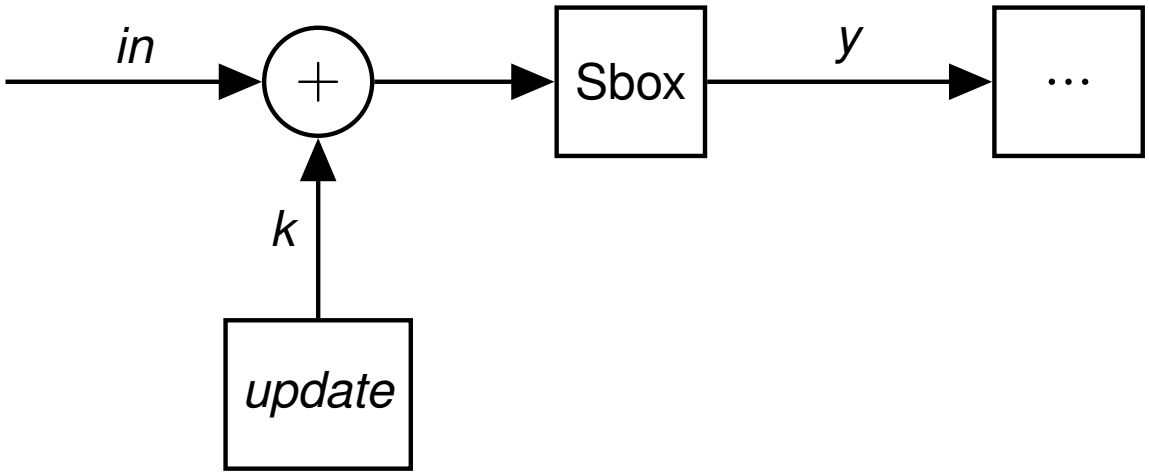

- Assume AES Sbox with a lookup table (LUT) stored in memory

# Timing side-channel: Cache attacks



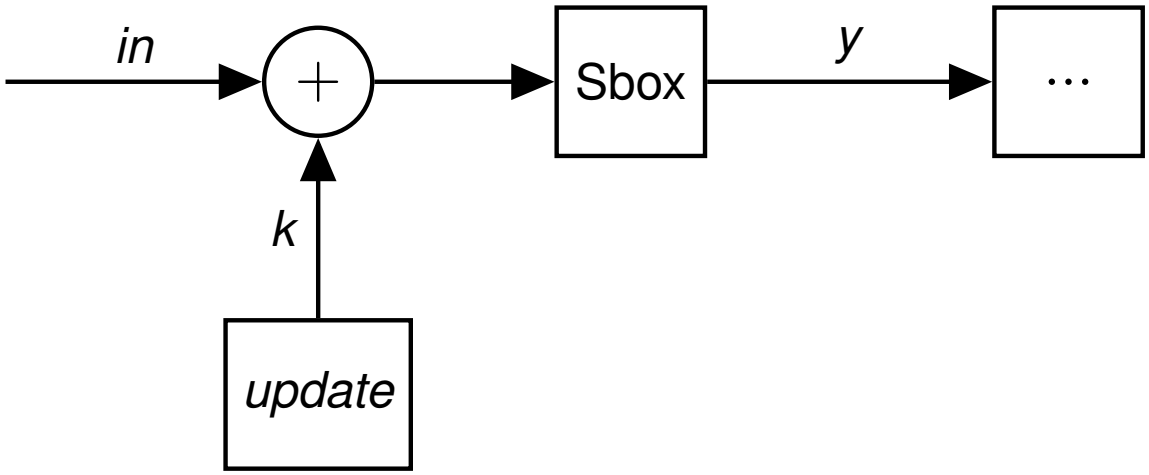

- Assume AES Sbox with a lookup table (LUT) stored in memory
- Every Sbox iteration creates a table index $j = input \oplus key$ and uses it to lookup the Sbox output, i.e. $y = LUT(j)$

# Timing side-channel: Cache attacks



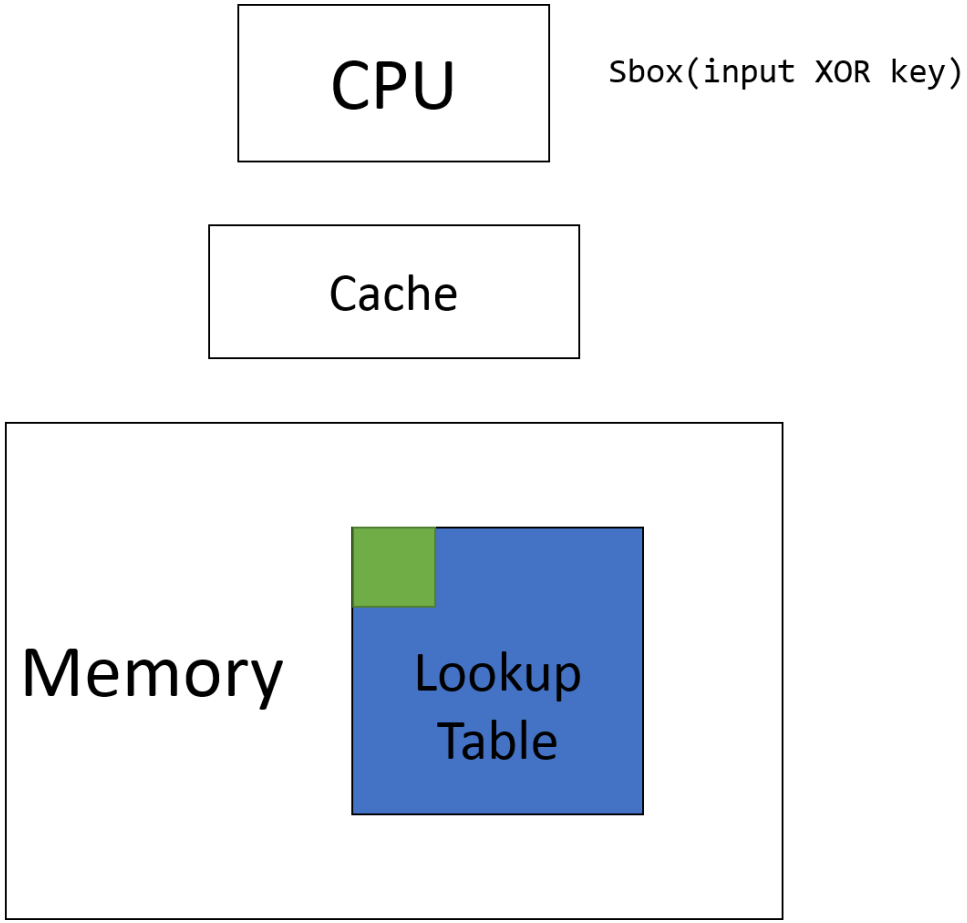

- Assume AES Sbox with a lookup table (LUT) stored in memory
- Every Sbox iteration creates a table index $j = input \oplus key$ and uses it to lookup the Sbox output, i.e. $y = LUT(j)$
- Accessing different parts of the lookup table may take different amount of time!

# Timing side-channel: Cache attacks

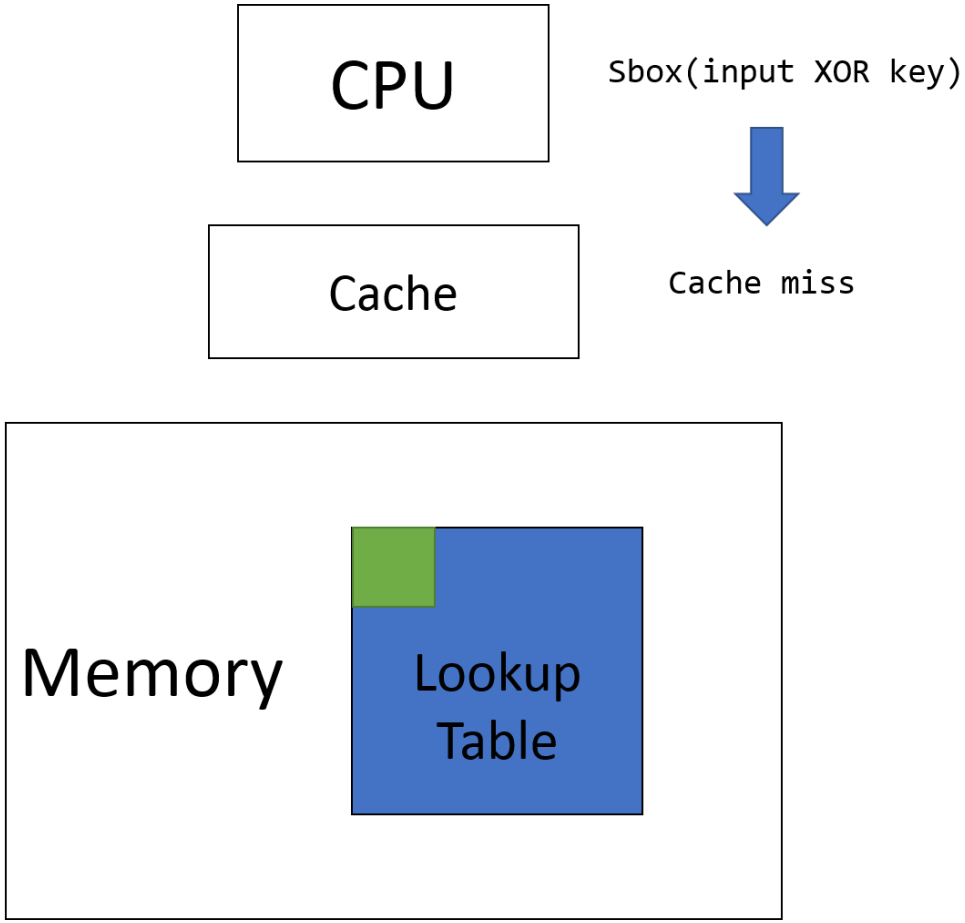



- Assume AES Sbox with a lookup table (LUT) stored in memory
- Every Sbox iteration creates a table index $j = input \oplus key$ and uses it to lookup the Sbox output, i.e. $y = LUT(j)$
- Accessing different parts of the lookup table may take different amount of time!
- The underlying cause is cache behavior of modern processors

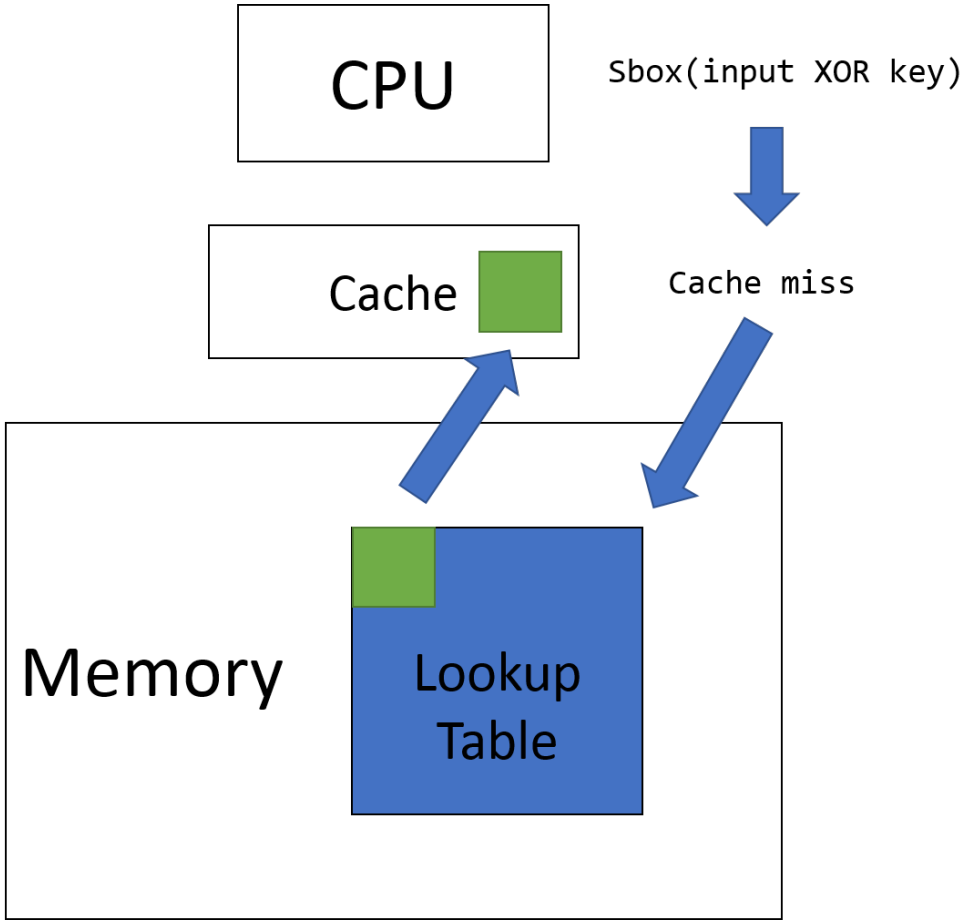
# Timing side-channel: Cache attacks



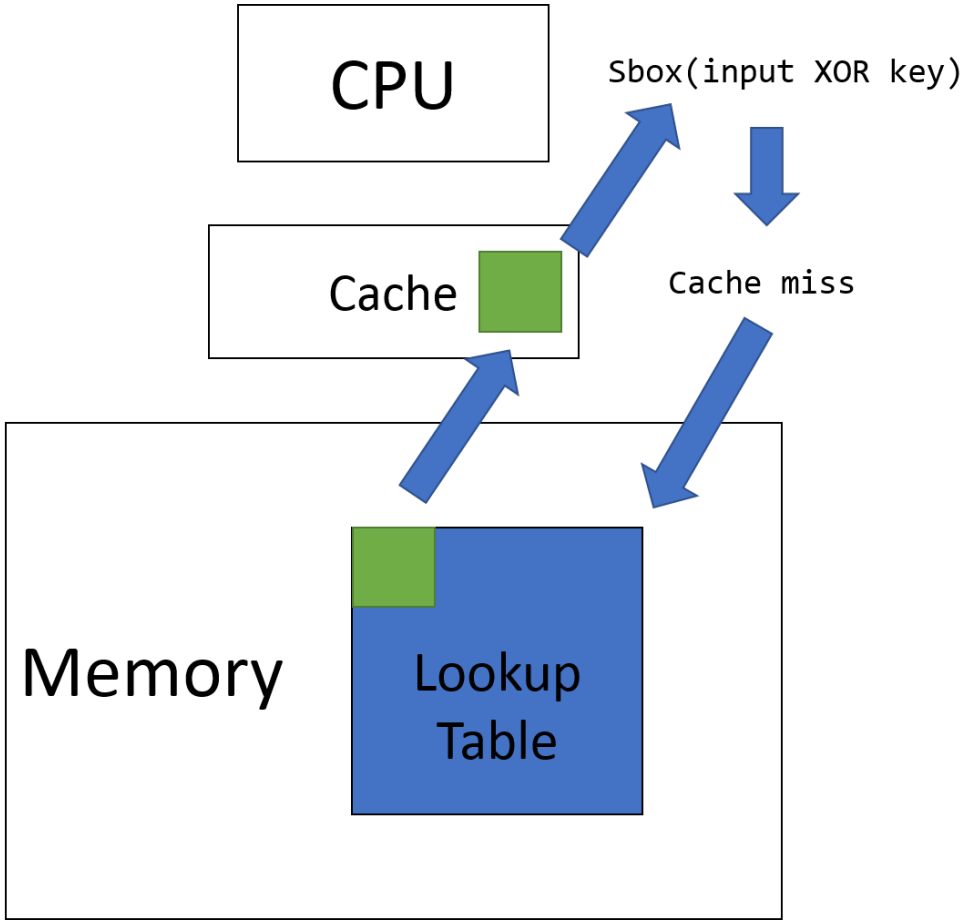
# Timing side-channel: Cache behavior



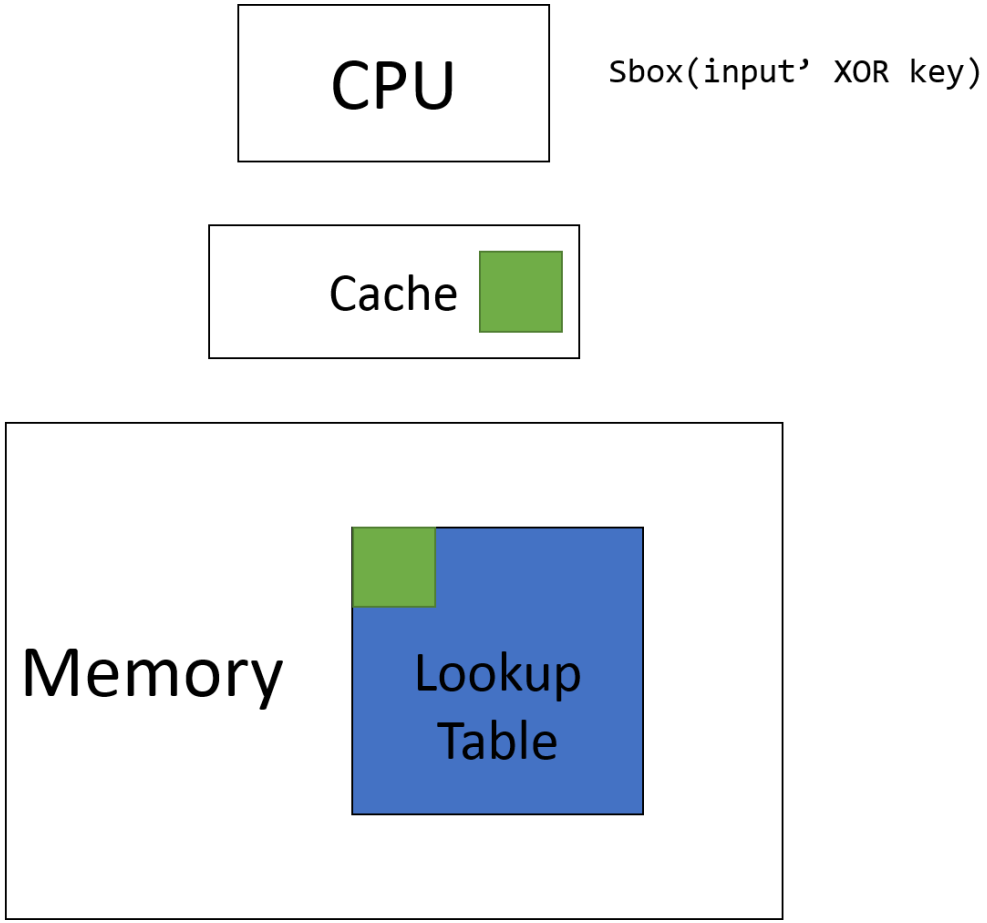
# Timing side-channel: Cache behavior



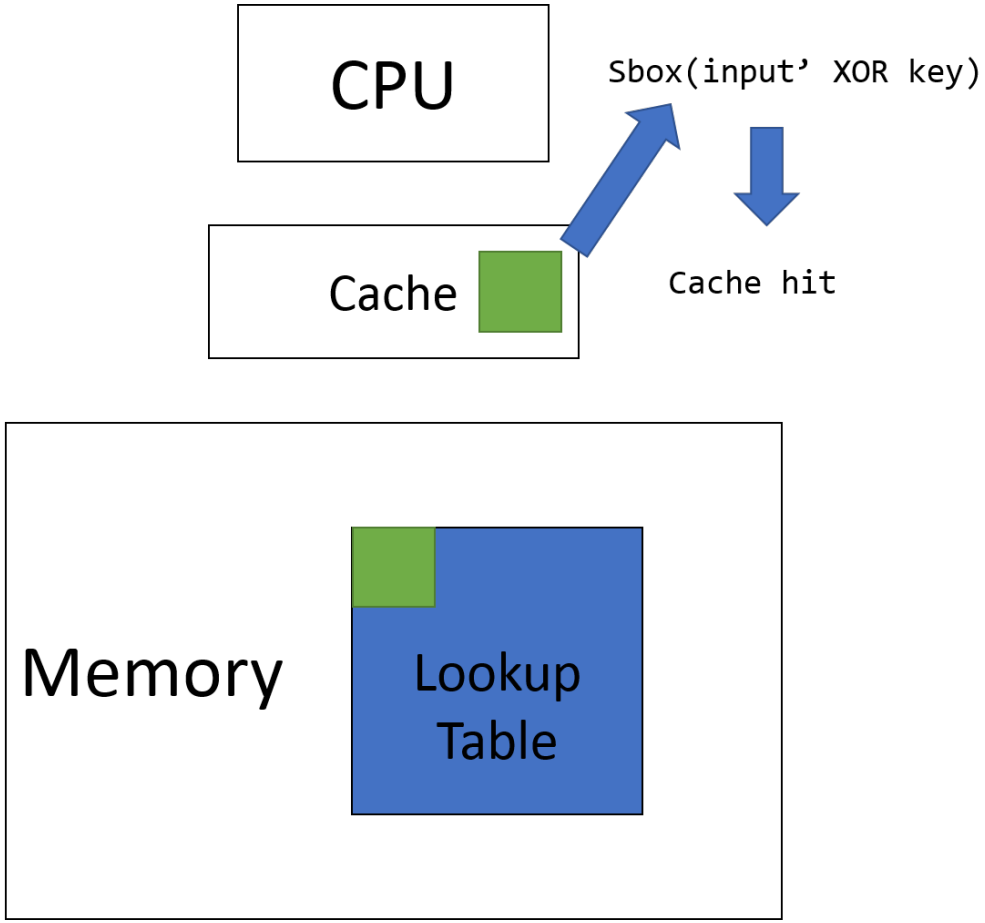
# Timing side-channel: Cache behavior



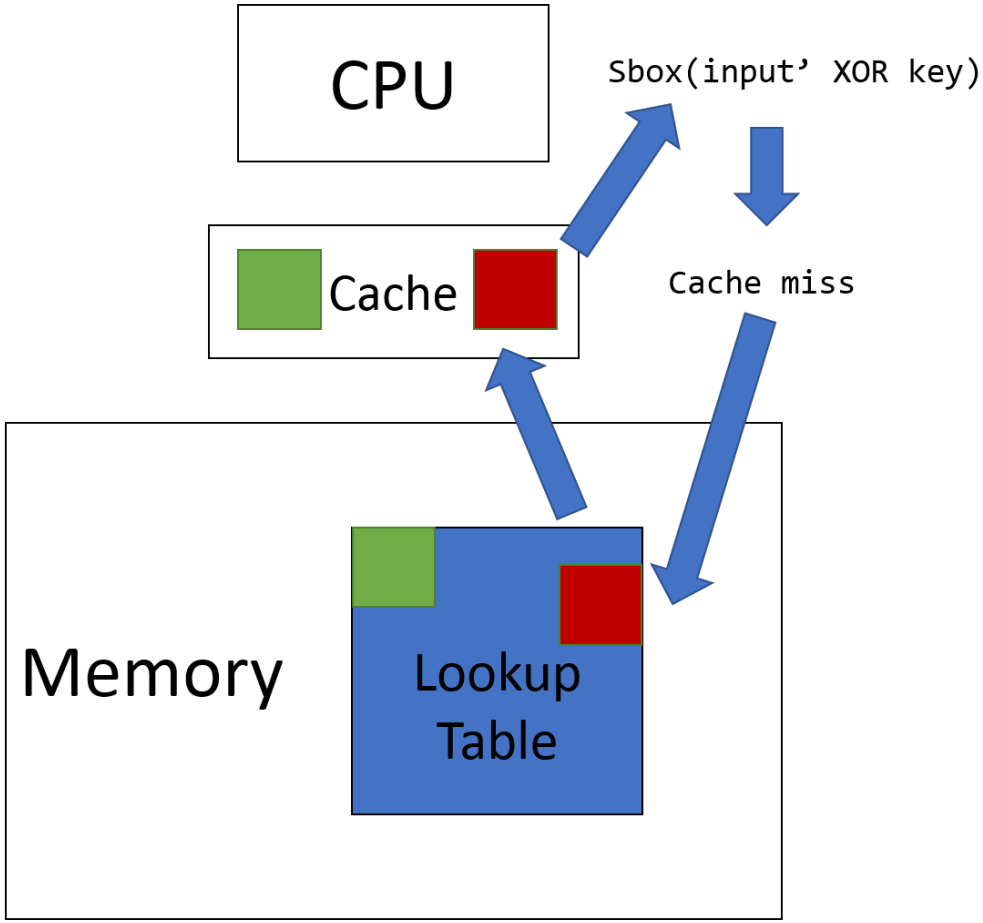
# Timing side-channel: Cache behavior



# Timing side-channel: Cache behavior



# Timing side-channel: Cache behavior



# Timing attacks notes and literature

- One of the earliest side-channel attacks due to easy measurements collection
- Can also be exploited remotely
- Exploits some not foreseen effects of caches to crypto implementations

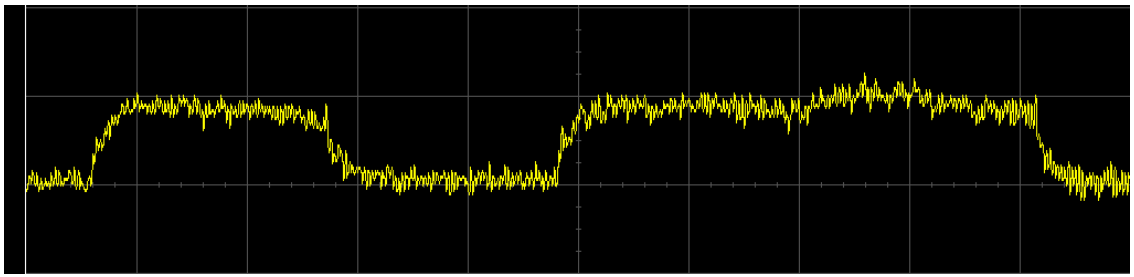
# Timing attacks notes and literature

- One of the earliest side-channel attacks due to easy measurements collection
- Can also be exploited remotely
- Exploits some not foreseen effects of caches to crypto implementations
- Applied to symmetric and asymmetric cryptography

# Timing attacks notes and literature

- One of the earliest side-channel attacks due to easy measurements collection
- Can also be exploited remotely
- Exploits some not foreseen effects of caches to crypto implementations
- Applied to symmetric and asymmetric cryptography
- P. Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”, CRYPTO 1996
- Daniel Page. “Theoretical use of cache memory as a cryptanalytic side-channel”, technical report CSTR-02-003, 2002.
  `https://eprint.iacr.org/2002/169.pdf`
- Daniel J. Bernstein. “Cache-timing attacks on AES.”
  `https://cr.yp.to/antiforgery/cachetiming-20050414.pdf`

# Power side-channel



# Power side channel: CMOS leakage

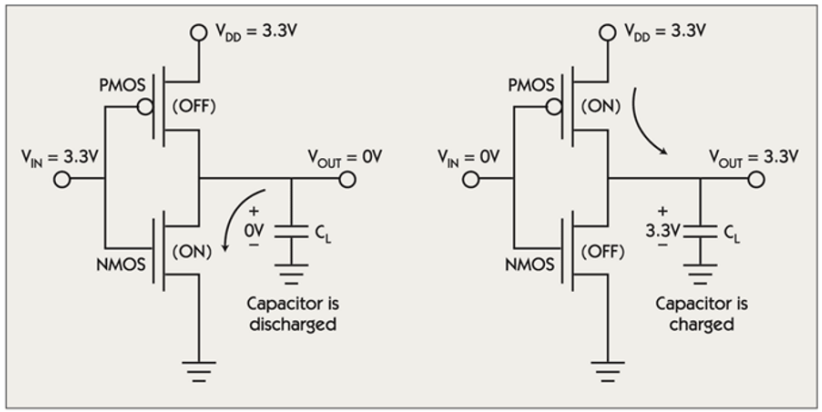



**Fig. 1** Dynamic power in a CMOS inverter.

- CMOS is the most popular technology and the CMOS circuits exhibit several types of leakage

# Power side channel: CMOS leakage

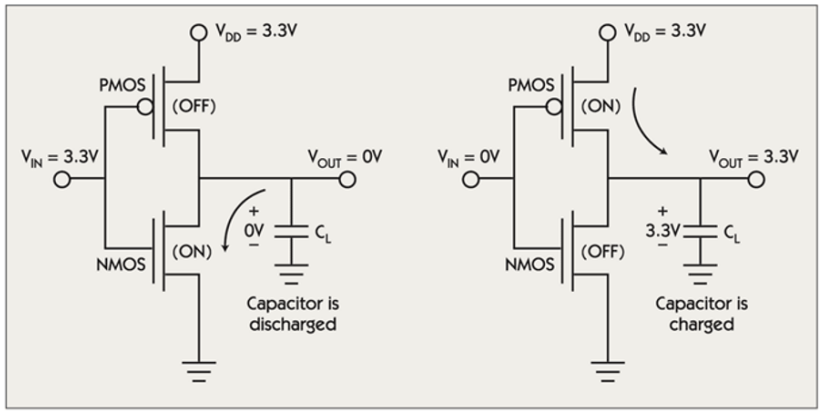



**Fig. 1** Dynamic power in a CMOS inverter.

- CMOS is the most popular technology and the CMOS circuits exhibit several types of leakage
- The most relevant for side-channel attacks is the charge and discharge of the CMOS load capacitance a.k.a dynamic power consumption

# Power side channel: CMOS leakage

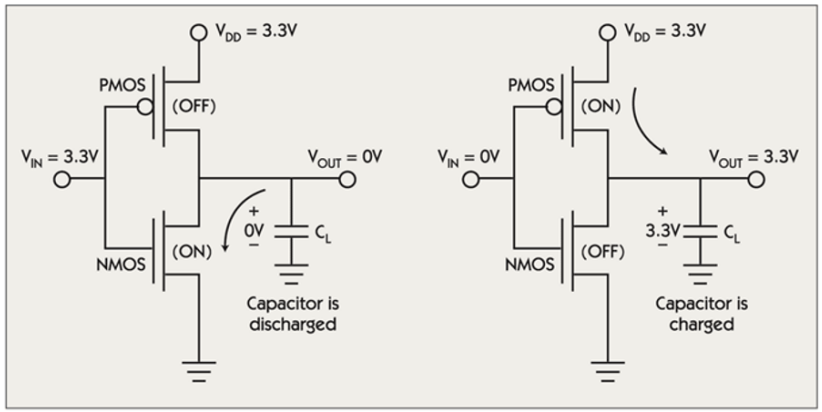



**Fig. 1** Dynamic power in a CMOS inverter.

- CMOS is the most popular technology and the CMOS circuits exhibit several types of leakage
- The most relevant for side-channel attacks is the charge and discharge of the CMOS load capacitance a.k.a dynamic power consumption
- Power analysis attack exploits the fact that the dynamic power consumption depends on the data and instructions being processed

# Power side channel: CMOS leakage

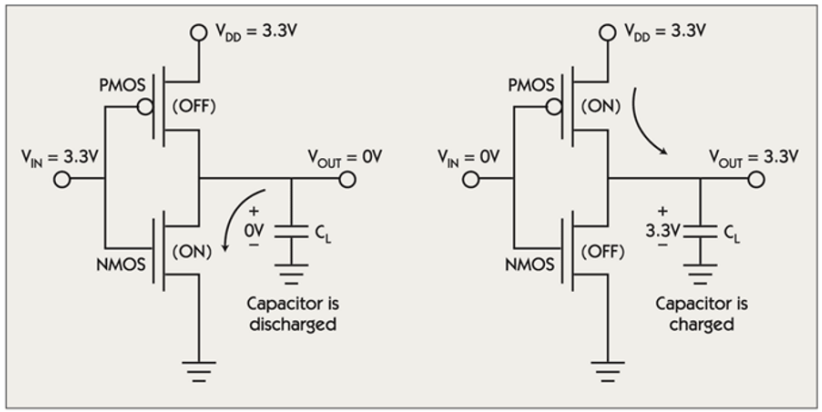



**Fig. 1** Dynamic power in a CMOS inverter.

- CMOS is the most popular technology and the CMOS circuits exhibit several types of leakage
- The most relevant for side-channel attacks is the charge and discharge of the CMOS load capacitance a.k.a dynamic power consumption
- Power analysis attack exploits the fact that the dynamic power consumption depends on the data and instructions being processed
- Dynamic power consumption ($P_{dyn}$) is produced by CMOS transitions from state 0 to 1 and from state 1 to 0

# Power side channel: CMOS leakage

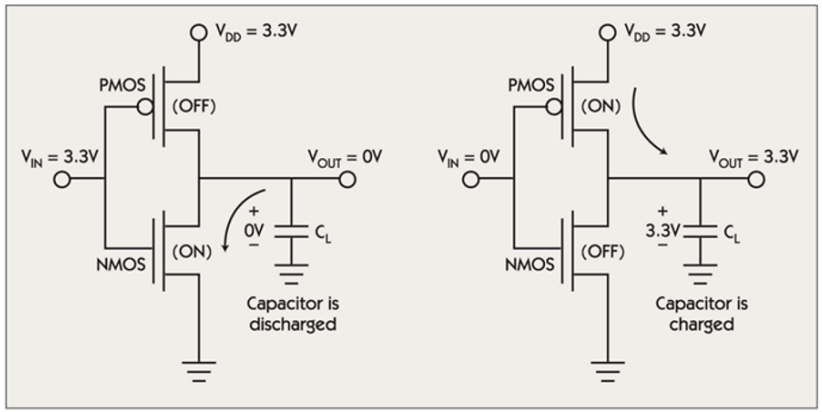



**Fig. 1** Dynamic power in a CMOS inverter.

- CMOS is the most popular technology and the CMOS circuits exhibit several types of leakage
- The most relevant for side-channel attacks is the charge and discharge of the CMOS load capacitance a.k.a dynamic power consumption
- Power analysis attack exploits the fact that the dynamic power consumption depends on the data and instructions being processed
- Dynamic power consumption ($P_{dyn}$) is produced by CMOS transitions from state 0 to 1 and from state 1 to 0
- $P_{dyn} = CV_{DD}^2 P_{0\to1} f$,
  where $C$ the transistor capacitance, $V_{DD}$ the power supply voltage, $f$ the frequency and $P_{0\to1}$ the probability of a $0 \to 1$ transition

# Power side channel: CMOS leakage

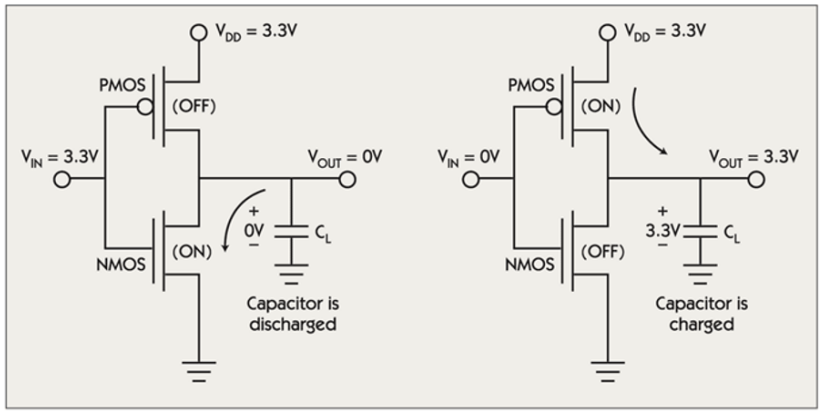



**Fig. 1** Dynamic power in a CMOS inverter.

- CMOS is the most popular technology and the CMOS circuits exhibit several types of leakage
- The most relevant for side-channel attacks is the charge and discharge of the CMOS load capacitance a.k.a dynamic power consumption
- Power analysis attack exploits the fact that the dynamic power consumption depends on the data and instructions being processed
- Dynamic power consumption ($P_{dyn}$) is produced by CMOS transitions from state 0 to 1 and from state 1 to 0
- $P_{dyn} = CV_{DD}^2 P_{0\to1} f$,
  where $C$ the transistor capacitance, $V_{DD}$ the power supply voltage, $f$ the frequency and $P_{0\to1}$ the probability of a $0 \to 1$ transition

# Power side channel: Modeling the leakage

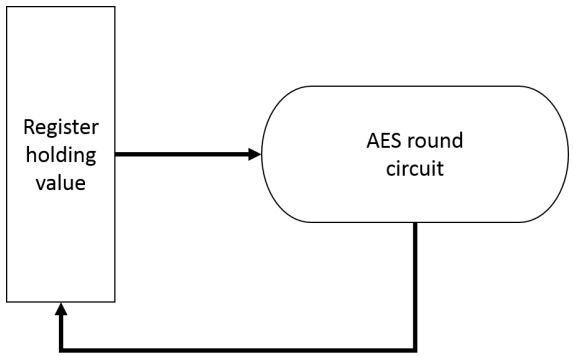
- Starting point: we use the number of transitions to model the leakage

# Power side channel: Modeling the leakage

- Starting point: we use the number of transitions to model the leakage
- The Hamming distance model counts the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions

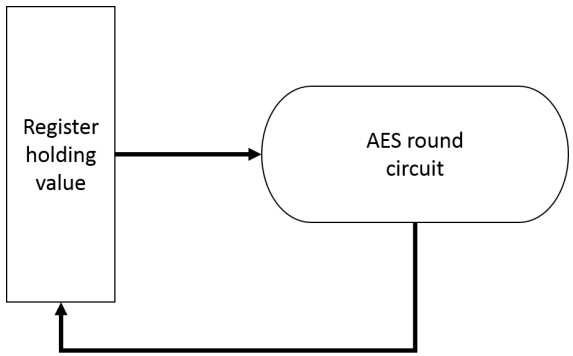
# Power side channel: Modeling the leakage

- Starting point: we use the number of transitions to model the leakage
- The Hamming distance model counts the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions
- Example 1: A register $R$ is storing the result of an AES round and initial value $v_0$ gets overwritten with $v_1$



# Power side channel: Modeling the leakage

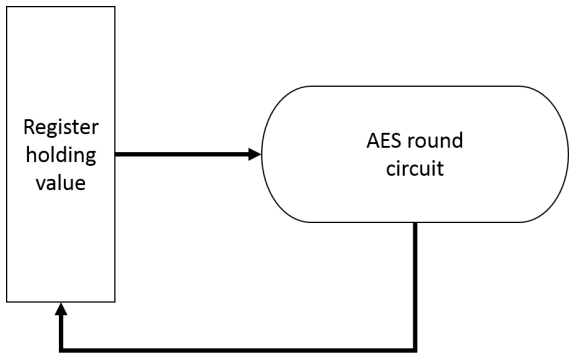
- Starting point: we use the number of transitions to model the leakage
- The Hamming distance model counts the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions
- Example 1: A register $R$ is storing the result of an AES round and initial value $v_0$ gets overwritten with $v_1$




- The power consumption because of the register transition $v_0 \rightarrow v_1$ is related to the number of bit flips that occurred

# Power side channel: Modeling the leakage

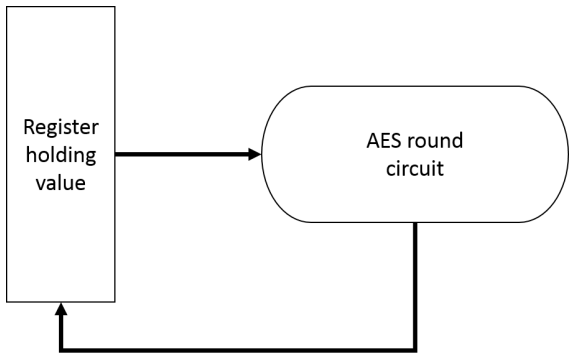
- Starting point: we use the number of transitions to model the leakage
- The Hamming distance model counts the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions
- Example 1: A register $R$ is storing the result of an AES round and initial value $v_0$ gets overwritten with $v_1$




- The power consumption because of the register transition $v_0 \rightarrow v_1$ is related to the number of bit flips that occurred
- Thus it can be modeled as HammingDistance$(v_0, v_1)$ = HammingWeight$(v_0 \oplus v_1)$

# Power side channel: Modeling the leakage

- Starting point: we use the number of transitions to model the leakage
- The Hamming distance model counts the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions
- Example 1: A register $R$ is storing the result of an AES round and initial value $v_0$ gets overwritten with $v_1$

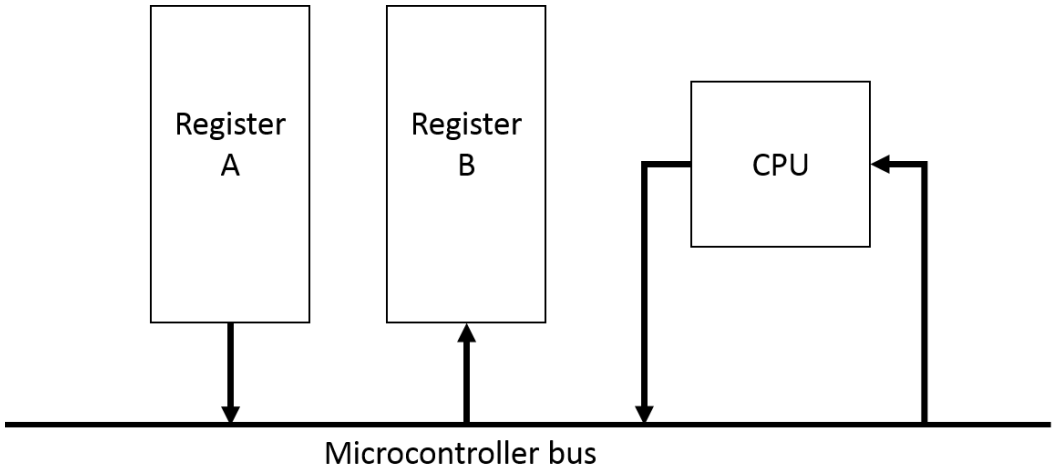



- The power consumption because of the register transition $v_0 \rightarrow v_1$ is related to the number of bit flips that occurred
- Thus it can be modeled as HammingDistance$(v_0, v_1)$ = HammingWeight$(v_0 \oplus v_1)$
- Common leakage model for hardware implementations (FPGA, ASIC)

# Power side channel: Modeling the leakage

- Example 2: In a microcontroller, a register $A$ contains value $v_0$ and an assembly instruction moves the content of register $A$ to $B$

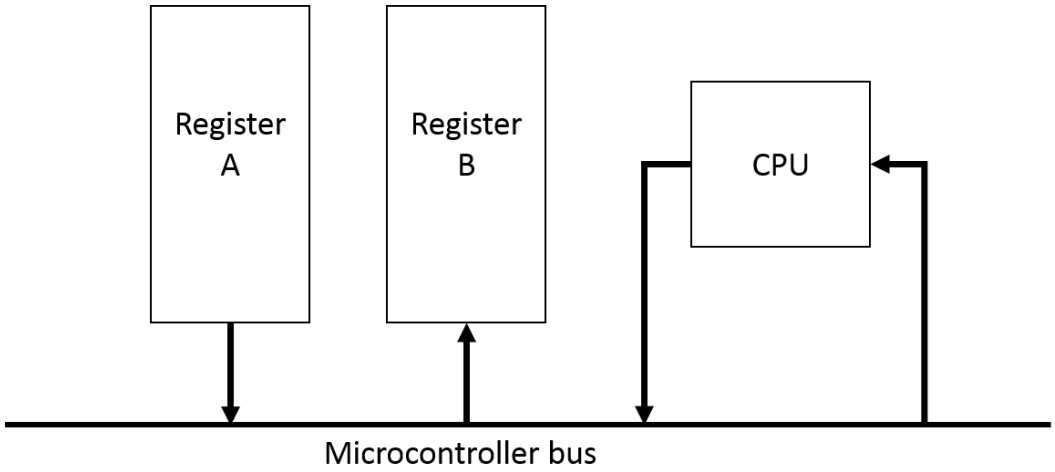
```
mov rB, rA
```



# Power side channel: Modeling the leakage

- Example 2: In a microcontroller, a register $A$ contains value $v_0$ and an assembly instruction moves the content of register $A$ to $B$

```
mov rB, rA
```

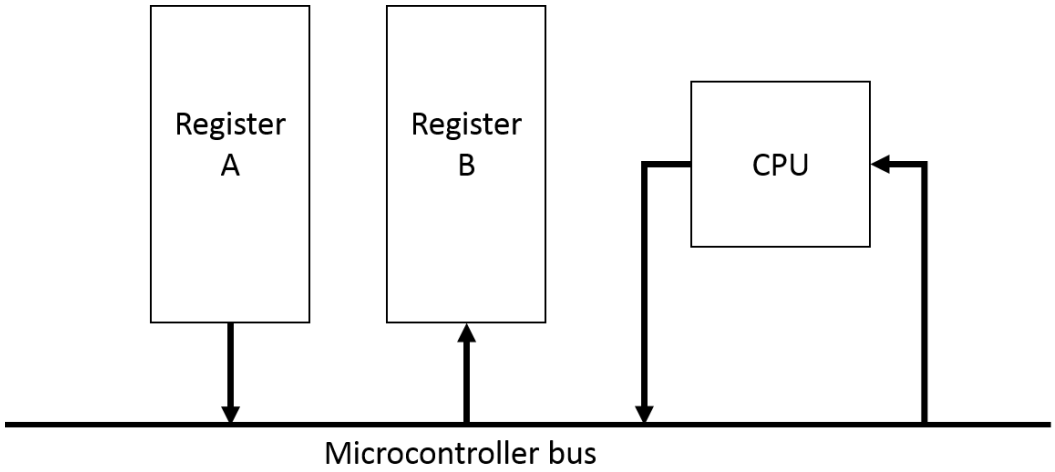



- This instruction transfers $v_0$ from $A$ to $B$ via the CPU, using the bus

# Power side channel: Modeling the leakage

- Example 2: In a microcontroller, a register $A$ contains value $v_0$ and an assembly instruction moves the content of register $A$ to $B$

```
mov rB, rA
```

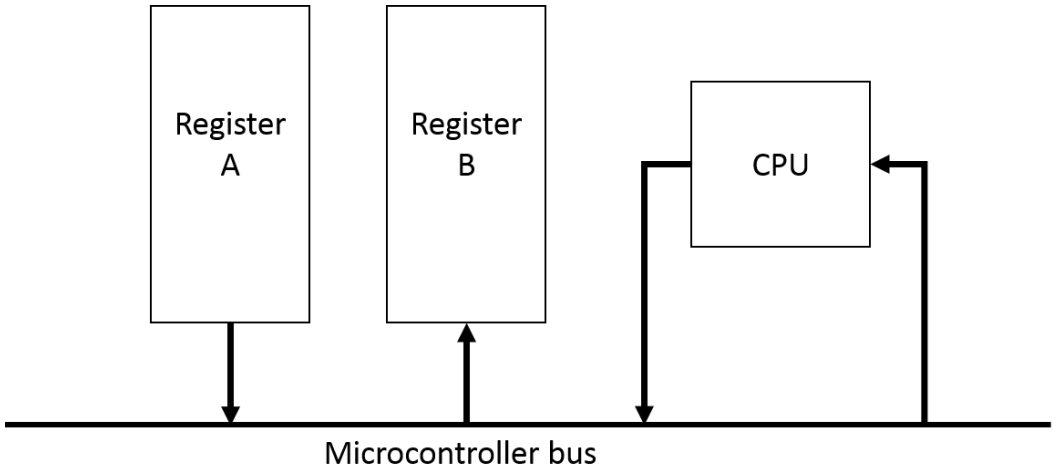



- This instruction transfers $v_0$ from $A$ to $B$ via the CPU, using the bus
- Typically the bus is precharged at all bits being zeros or one (busInitialValue)

# Power side channel: Modeling the leakage

- Example 2: In a microcontroller, a register $A$ contains value $v_0$ and an assembly instruction moves the content of register $A$ to $B$

```
mov rB, rA
```

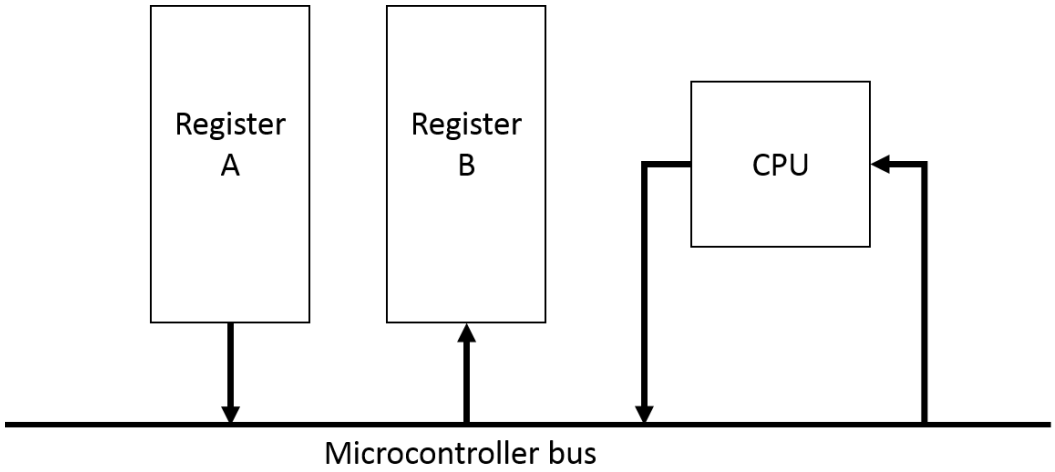



- This instruction transfers $v_0$ from $A$ to $B$ via the CPU, using the bus
- Typically the bus is precharged at all bits being zeros or one (busInitialValue)
- The power consumption of the instruction can be modeled as HammingDistance(busInitialValue,$v_0$) = HammingWeight($v_0 \oplus 0$) = HW($v_0$)

# Power side channel: Modeling the leakage

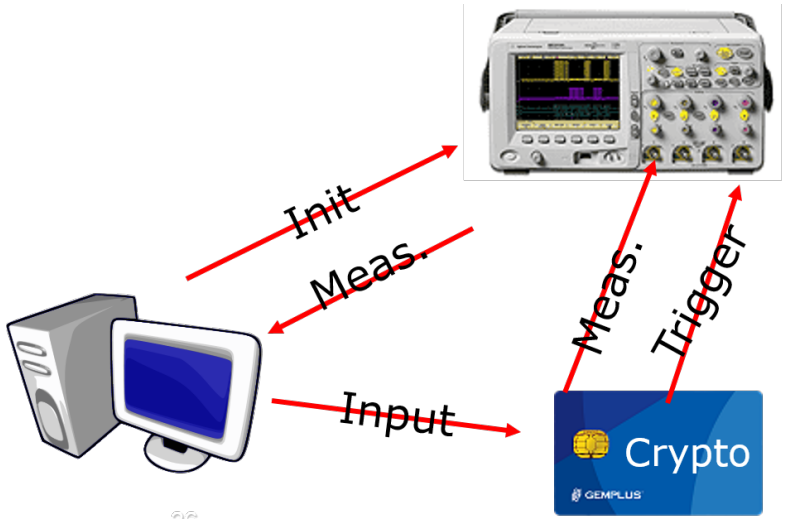
- Example 2: In a microcontroller, a register $A$ contains value $v_0$ and an assembly instruction moves the content of register $A$ to $B$

```
mov rB, rA
```




- This instruction transfers $v_0$ from $A$ to $B$ via the CPU, using the bus
- Typically the bus is precharged at all bits being zeros or one (busInitialValue)
- The power consumption of the instruction can be modeled as HammingDistance(busInitialValue,$v_0$) = HammingWeight($v_0 \oplus 0$) = HW($v_0$)
- Common leakage model for software implementations (AVR/ARM)

# Measurement setup schematics

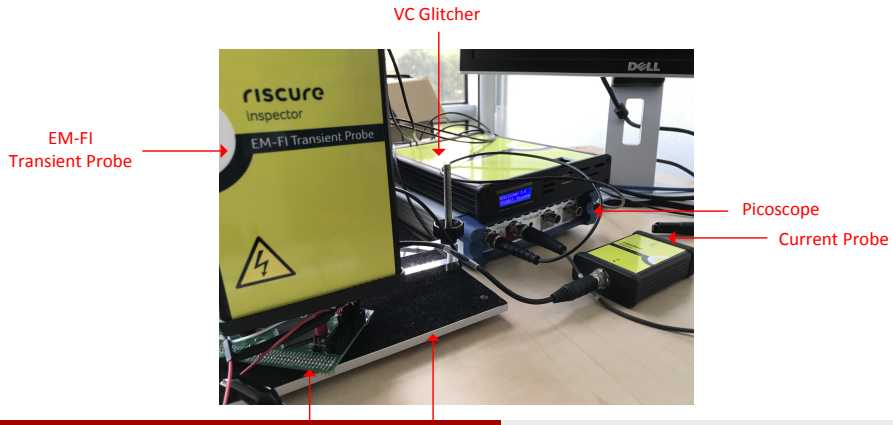



26

- Usually power measurements requires physical proximity to the device and customized measurement equipment (resistor, oscilloscope)

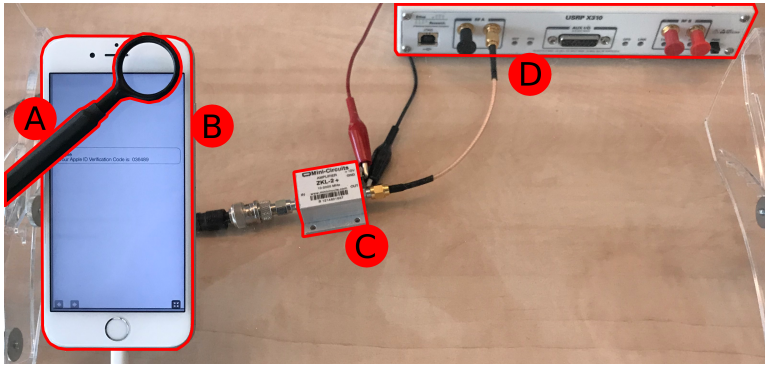
# Actual setups

## FA setup

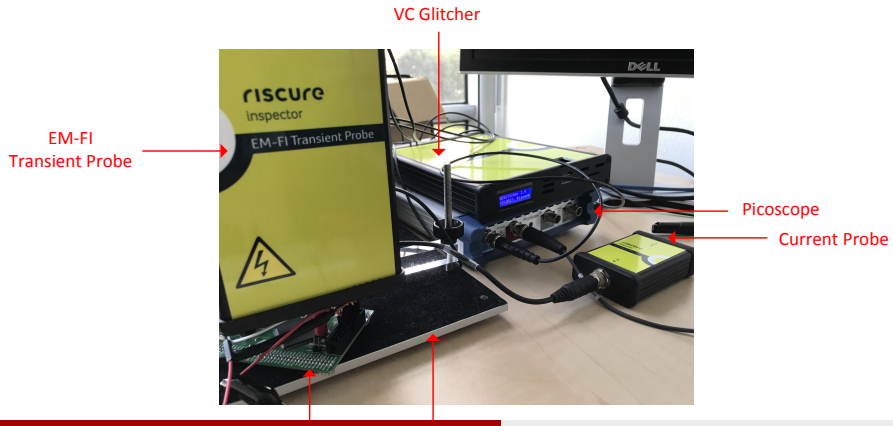


# Actual setups

## Tempest

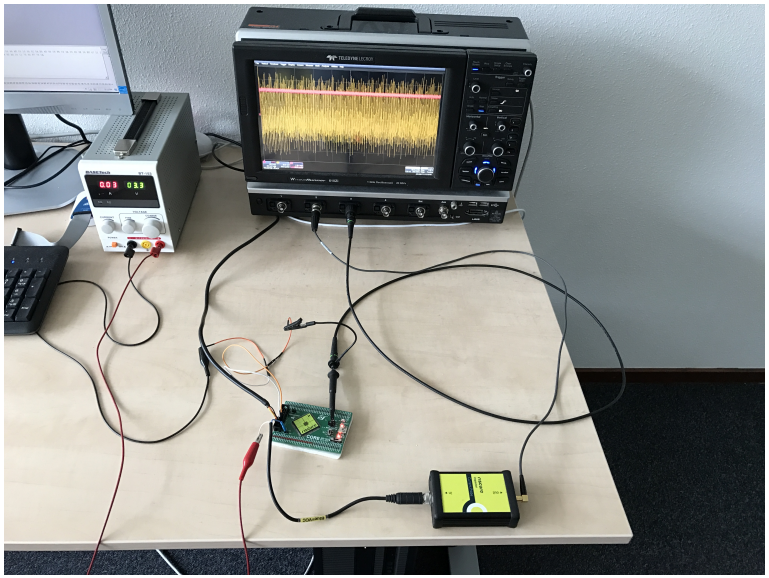



## FA setup

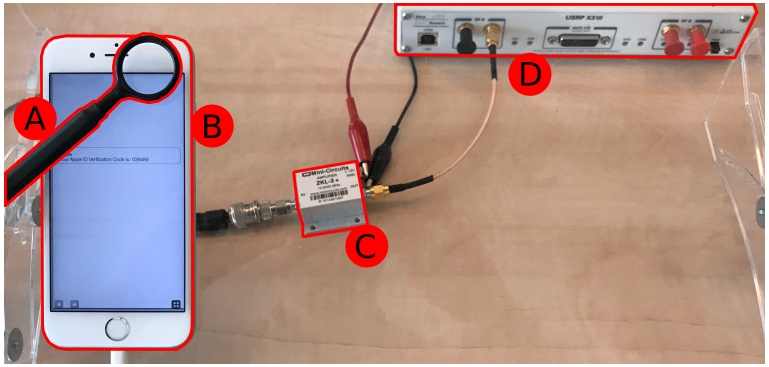


# Actual setups

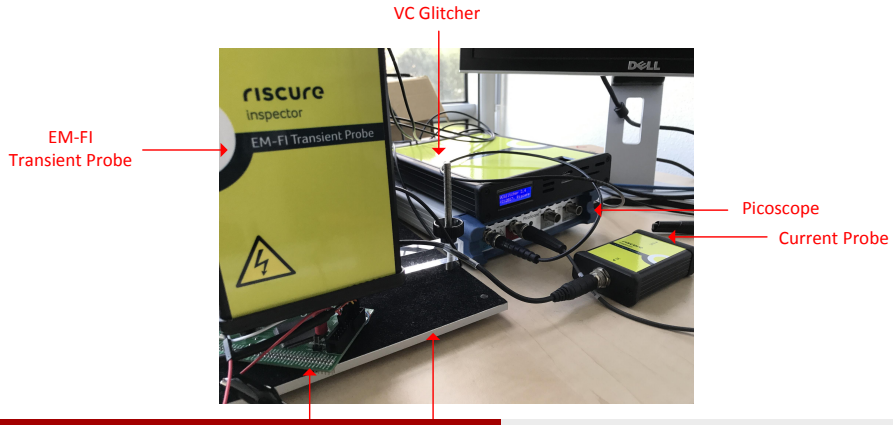
## DPA setup with ARM CortexM4



## Tempest

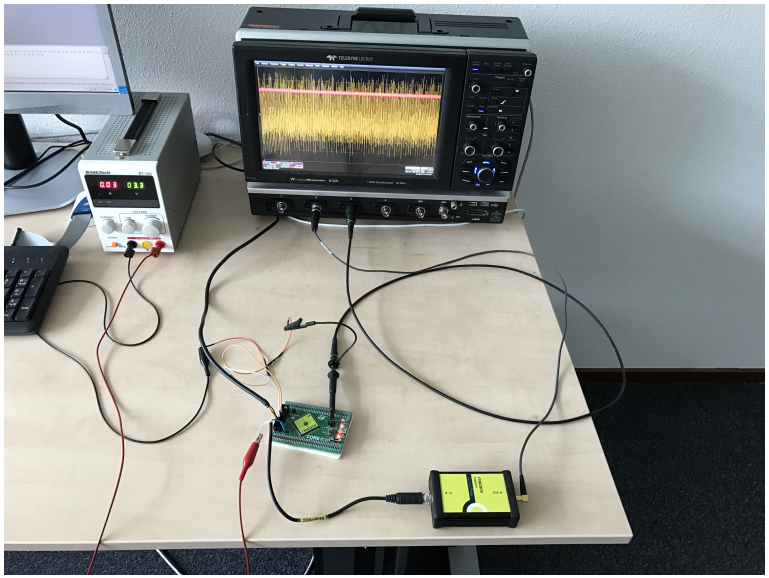



## FA setup

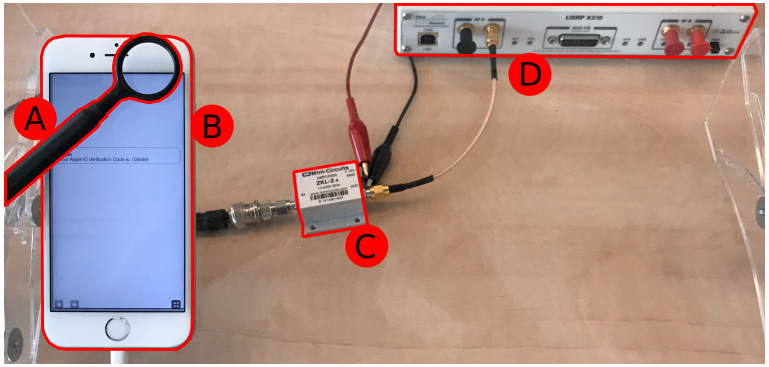


# Actual setups

## DPA setup with ARM CortexM4

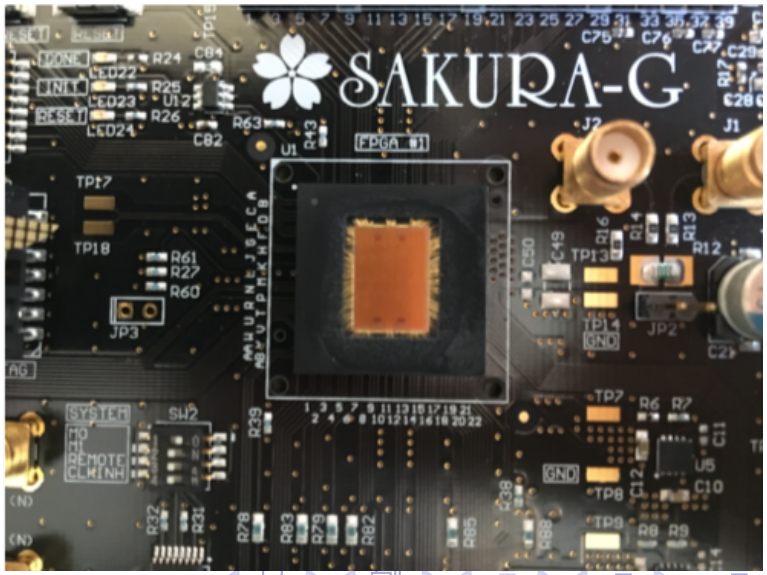


## Tempest

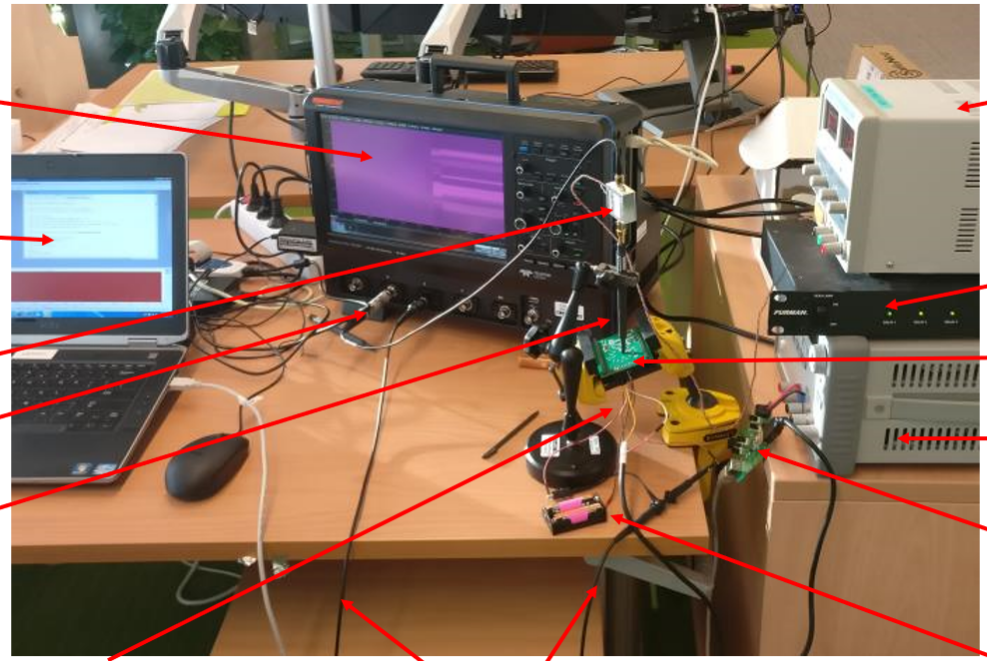



## FA setup


## FPGA board for SCA



# Actual (overcomplicated?) setup



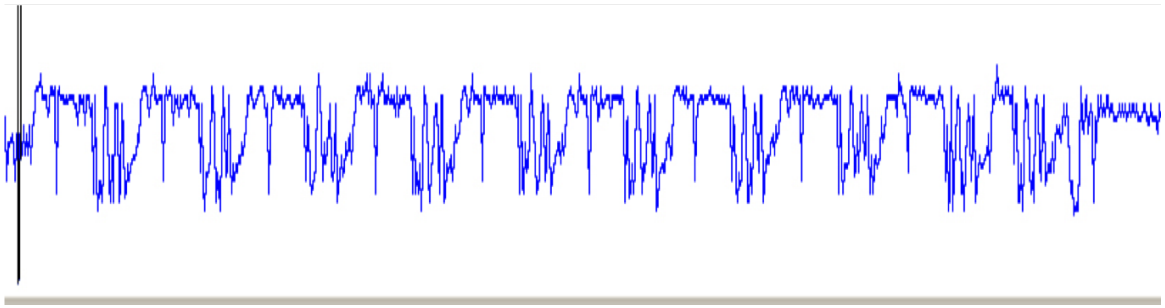
# Analysis capabilities

- Simple Power Analysis (SPA): one or a few measurements - visual inspection or some simple signal processing
- Differential attacks (DPA): multiple measurements - use of statistics, signal processing, etc.
- Higher order attacks: Univariate vs multivariate
- Profiled attacks: more in the lecture on Template attacks
- Combining two or more side-channels
- Combining side-channel attack with theoretical cryptanalysis

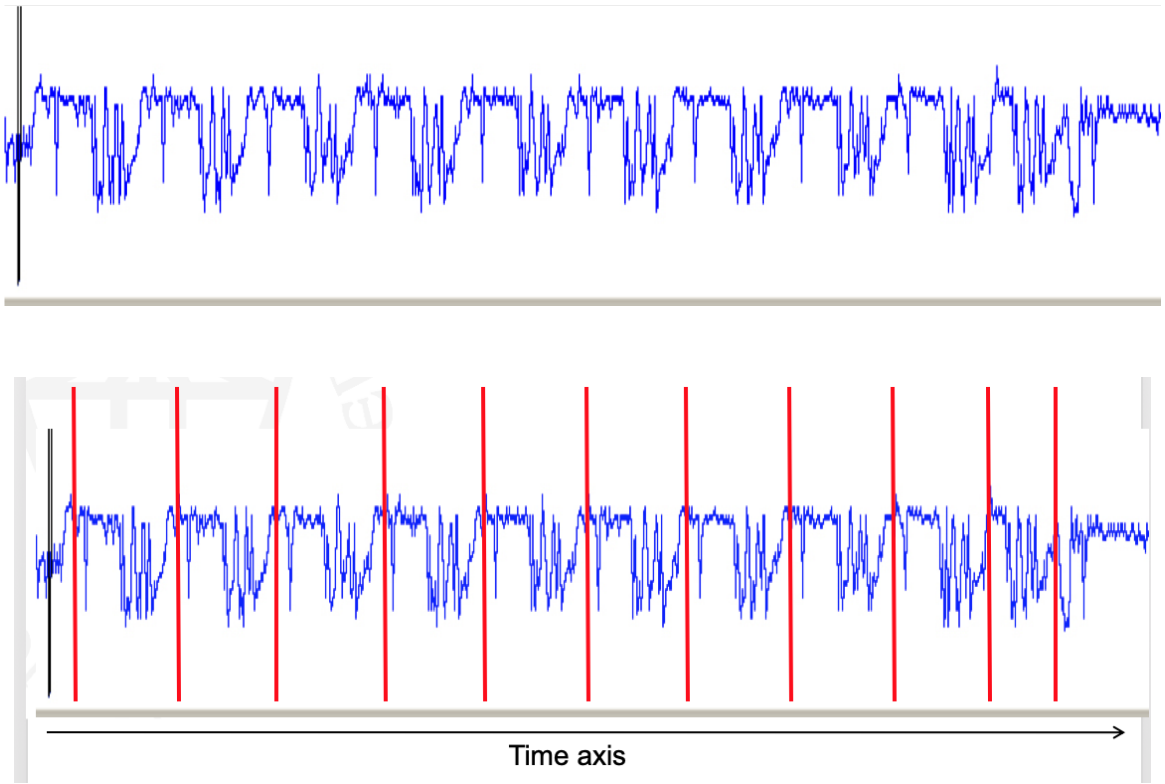
# Simple Power Analysis (SPA)

- Based on one or a few measurements
- Mostly discovery of data-(in)dependent but instruction-dependent properties e.g.
  - Symmetric:
    - Number of rounds (resp. key length)
    - Memory accesses (usually higher power consumption)
  - Asymmetric:
    - The key (if badly implemented, e.g. RSA / ECC)
    - Key length
    - Implementation details: for example RSA w/wo CRT

# SPA example 1



# SPA example 1



# SPA on RSA (Square-and-Multiply)

- RSA modular exponentiation

```
Input: integers x, e, n, length l of e
Output: x^e mod n

Process ModularExponentiation(x, e, n, l)

r=1;
for j=l-1 down to 0
   r=r^2 mod n //square
   if (bit j of e) == 1
      r= r*x mod n //multiply
end
return r;
EndProcess
```

# SPA on RSA (Square-and-Multiply)

- RSA modular exponentiation

```
Input: integers x, e, n, length l of e
Output: x^e mod n

Process ModularExponentiation(x, e, n, l)

r=1;
for j=l-1 down to 0
   r=r^2 mod n //square
   if (bit j of e) == 1
      r= r*x mod n //multiply
end
return r;
EndProcess
```

- Do you already see a timing attack?

# SPA on RSA (Square-and-Multiply)

- RSA modular exponentiation

```
Input: integers x, e, n, length l of e
Output: x^e mod n

Process ModularExponentiation(x, e, n, l)

r=1;
for j=l-1 down to 0
   r=r^2 mod n //square
   if (bit j of e) == 1
      r= r*x mod n //multiply
end
return r;
EndProcess
```

- Do you already see a timing attack?
- The exponent-dependent branch is causing it!

# SPA on RSA (Square-and-Multiply)

- RSA modular exponentiation

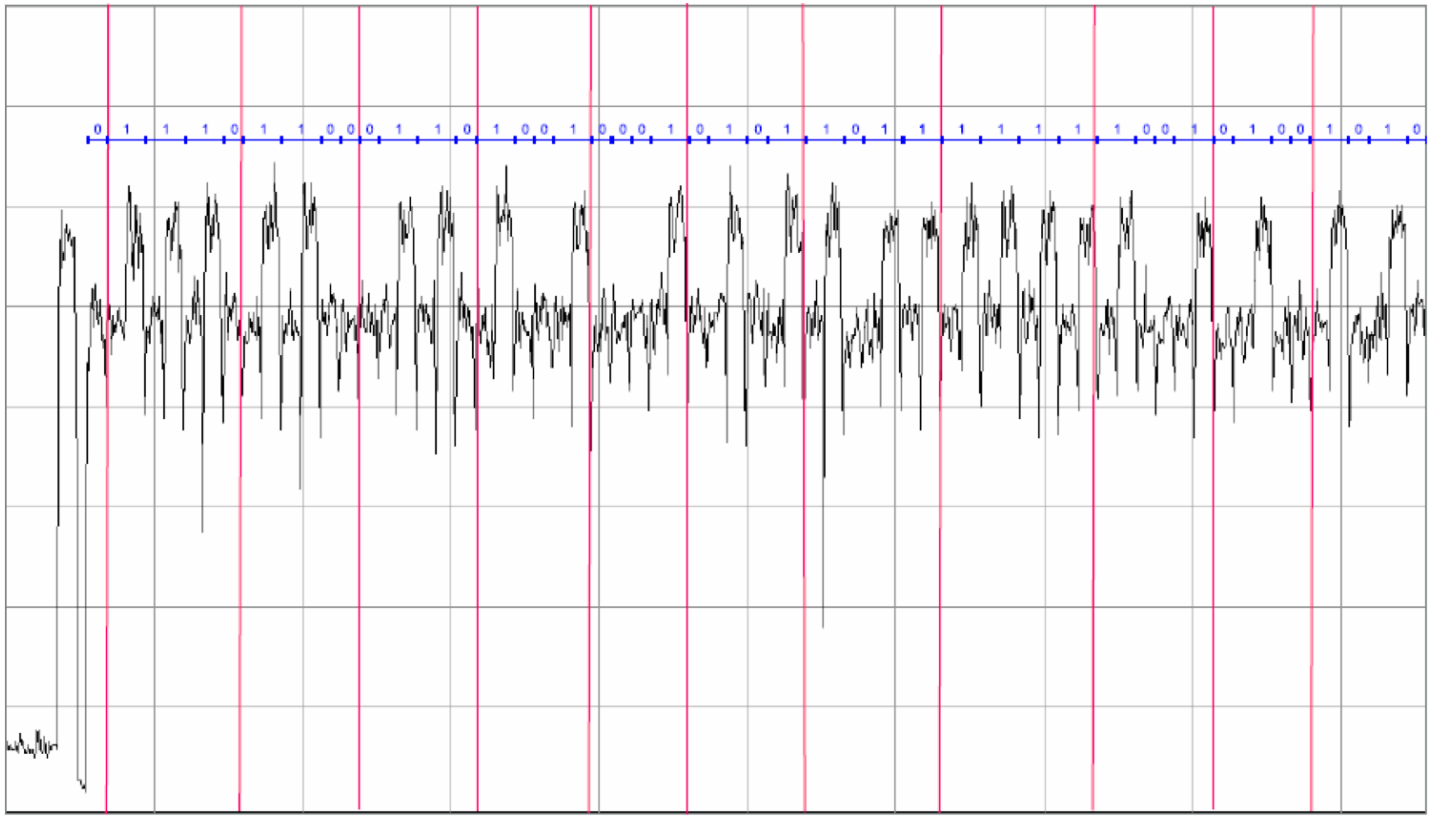
```
Input: integers x, e, n, length l of e
Output: x^e mod n

Process ModularExponentiation(x, e, n, l)

r=1;
for j=l-1 down to 0
   r=r^2 mod n //square
   if (bit j of e) == 1
      r= r*x mod n //multiply
end
return r;
EndProcess
```

- Do you already see a timing attack?
- The exponent-dependent branch is causing it!
- Do you see another side-channel attack?

# SPA example 2: RSA exponentiation



# DPA: main concepts

The DPA assumption: the attack is possible assuming the existence of a sensitive variable (for which exhaustive key search is possible) that depends on something we know (msg) and something we want to learn (key).

# DPA: main concepts

The DPA assumption: the attack is possible assuming the existence of a sensitive variable (for which exhaustive key search is possible) that depends on something we know (msg) and something we want to learn (key).

Main steps

1. Choose your sensitive variable

# DPA: main concepts

The DPA assumption: the attack is possible assuming the existence of a sensitive variable (for which exhaustive key search is possible) that depends on something we know (msg) and something we want to learn (key).

Main steps

1. Choose your sensitive variable
2. Collect measurements, known plaintext/ciphertext, sub-key guesses

# DPA: main concepts

The DPA assumption: the attack is possible assuming the existence of a sensitive variable (for which exhaustive key search is possible) that depends on something we know (msg) and something we want to learn (key).

Main steps

1. Choose your sensitive variable
2. Collect measurements, known plaintext/ciphertext, sub-key guesses
3. Predict (hypothetical) intermediate values

# DPA: main concepts

The DPA assumption: the attack is possible assuming the existence of a sensitive variable (for which exhaustive key search is possible) that depends on something we know (msg) and something we want to learn (key).

Main steps

1. Choose your sensitive variable
2. Collect measurements, known plaintext/ciphertext, sub-key guesses
3. Predict (hypothetical) intermediate values
4. Decide on the leakage model

# DPA: main concepts

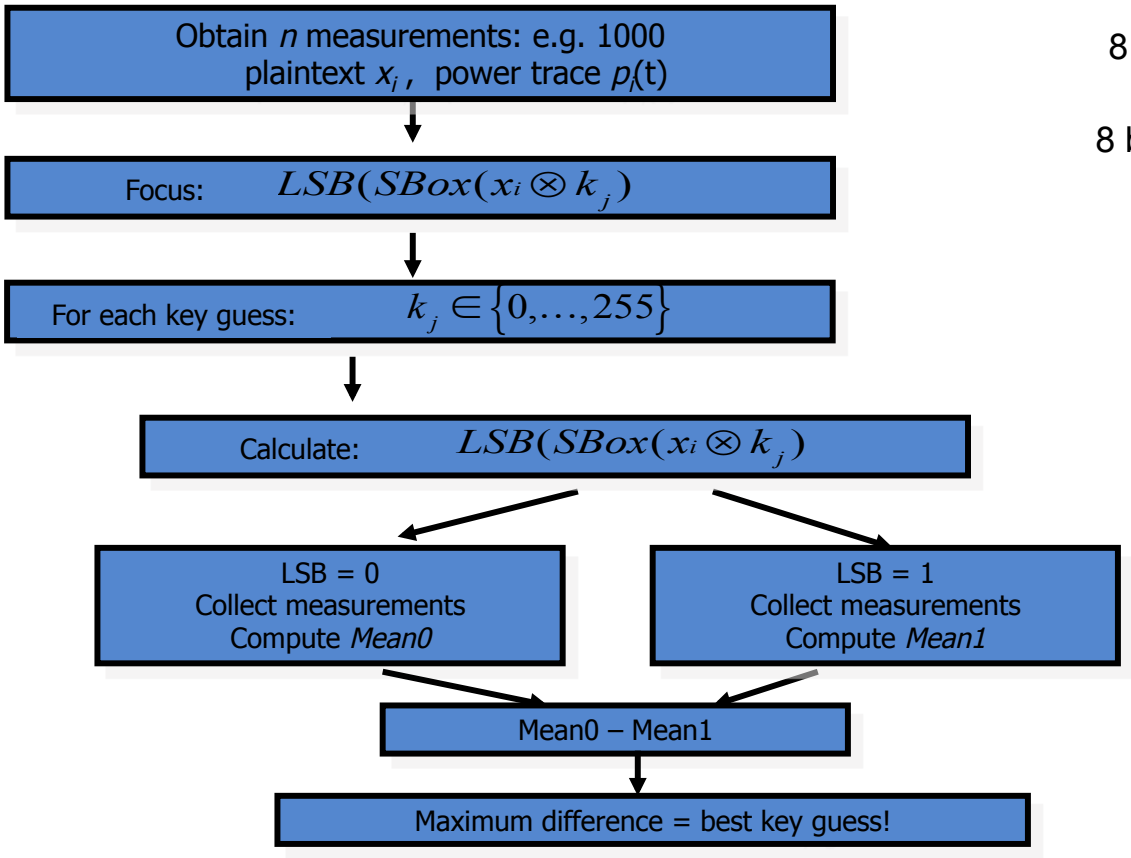
The DPA assumption: the attack is possible assuming the existence of a sensitive variable (for which exhaustive key search is possible) that depends on something we know (msg) and something we want to learn (key).

Main steps

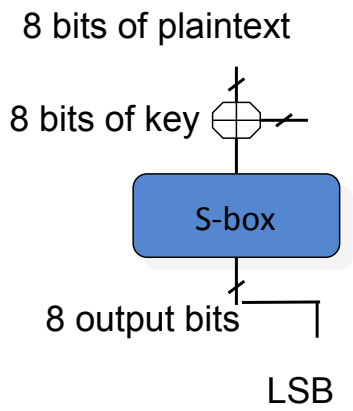
1. Choose your sensitive variable
2. Collect measurements, known plaintext/ciphertext, sub-key guesses
3. Predict (hypothetical) intermediate values
4. Decide on the leakage model
5. Recover the key by statistical means, using partition or comparison method as the side-channel distinguisher

# DPA with Distance of Means (DoM) as distinguisher

## Classical 1-bit DPA on AES using DoM




**AES impl.**




1000 measurements *
time window $t$ *
256 key guesses

[Kocher et al.]

# Power analysis notes and literature

- Very powerful attacks that require contact with the target

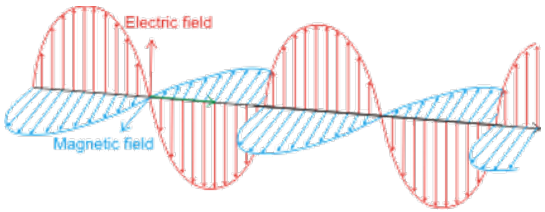
# Power analysis notes and literature

- Very powerful attacks that require contact with the target
- Countermeasures on different layers required i.e. algorithm, implementation, transistor

# Power analysis notes and literature

- Very powerful attacks that require contact with the target
- Countermeasures on different layers required i.e. algorithm, implementation, transistor
- P. Kocher, J. Jaffe, B. Jun. “Differential Power Analysis”, CRYPTO 1999.
- T. Eisenbarth et al. “On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoqCode Hopping Scheme”, CRYPTO 2008.
- Mangard et al. Power Analysis Attacks, Springer, 2006.
- T. Kasper et al. “All You Can Eat or Breaking a Real-World Contactless Payment System”, Financial Cryptography 2010.
- J. Balasch et al. “Power Analysis of Atmel CryptoMemory - Recovering Keys from Secure EEPROMs”, CT-RSA 2012.
- N. Samwel et al. “Breaking Ed25519 in WolfSSL.”, CT-RSA 2018.

# Electromagnetic side channel



# EM side channel: Probing

- Observing a power signal in more complex systems can be messy

# EM side channel: Probing

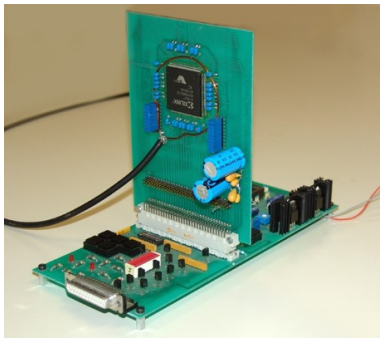
- Observing a power signal in more complex systems can be messy
- Complicated SoCs with multiple peripherals

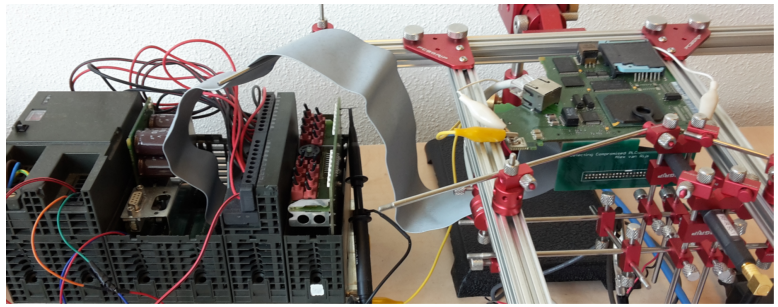
# EM side channel: Probing

- Observing a power signal in more complex systems can be messy
- Complicated SoCs with multiple peripherals
- Countermeasures trying to flatten the power consumption signal

# EM side channel: Probing

- Observing a power signal in more complex systems can be messy
- Complicated SoCs with multiple peripherals
- Countermeasures trying to flatten the power consumption signal
- Use an electromagnetic probe instead

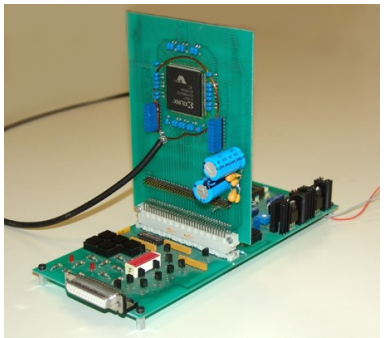


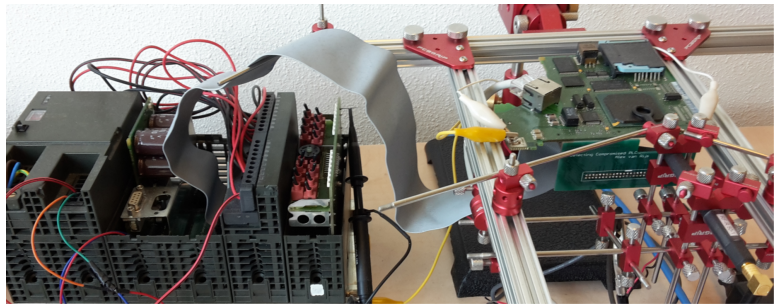


- A probe is used to access the power consumption with less board modifications

# EM side channel: Probing

- Observing a power signal in more complex systems can be messy
- Complicated SoCs with multiple peripherals
- Countermeasures trying to flatten the power consumption signal
- Use an electromagnetic probe instead

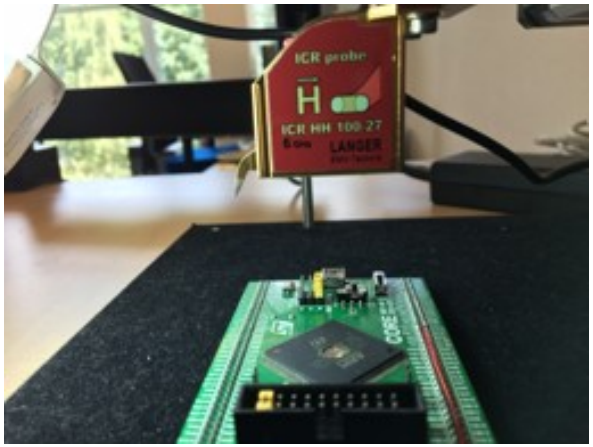




- A probe is used to access the power consumption with less board modifications
- Smaller probes can focus on interesting locations and ignore interference from unrelated el. components

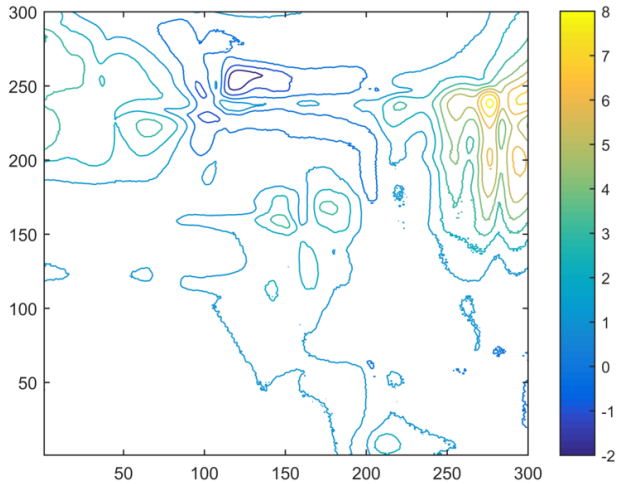
# EM side channel: Decapsulation and Microprobing

- To improve spatial resolution of analysis use a micrometer-sized antenna
- To exploit more leakage decapsulate the chip using chemicals




# EM side channel: Decapsulation and Microprobing

- Left: close inspection of decapsulated ARM processor using a microscope
- Right: EM emission heat-map of the same chip



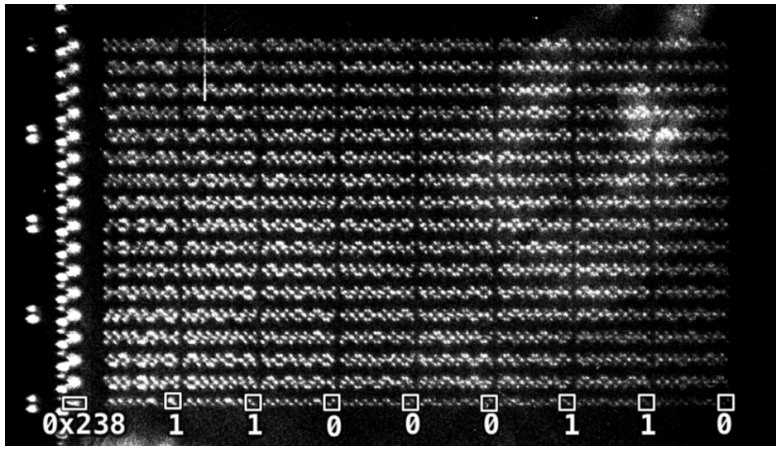
# EM side channel: notes and literature

- EM enables side-channel attacks both in high proximity scenarios and distance scenarios
- Main side channel for SoCs, FPGAs, contactless cards due to their complexity and communication methods
- TEMPEST-like attacks are also targeting private data and authentication methods such as code etc.
- Gandolfi et al.: Electromagnetic Analysis: Concrete Results, CHES 1999.
- Andrikos et al.: Location, location, location: Revisiting modeling and exploitation for location-based side channel leakages, ASIACRYPT2019

# Exotic side channels

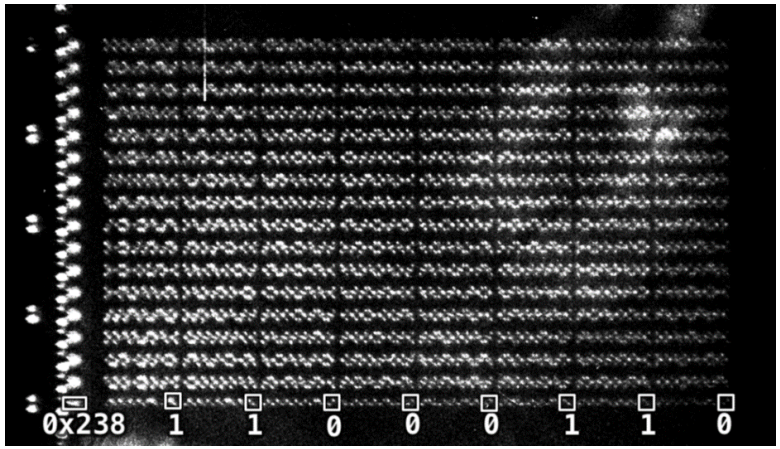
## Exotic side channels

# Optical Emission




- Accessing the chip SRAM cells emits photons that can be detected by a high-resolution camera
- Visual inspection can reveal the memory location accessed

# Optical Emission

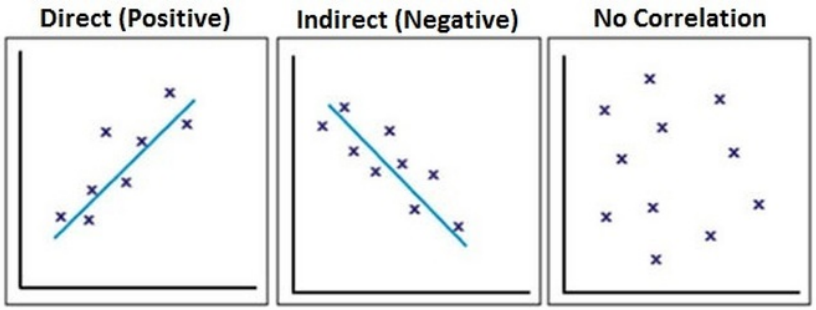



- Accessing the chip SRAM cells emits photons that can be detected by a high-resolution camera
- Visual inspection can reveal the memory location accessed
- The memory location maps to a specific value (e.g. in the AES LUT), i.e. it maps directly to $Sbox(in \oplus key)$
- Since the input *in* is known, knowledge of the memory location reveals the key
- Schlösser et al.: Simple Photonic Emission Analysis of AES, CHES 2012.

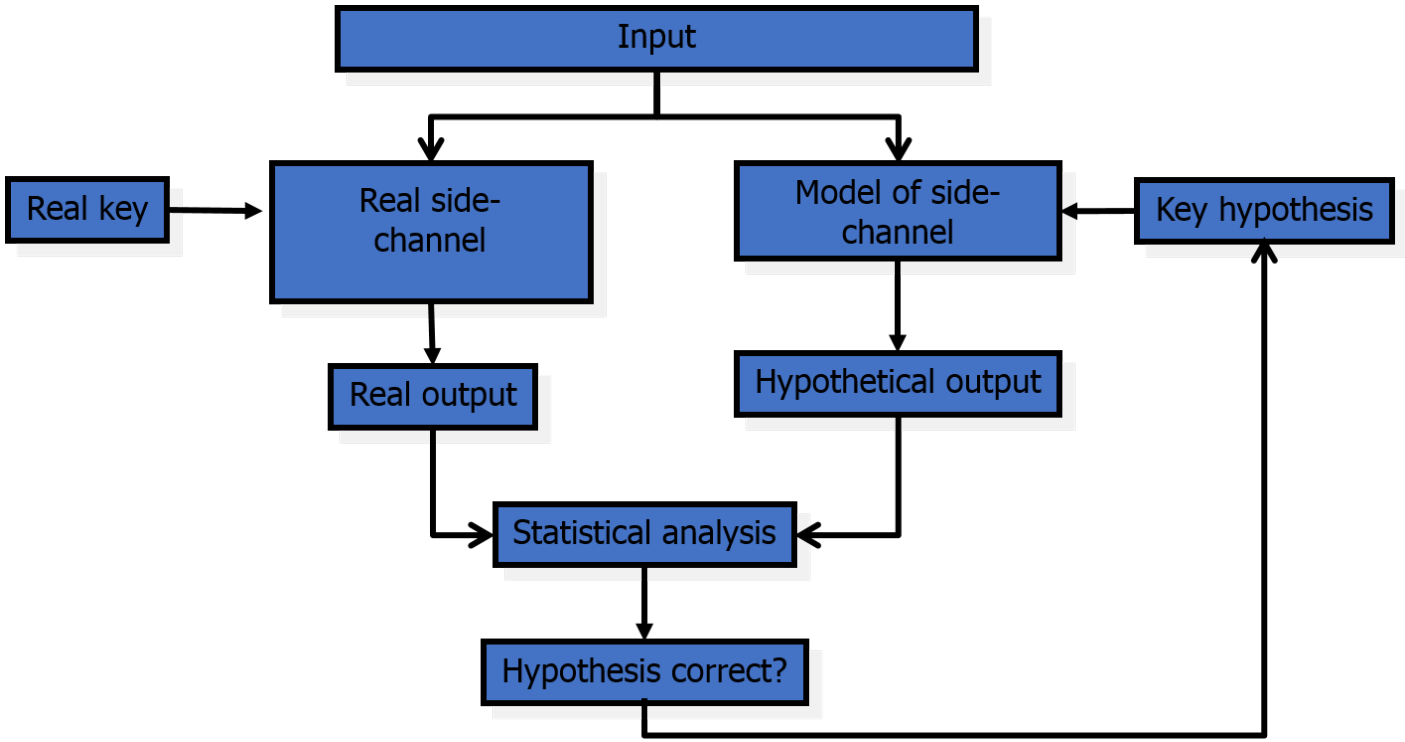
# Out-of-order and speculative execution


- Meltdown and Spectre Attacks
- Kernel addresses were access unintentionally due to out-of-order execution
- Taking all possible branches may also cause issues
- Foreshadow as a variant of Meltdown on SGX
- Recent ones: RIDL, ZombieLoad, ...

# Correlation Power Analysis (CPA)

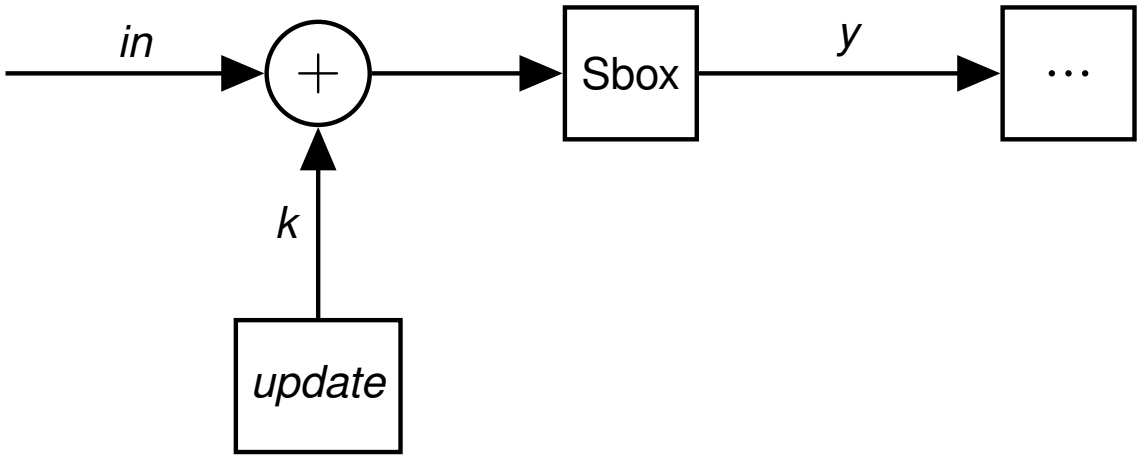


# CPA: the main principle



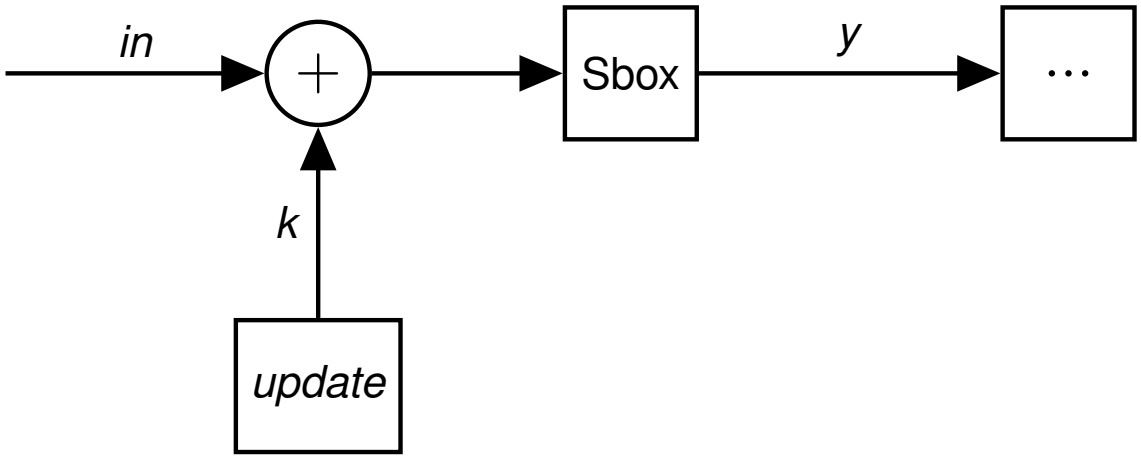

- Brier et al.: Correlation Power Analysis with a Leakage Model, CHES 2004

# CPA Step 1: Choose intermediate value and decide on the leakage model



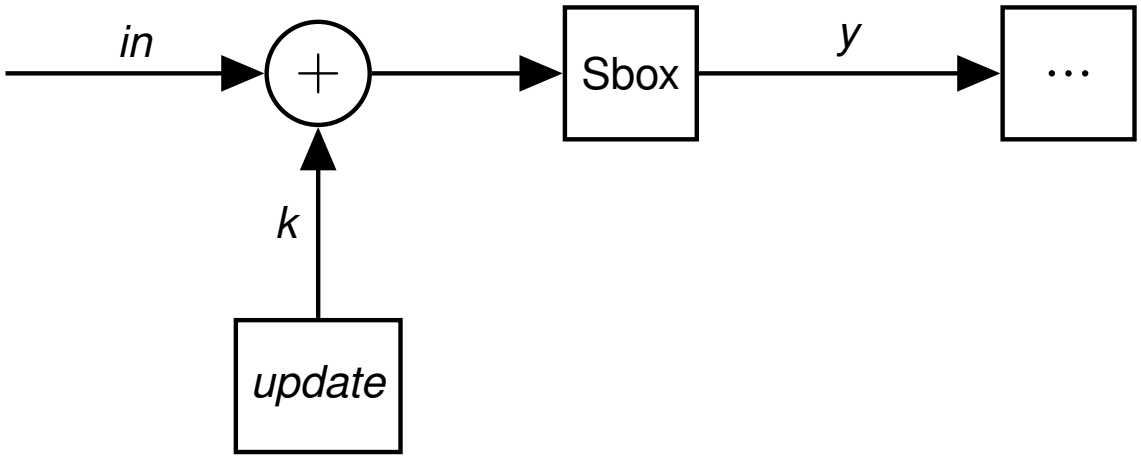

- Assume a software AES implementation e.g. for AVR microcontrollers (8-bit arch.)

# CPA Step 1: Choose intermediate value and decide on the leakage model



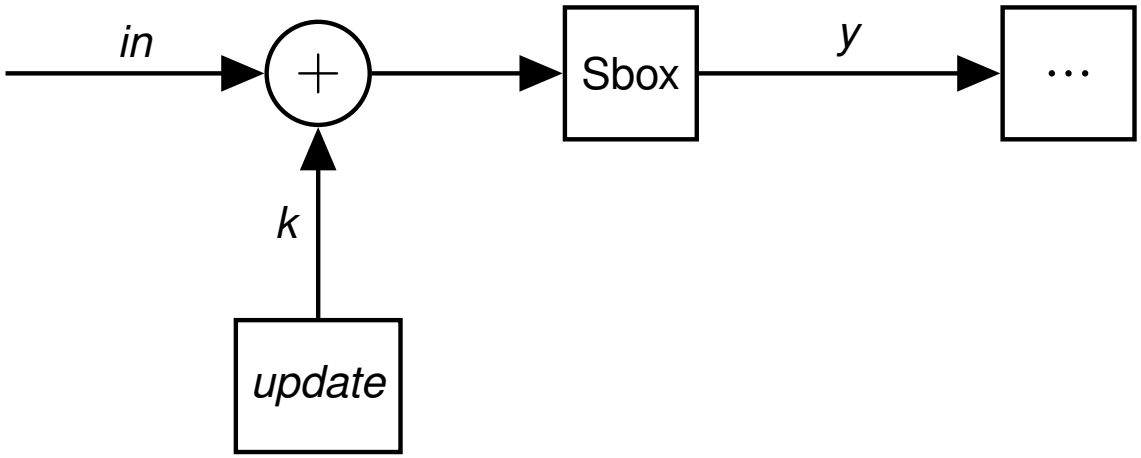

- Assume a software AES implementation e.g. for AVR microcontrollers (8-bit arch.)
- Step 1: Choose an intermediate value $v$ of the AES cipher to attack

# CPA Step 1: Choose intermediate value and decide on the leakage model



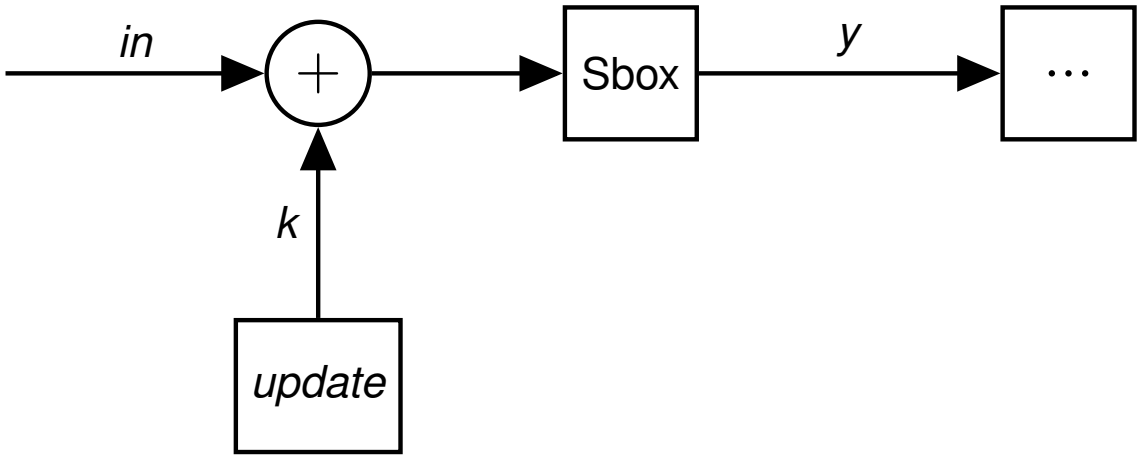

- Assume a software AES implementation e.g. for AVR microcontrollers (8-bit arch.)
- Step 1: Choose an intermediate value $v$ of the AES cipher to attack
- The value $v$ must be a function of the input and the key, i.e. $v = f(in, k)$

# CPA Step 1: Choose intermediate value and decide on the leakage model



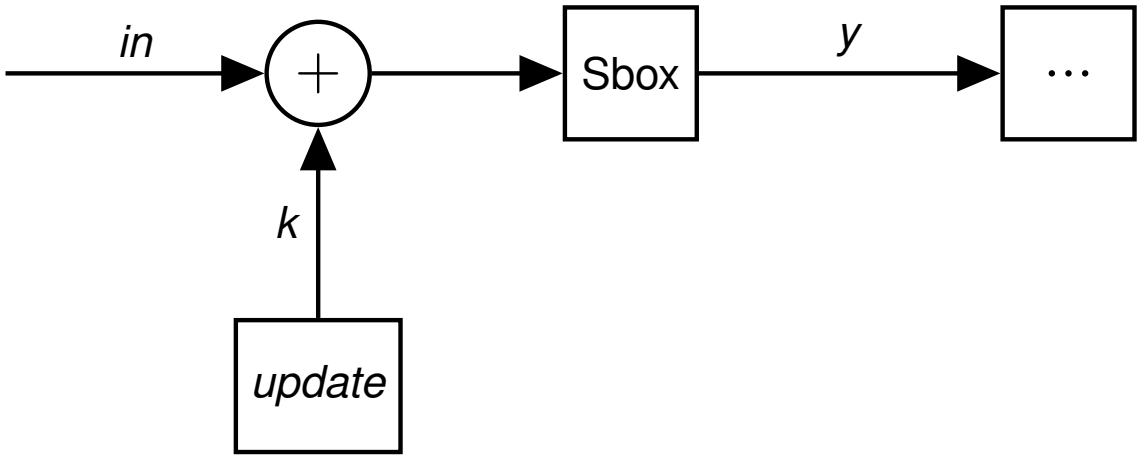

- Assume a software AES implementation e.g. for AVR microcontrollers (8-bit arch.)
- Step 1: Choose an intermediate value $v$ of the AES cipher to attack
- The value $v$ must be a function of the input and the key, i.e. $v = f(in, k)$
- A common choice for $v$ is the Sbox output, i.e. $v = y = Sbox(in \oplus k)$

# CPA Step 1: Choose intermediate value and decide on the leakage model




- Assume a software AES implementation e.g. for AVR microcontrollers (8-bit arch.)
- Step 1: Choose an intermediate value $v$ of the AES cipher to attack
- The value $v$ must be a function of the input and the key, i.e. $v = f(in, k)$
- A common choice for $v$ is the Sbox output, i.e. $v = y = Sbox(in \oplus k)$
- Throughout the attack the key $k$ must remain constant

# CPA Step 1: Choose intermediate value and decide on the leakage model




- Assume a software AES implementation e.g. for AVR microcontrollers (8-bit arch.)
- Step 1: Choose an intermediate value $v$ of the AES cipher to attack
- The value $v$ must be a function of the input and the key, i.e. $v = f(in, k)$
- A common choice for $v$ is the Sbox output, i.e. $v = y = Sbox(in \oplus k)$
- Throughout the attack the key $k$ must remain constant
- Throughout the attack the input $in$ is random

# CPA Step 2: Measure the power consumption

- In Step 2 we record power consumption traces for multiple random inputs

# CPA Step 2: Measure the power consumption

- In Step 2 we record power consumption traces for multiple random inputs
- Generate randomly $n$ 8-bit inputs. Typically $n$ is large (thousands to millions!)

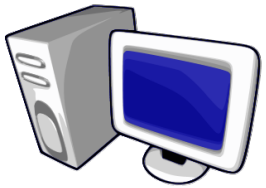
# CPA Step 2: Measure the power consumption

- In Step 2 we record power consumption traces for multiple random inputs
- Generate randomly $n$ 8-bit inputs. Typically $n$ is large (thousands to millions!)
- Store the inputs in vector $\mathbf{in} = [in_1\, in_2\, in_3 \ldots in_n]^\intercal$

# CPA Step 2: Measure the power consumption

- In Step 2 we record power consumption traces for multiple random inputs
- Generate randomly $n$ 8-bit inputs. Typically $n$ is large (thousands to millions!)
- Store the inputs in vector $\mathbf{in} = [in_1\, in_2\, in_3 \ldots in_n]^\intercal$

| 5 |
|---|
| 220 |
| 56 |
| 121 |
| 12 |
| 42 |




# CPA Step 2: Measure the power consumption

- For every generated 8-bit input we measure the power consumption of the AES implementation over time

# CPA Step 2: Measure the power consumption

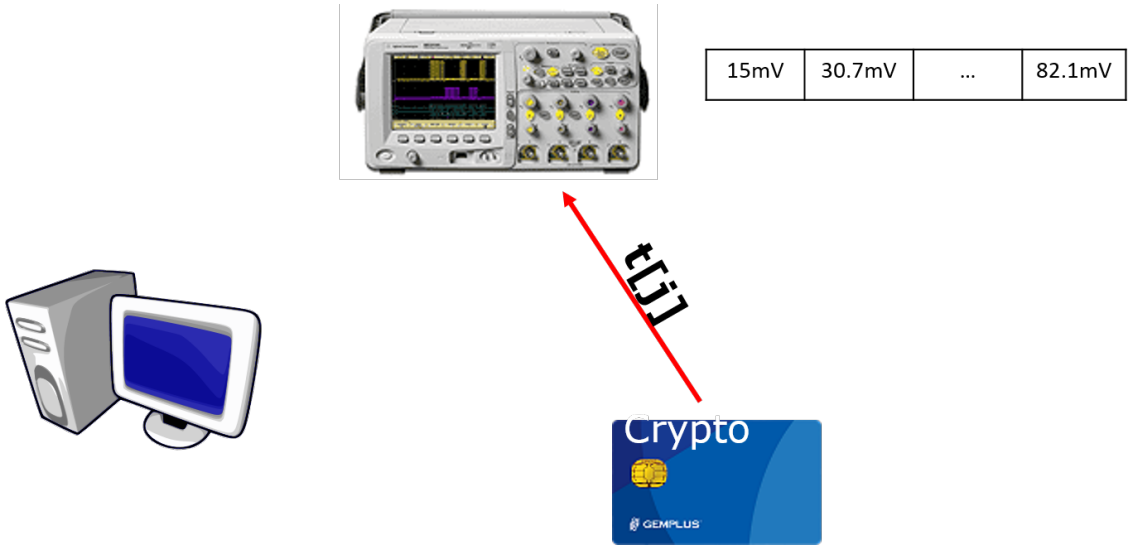
- For every generated 8-bit input we measure the power consumption of the AES implementation over time
- For every input $in_j, j = 1, \ldots, n$ we capture a digitized trace over time

# CPA Step 2: Measure the power consumption

- For every generated 8-bit input we measure the power consumption of the AES implementation over time
- For every input $in_j, j = 1, \ldots, n$ we capture a digitized trace over time
- We denote the trace related to input $in_j$ as $\mathbf{t_j} = [t_j^1\, t_j^2 \ldots t_j^m]^\intercal$. It contains $m$ points in time (a.k.a. samples)

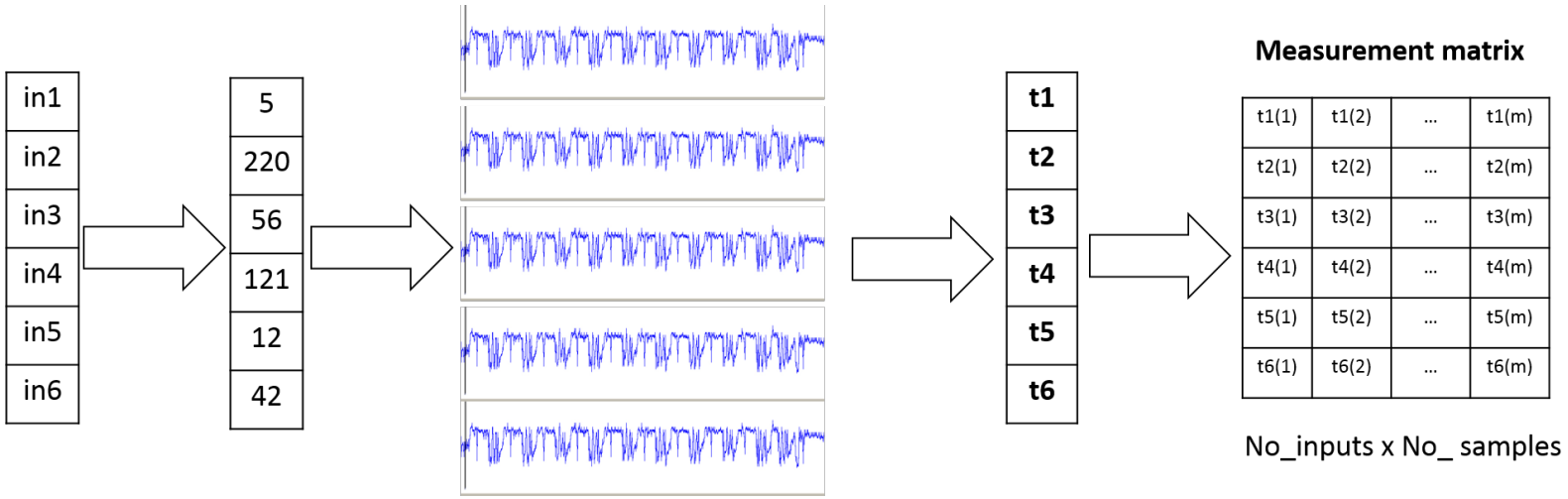
# CPA Step 2: Measure the power consumption

- For every generated 8-bit input we measure the power consumption of the AES implementation over time
- For every input $in_j, j = 1, \ldots, n$ we capture a digitized trace over time
- We denote the trace related to input $in_j$ as $\mathbf{t_j} = [t_j^1\, t_j^2 \ldots t_j^m]^\intercal$. It contains $m$ points in time (a.k.a. samples)



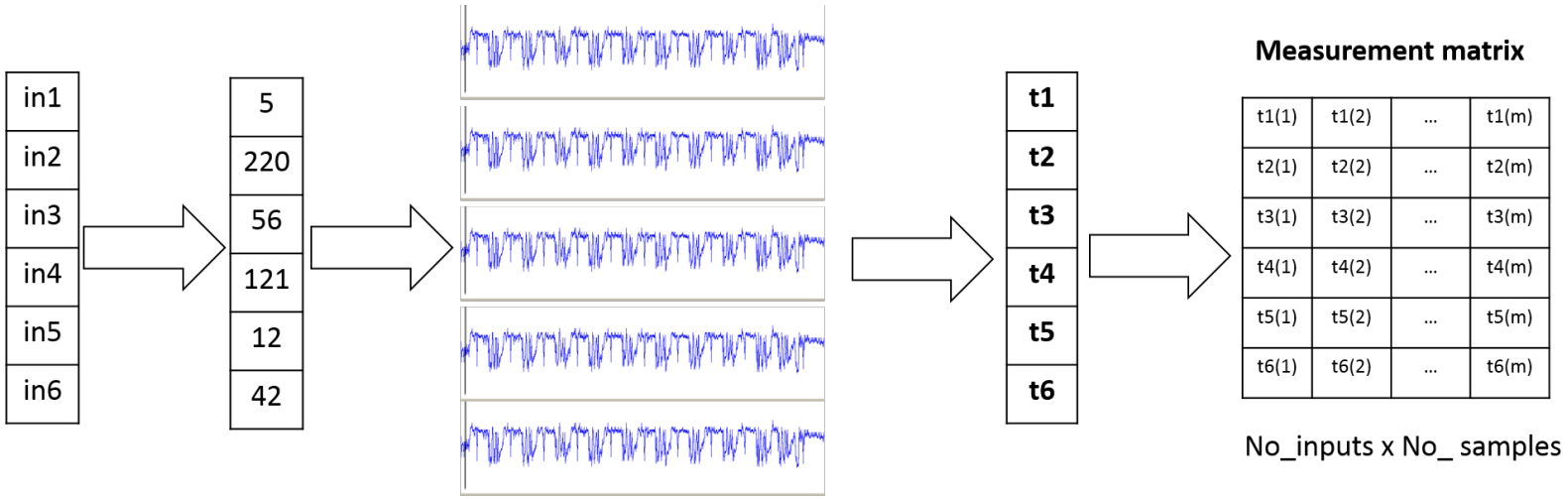
# CPA Step 2: Measure the power consumption

- Capturing 6 power traces with *m* time points (samples) results in the following measurement matrix



# CPA Step 2: Measure the power consumption

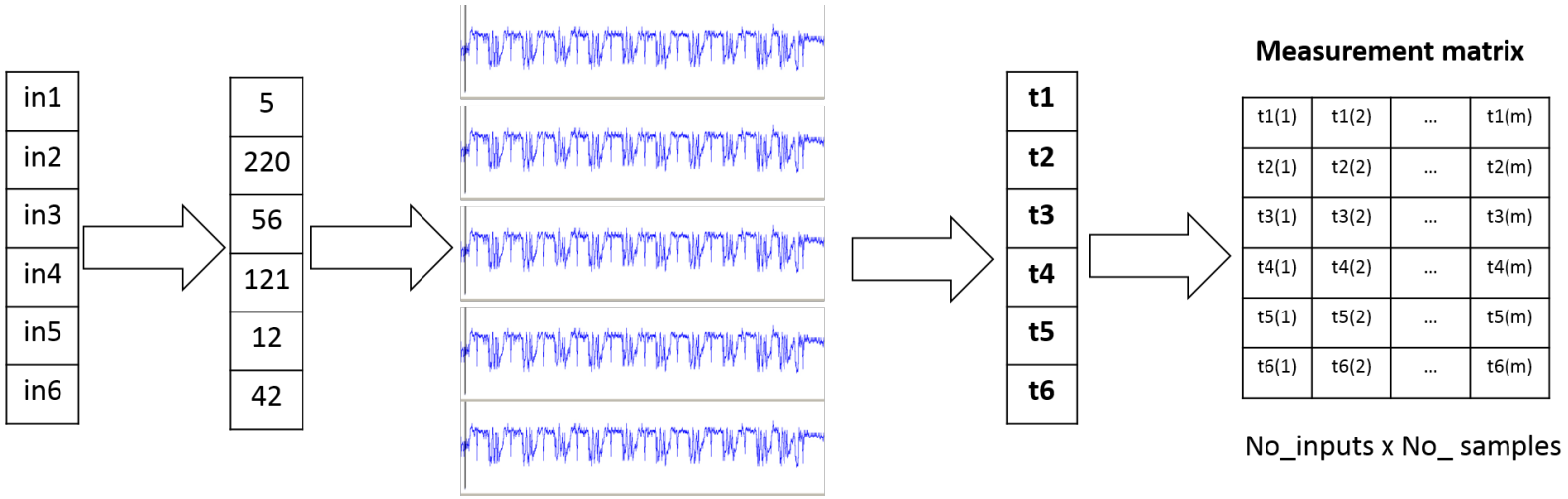
- Capturing 6 power traces with *m* time points (samples) results in the following measurement matrix




- Note that the power traces originate from the device, i.e. they are related to the secret key stored inside the device

# CPA Step 2: Measure the power consumption

- Capturing 6 power traces with *m* time points (samples) results in the following measurement matrix




- Note that the power traces originate from the device, i.e. they are related to the secret key stored inside the device
- We will refer to the unknown key that is stored in the device as $k_{dev}$

# CPA Step 3: Predict (hypothetical) intermediate values

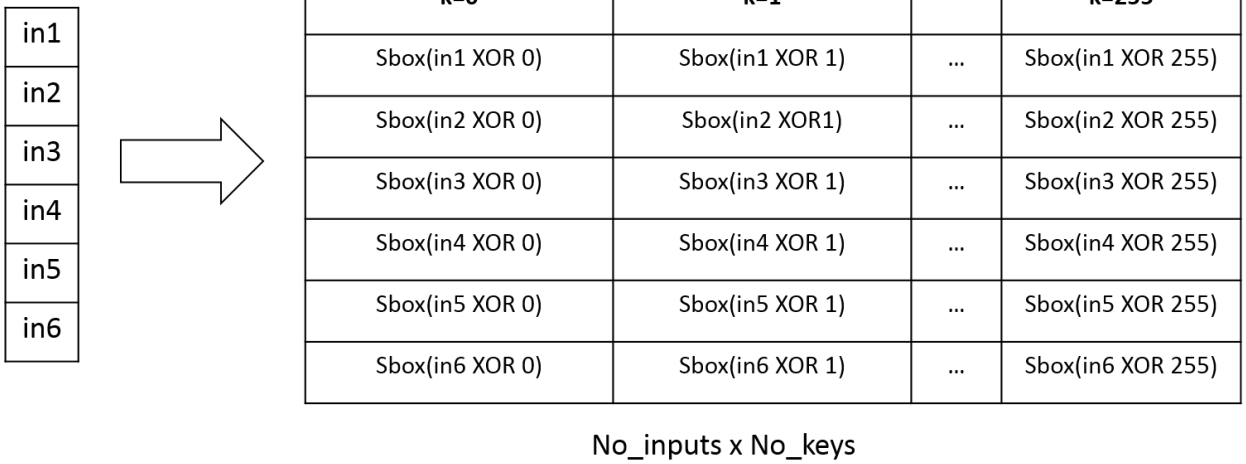
- In our device $y = Sbox(in \oplus k_{dev})$, but $k_{dev}$ is unknown!
- For a given input $in$ we can compute the value $y$ for all possible keys $k \in \{0, 1, \ldots, 255\}$

# CPA Step 3: Predict (hypothetical) intermediate values

- In our device $y = Sbox(in \oplus k_{dev})$, but $k_{dev}$ is unknown!
- For a given input $in$ we can compute the value $y$ for all possible keys $k \in \{0, 1, \ldots, 255\}$
- ForAll $in \in$ **in**
  ForAll $k \in \{0, 1, \ldots, 255\}$
  Compute $y(in, k) = Sbox(in \oplus k)$

# CPA Step 3: Predict (hypothetical) intermediate values

- In our device $y = Sbox(in \oplus k_{dev})$, but $k_{dev}$ is unknown!
- For a given input $in$ we can compute the value $y$ for all possible keys $k \in \{0, 1, \ldots, 255\}$
- ForAll $in \in$ **in**
  ForAll $k \in \{0, 1, \ldots, 255\}$
  Compute $y(in, k) = Sbox(in \oplus k)$




- One of the columns of this value-prediction matrix is the correct one!
- Divide and Conquer strategy

# CPA Step 4: Leakage model

- We map the hypothetical intermediate values to hypothetical power consumption values, producing the power-prediction matrix

**Power-prediction matrix**

| k=0 | k=1 | | k=255 |
|---|---|---|---|
| HW(Sbox(in1 XOR 0)) | HW(Sbox(in1 XOR 1)) | ... | HW(Sbox(in1 XOR 255)) |
| HW(Sbox(in2 XOR 0)) | HW(Sbox(in2 XOR1)) | ... | HW(Sbox(in2 XOR 255)) |
| HW(Sbox(in3 XOR 0)) | HW(Sbox(in3 XOR 1)) | ... | HW(Sbox(in3 XOR 255) |
| HW(Sbox(in4 XOR 0)) | HW(Sbox(in4 XOR 1)) | ... | HW(Sbox(in4 XOR 255)) |
| HW(Sbox(in5 XOR 0)) | HW(Sbox(in5 XOR 1)) | ... | HW(Sbox(in5 XOR 255)) |
| HW(Sbox(in6 XOR 0)) | HW(Sbox(in6 XOR 1)) | ... | HW(Sbox(in6 XOR 255)) |

No_inputs x No_keys

- A common choice is Hamming weight but keep in mind that other models may be applicable

# CPA Step 5: Comparison

- Compare the hypothetical power consumption values with the real measurements using Pearson correlation

# CPA Step 5: Comparison

- Compare the hypothetical power consumption values with the real measurements using Pearson correlation
- ForAll columns of measurement matrix
  ForAll columns of power prediction matrix
  Compute the correlation between columns

# CPA Step 5: Comparison

- Compare the hypothetical power consumption values with the real measurements using Pearson correlation
- ForAll columns of measurement matrix
  ForAll columns of power prediction matrix
  Compute the correlation between columns
- The highest correlation value reveals the key

# CPA Step 5: Comparison

- Compare the hypothetical power consumption values with the real measurements using Pearson correlation
- ForAll columns of measurement matrix
  ForAll columns of power prediction matrix
  Compute the correlation between columns
- The highest correlation value reveals the key

Pearson correlation coefficient $\rho_k(L, HW(V_g))$ for leakage $L$, Hamming weight of the intermediate value computed $HW(V)$ and key guess $K_g$ :

$$\rho_k(L, HW(V_g)) = \frac{cov[L, HW(V_g)]}{\sqrt{Var[L] \cdot Var[HW(V_g)]}} \tag{1}$$

# CPA Step 5: Comparison