# An Introduction to Physical Security

Christophe Clavier

University of Limoges

Master 2 Cryptis





## What is Physical Security?

Physical security is concerned by all means to jeopardize the security of a device by exploiting its physical properties or its behaviour when operating.

When applied to secure embedded devices such as smart cards, this may be performed by:

- Observing and analysing the duration of commands or operations
- Measuring the power consumption of the devices when it operates
- Perturbing the normal functioning, and analysing its abnormal behaviour or its faulty output
- Observing, probing or altering the surface of the chip (not covered in this presentation)
- ...

## Cryptology versus Physical Security

| | Cryptology | $\neq$ | Physical Security |
|---|---|---|---|
| | ↓ | | ↓ |
| Domain | Mathematics | | Physics |
| | ↓ | | ↓ |
| Objects | Virtual function (inputs, outputs) | | Implemented function (measurements, stress) |
| | ↓ | | ↓ |
| Tools | Algebra<br>Number theory<br>... | | Oscilloscope<br>Laser<br>... |
| | ↓ | | ↓ |
| Threats | Cryptanalysis | | Side Channel Analysis<br>Fault Analysis |





## Cryptology versus Physical Security

Physical attacks put a big trouble in cryptologists community ...

As for cryptanalysis they allow to recover secret keys of cryptographic functions, but ...

- They exploit the information leakage that is observed through physical measurements
- The cryptographic function is not broken. Only a weakness in its implementation is revealed and exploited.

Nevertheless, physical attacks are very important as they may threaten the security of an application much more easily than mathematical techniques would do.

## An History Perspective

Many physical attacks have been publicly revealed between 1996 and 1998:

- Timing Analysis (TA) has been published by P. Kocher in 1996
- Power Analysis (SPA and DPA) have been published by P. Kocher in 1998.
  Use of electromagnetic emanation instead of power consumption has first been practically performed simultaneously by Gemplus and IBM security teams in 2001
- Fall 1996, many announcements revealed ways to retrieve cryptographic keys by means of Fault Analysis (FA):
  - Bellcore's attack applies to RSA in CRT mode
  - Another technique appeared that apply to RSA in standard mode
  - E. Biham and A. Shamir invented Differential Fault Analysis (DFA) which exploits differential on DES output when a fault is injected in the penultimate round

All these concerns are still very active research domains in smartcard industry and academic community (patents, publications, improvements, counter-measures)

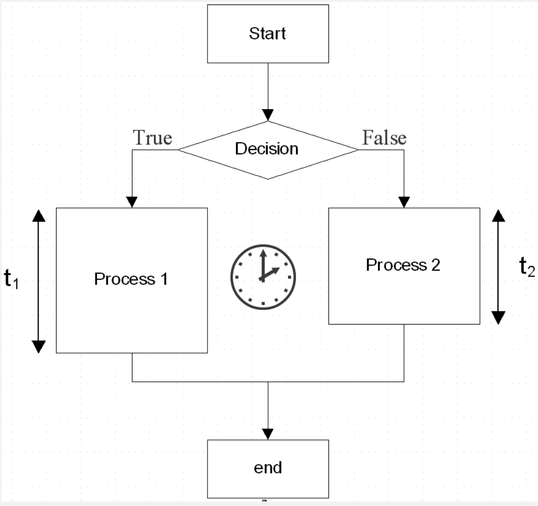




## General principle

- Some part of a computation takes longer or shorter depending on the value of some secret data
- The attacker is assumed to be able to measure execution timings (or it least its differential)
- The processing time
  - depends on the value of the secret data
  - leaks information about the secret

## A basic scenario




- The first part is unconditionally executed
- A test based on secret data is performed that leads to a Boolean decision
- Depending on the Boolean condition, the process may be long ($t_1$) or short ($t_2$)
- The end of the command is unconditionally executed

Université de Limoges





## PIN code verification

- A secret authentication data is securely stored in the smartcard
  - Example: A PIN code, 4 digits long
- Like passwords on a PC, authentication is based on the secret
- A dedicated function exists in the smartcard software:
  - The *VerifyPIN* command which:
    - Receives the proposed value for the PIN received from the terminal (`PIN_term`)
    - Compares it with the card PIN value (`PIN_card`)
    - Grants access rights if the comparison is successful

Université de Limoges

## PIN code verification

**Algorithm 1** PIN verification command (straightforward implementation)

**Input:** `PIN_card`: The 4-digit PIN value stored in the card
`PIN_term`: The PIN guess proposed by the user
**Output:** An answer `OK` or `KO`

```
procedure VERIFYSECRET(PIN_card, PIN_term)
    for i from 0 to 3 do
        if PIN_term[i] ≠ PIN_card[i] then
            return KO
        end if
    end for
    return OK
end procedure
```

Is this implementation functionally correct?

Is this implementation secure?





## How to recover the PIN?

- The attack assumes that any number of wrong presentations is allowed
  - Not really realistic when a ratification counter is implemented: the card is blocked after three erroneous attempts
- Attack implementation:
  - Propose all 10 possible values for `PIN_term[0]` (other digits do not matter)
  - Measure the corresponding command durations
  - Note that all command durations should be equal except one of them
  - The largest duration (one more loop) reveals the first PIN digit
  - Fix `PIN_term[0]` to the correct value and iterate successively for other digits `PIN_term[i]`
- Complexity:
  - Worst case: $4 \times 10$ commands (instead of $10^4$ for exhaustive search)

## Possible counter-measure

**Algorithm 2** PIN verification command (secure implementation)

**Input:** `PIN_card`: The 4-digit PIN value stored in the card
`PIN_term`: The PIN guess proposed by the user
**Output:** An answer `True` or `False`

```
procedure VerifySecretSecure(PIN_card, PIN_term)
    answer ← True
    for i from 0 to 3 do
        answer ← answer && (PIN_term[i] == PIN_card[i])
    end for
    return answer
end procedure
```

**Implementation rule**

Avoid any secret-related conditional branching

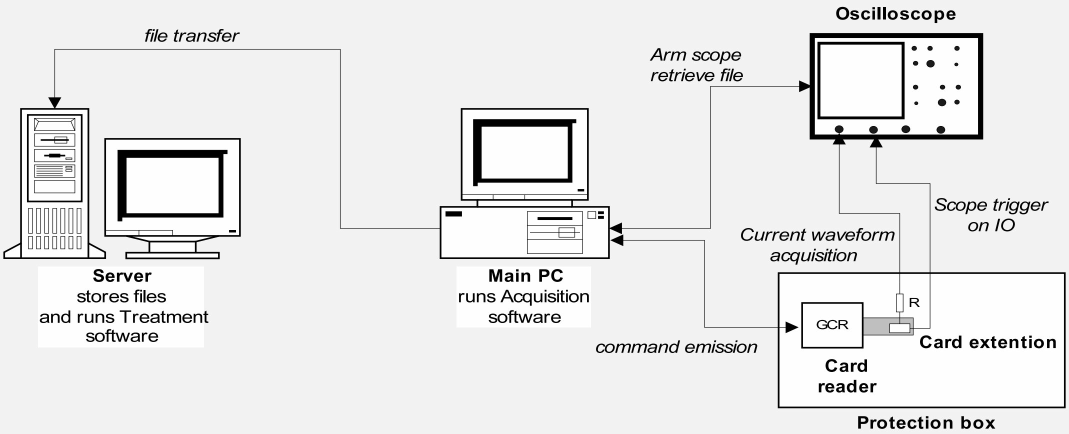




## Simple Power Analysis (content)

- Introduction to Simple Power Analysis
  - Experimental setup
  - Information leakage through the power
- Example
  - Electrical signatures
  - Interpretation
- Basic reverse engineering
  - Algorithm structure, implementation choices
- Key recovery
  - SPA against RSA private exponentiation
- Counter-measures

## Experimental setup



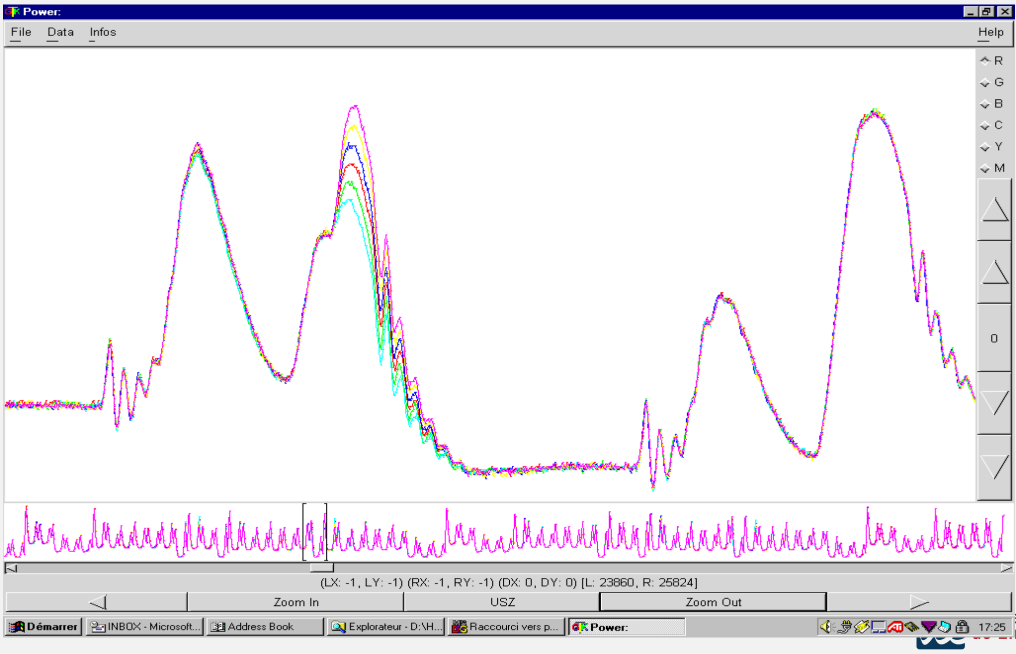





## Information leakage

- The power consumption of a chip depends on:
  - The executed instruction
  - The manipulated data
- Leakage models
  - Hamming weight of whatever data put on the bus: data, address, operation code, . . .
    - $W = a \cdot \mathrm{HW}(data) + b$
  - Hamming distance (bus transition weight) w.r.t. a reference state
    - $W = a \cdot \mathrm{HD}(data_t, RS) + b \;=\; a \cdot \mathrm{HW}(data_t \oplus RS) + b$
    - $RS$ : $data_{t-1}$ or $data_{t+1}$
  - Other models, chip & technologies, . . .

## Information leakage

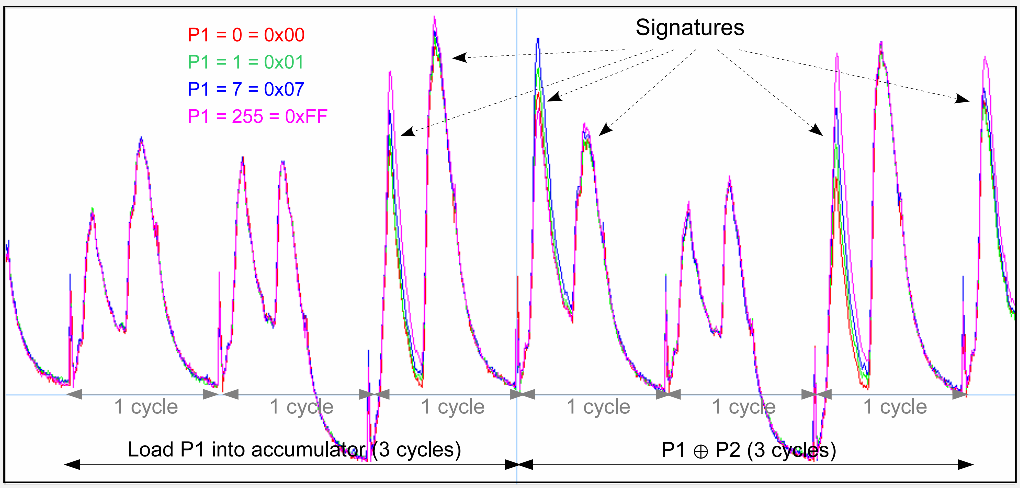




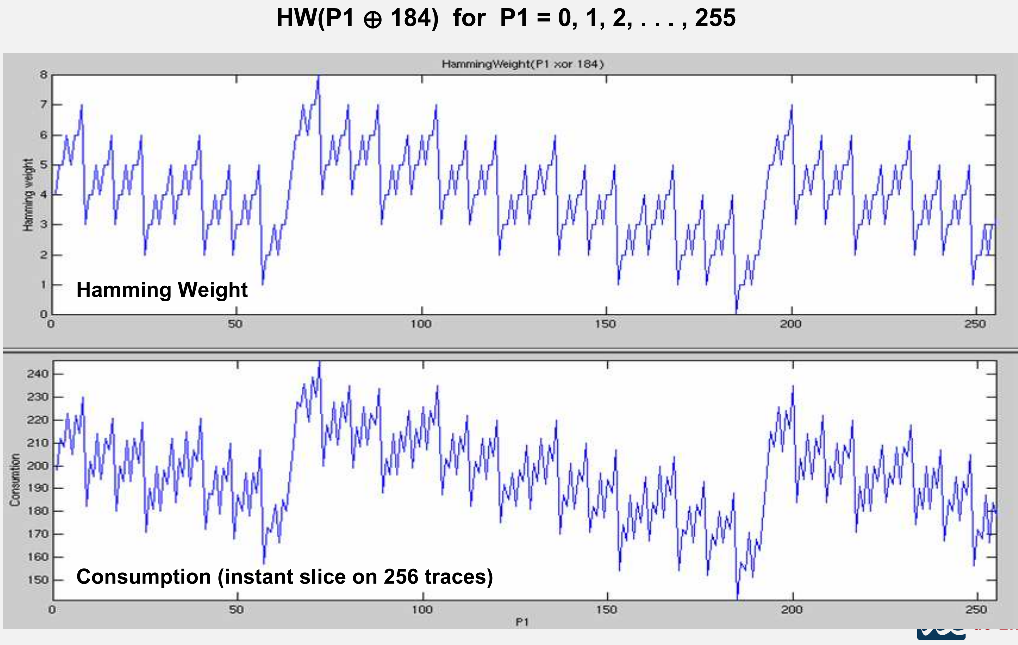



## Information leakage

**Load P1 and XOR with P2 = 0 (P1 ⊕ P2 = 0, 1, 7, 255)**



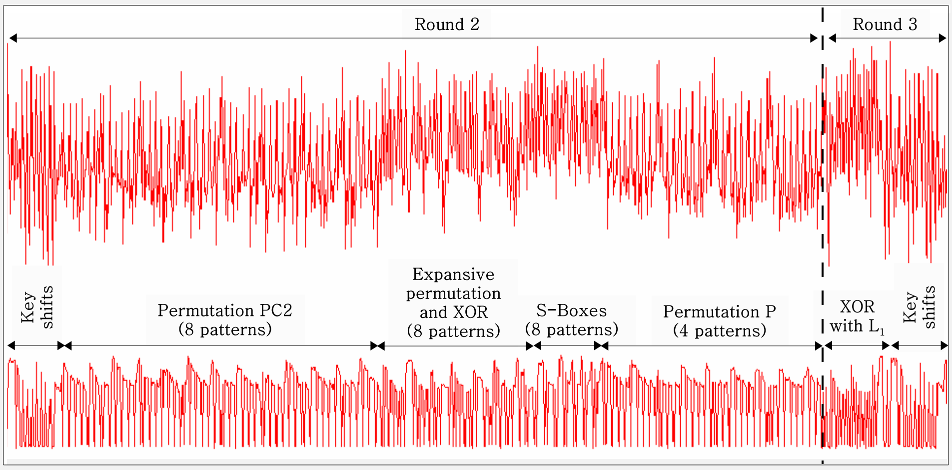
## Information leakage





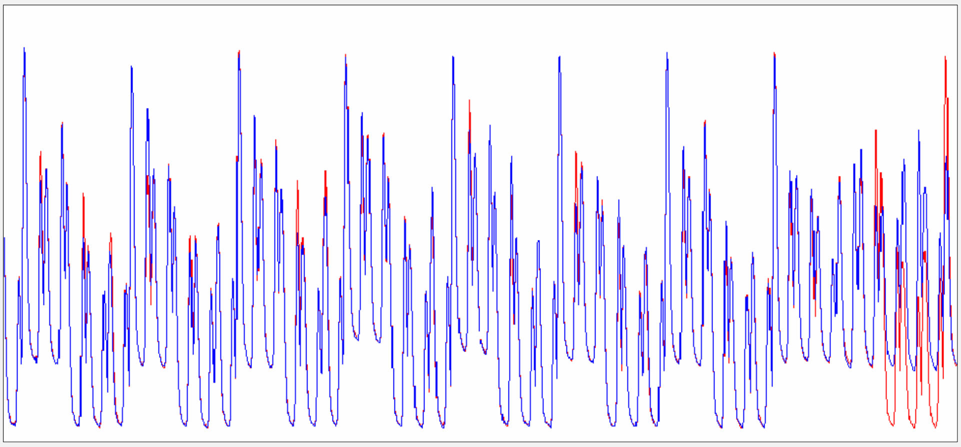



## Basic reverse engineering




- A simple power trace shows the different parts of a DES computation
- SPA may reveal the structure of algorithms and possibly implementation choices

## Basic reverse engineering



- Two power traces reveal whether an instruction is executed or not
- SPA may point the attacker to conditional branchings





## SPA attack on standard RSA

- RSA signature computation requires arithmetic operations on large integer operands
- On some crypto-coprocessors, the power consumption may depend on the type of arithmetic operation performed.
- SPA against the RSA signature private exponentiation

$$s = m^d \bmod n$$

  - $m$ is the message and $s$ is the signature
  - $n = pq$ is a large modulus (say 1024 bits), with $p$ and $q$ two large primes
  - $d$ is the private exponent such that $ed \equiv 1 \pmod{(p-1)*(q-1)}$
    (with $e$ the public exponent)

The attacker aims at retrieving $d$

## SPA attack on standard RSA

**Algorithm 3** RSA signature (classical left-to-right 'Square & Multiply')

**Input:** $d = (d_{k-1}, \ldots, d_0)$ the $k$-bit private exponent, $m$ the input
**Output:** $s$ the signature of $m$

```
procedure SIGN(m)
    s ← 1
    for i from k − 1 down to 0 do
        s ← s ∗ s mod n
        if d_i = 1 then
            s ← s ∗ m mod n
        end if
    end for
    return s
end procedure
```

Example: $s = m^{13} = m^{1101_b}$

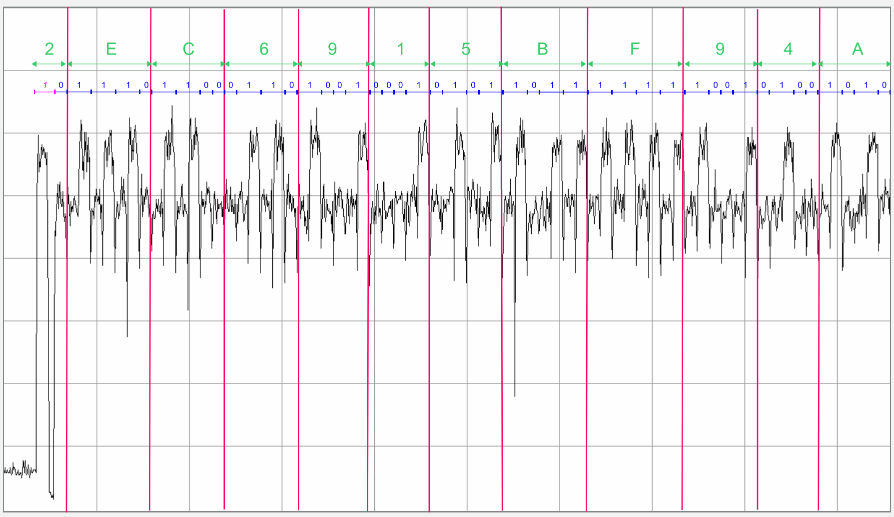
| | |
|---|---|
| $i = 3 \quad (d_3 = 1)$ | $s = (1)^2 * m = m^1$ |
| $i = 2 \quad (d_2 = 1)$ | $s = (m^1)^2 * m = m^3$ |
| $i = 1 \quad (d_1 = 0)$ | $s = (m^3)^2 = m^6$ |
| $i = 0 \quad (d_0 = 1)$ | $s = (m^6)^2 * m = m^{13}$ |





## SPA attack on standard RSA

Try to find the private key!




$d$ = 0x 2E C6 91 5B F9 4A

## Summary

- SPA 'visually' analyses one or few power (or EM) traces
  - Implementation related patterns
    - code structure: loops,...
    - sequence of operations: square and multiply
  - Data related variations: conditional branchings
- Often needs knowledge (or guess) of the algorithm
- May lead to partial reverse engineering
- A useful prior characterisation tool to make easier more elaborate attacks: DPA, CPA, fault attacks
- Counter-measures exist

Université de Limoges





## Software based counter-measures

- Prohibit any code branching conditioned by secret bits
- Random insertion of fake code (more dedicated to DPA or CPA)
  - Ensure variation of the relevant instants on power traces
- For RSA modular exponentiation (or ECC scalar multiplication)
  - Randomize the inputs : $m, d, n$ (more dedicated to DPA or CPA)

    Message blinding: $m^* = m + r_1 \cdot n$

    Modulus blinding: $n^* = r_2 \cdot n$

    Exponent blinding: $d^* = d + r_3 \cdot \Phi(n)$
  - Modify the structure (regular sequence of squarings and multiplications)
    - Square and Multiply always → S M S M S M S M...
    - Atomicity principle → M M M M M...
    - Square always → S S S S S...
    - Montgomery ladder, Joye ladder,...

Université de Limoges

## Hardware based counter-measures

### Chip's security features

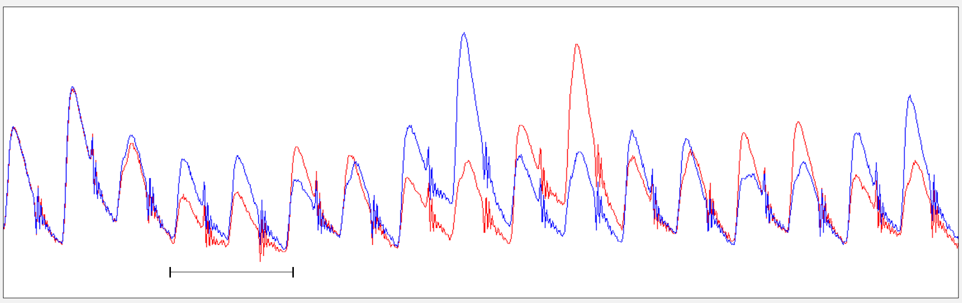
- Current scrambler ($y$ axis)
  - Introduce a noise on the power consumption
- Hardware desynchronizations ($x$ axis)
  - Waitstates (clock stealer)
  - Unstable internal clock





## Hardware desynchronization

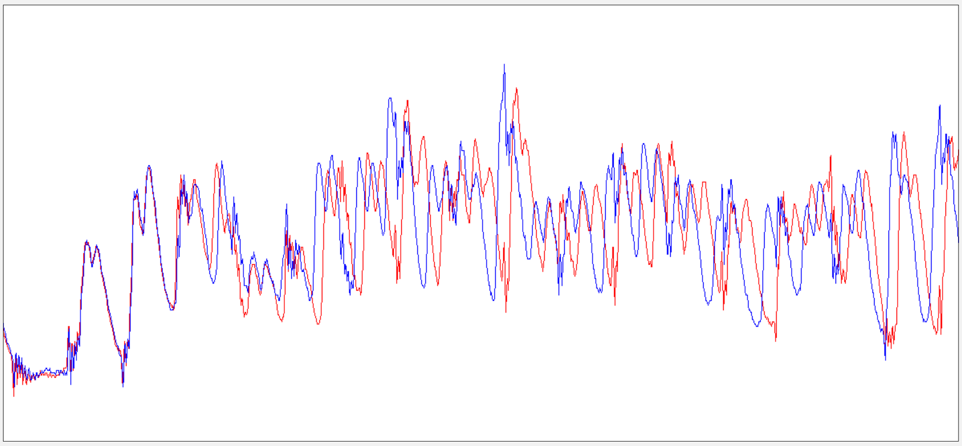
### Waitstates



- Some useless clock cycle are randomly inserted

## Hardware desynchronization

### Unstable clock



- Execution is internally clocked with an unstable oscillator
- Traces interpretation/superposition become more difficult





## Basics

### What is it about?

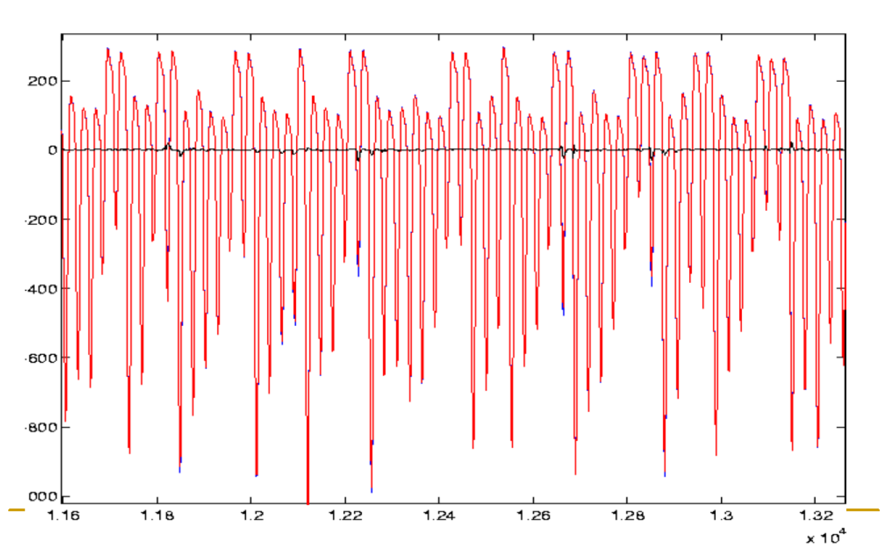
Differential Power Analysis is a mean to isolate and enhance the tiny contribution of an arbitrary bit (belonging to an arbitrary word) on a large set the power consumption traces.

### For what usage?

Differential Power Analysis comes in two flavours:

- DPA on *known* data: allows to identify locations on the power trace where a known data (*e.g.* fourth byte of input message) is processed → characterisation, reverse engineering
- DPA on *key dependant* data: results in an hypothesis test used to identify the value of a small part of the key → key recovery

## Small contribution of a bit on power consumption



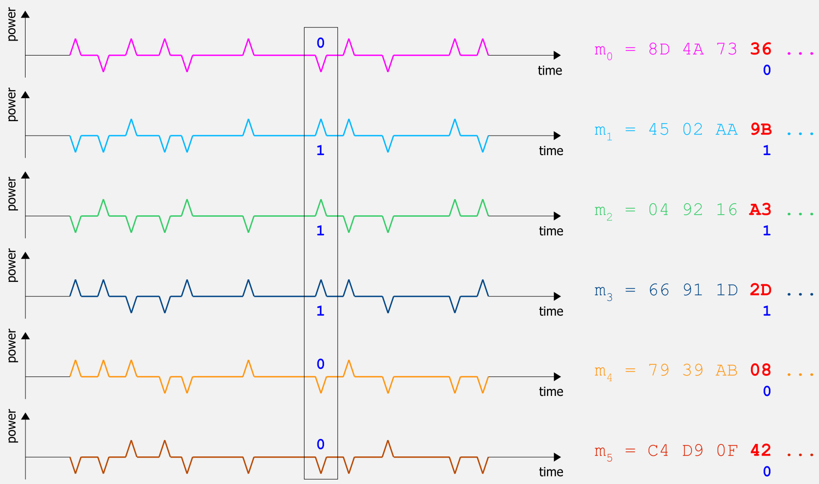

It seems difficult to identify when some particular bit or word is processed

Comparing traces with respect to the averaged signal should be easier





## Individual power consumptions w.r.t. average




Where is processed the fourth byte of the message?

One expect higher power when *e.g.* least significant bit is 1 rather than 0

Compare the series of targeted bit values with series of power consumptions

Consistency occurs when targeted bit is processed

## Enhancing the bit contribution

- Problem: Manually comparing the series of bit values with the series of traces at each instant is quite complex
- Solution: Split all traces according to the value $b$ of the targeted bit
  - Let $\mathcal{S}_1$ (resp. $\mathcal{S}_0$) be the set of traces for which $b = 1$ (resp. $b = 0$)
  - All traces in $\mathcal{S}_1$ (resp. in $\mathcal{S}_0$) have slightly higher (lower) power consumptions when the fourth message byte (which contains the targeted bit) is processed

    $\longrightarrow$ sets are significantly different
  - At other instants both sets contain high and low consumptions

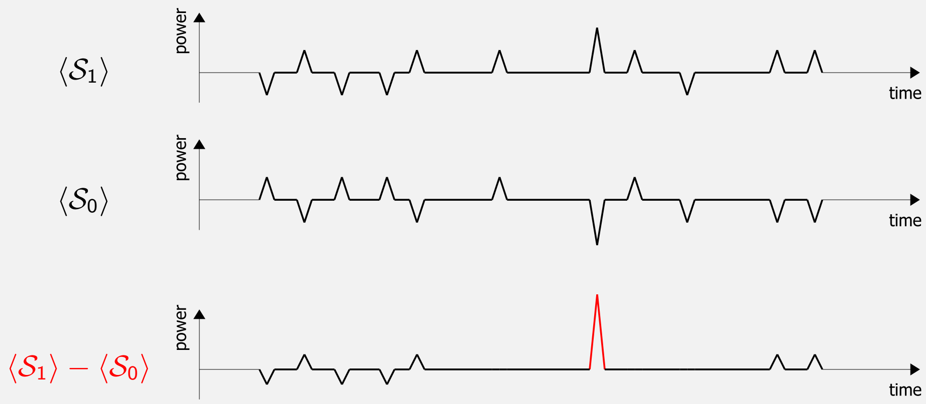
    $\longrightarrow$ sets are statistically similar
- Average each set and subtract them to each other:

$$\text{DPA trace} = \langle \mathcal{S}_1 \rangle - \langle \mathcal{S}_0 \rangle$$





## DPA trace as a difference of mean



## Summary

- The cryptographic function is executed $N$ times (N ranging from hundreds to dozens of thousands) with known and varying inputs $m_i$ $\rightarrow$ power traces $\mathcal{T}_i$
- The attacker arbitrary selects an intermediate bit whose value only depends on known inputs (message or cipher)
- When this selected bit $b$ is processed, the power consumption is assumed to slightly depend on the value of $b$ (*e.g.* higher if $b = 1$ than if $b = 0$)
- Splitting the set of traces according to $b$ results in two subsets whose mean power consumptions statistically differ if and when the selected bit is manipulated.

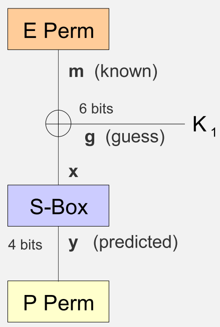
**Lesson:** The difference of mean operator results in a DPA peak located at the instant when a targeted bit is processed, because the series of targeted bit values is consistent with that actually processed by the device





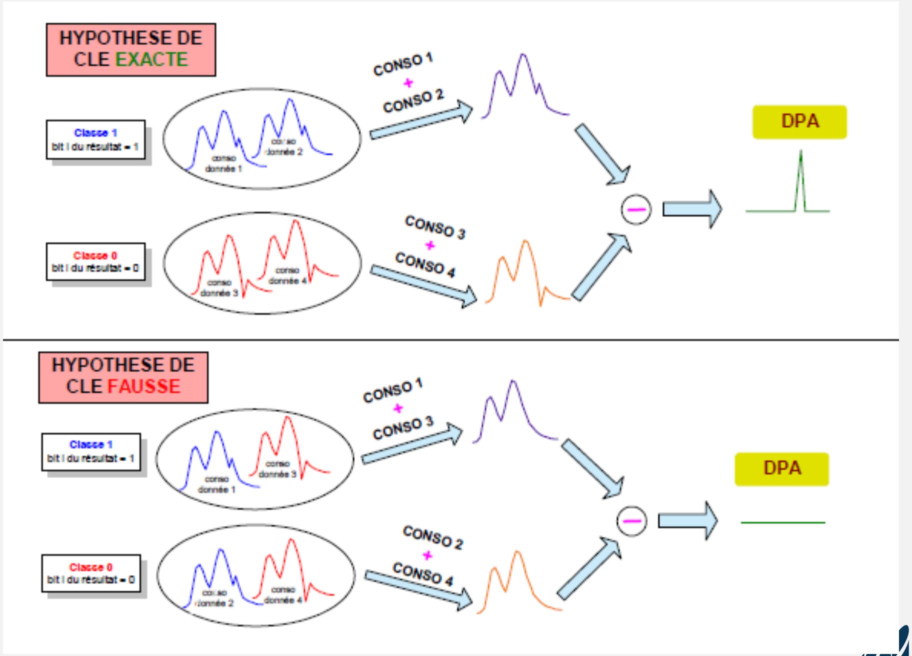
## Key recovery DPA

- DPA is applicable on any varying and known data:
  - Plaintext, cipher
  - Initial and final permutations (DES)
  - Output of expansive permutation in the first round (DES)
- But what if the targeted data is not known?
  - *e.g.* an S-Box output in the first round




- Focus on one S-Box: $y = S(x) = S(m \oplus g)$
  - Target one bit $b$ of the S-Box output $y$
  - Make a guess $g$ on the subkey $\rightarrow$ predict $y$
  - $\rightarrow$ a series of predicted values for $b$
  - Compute the DPA trace for this series of predicted bit values
  - Only for the correct guess the series perfectly reflects what was actually processed in the device
  - Exhaust all $2^6$ guess: the subkey value is identified by the presence of a DPA peak

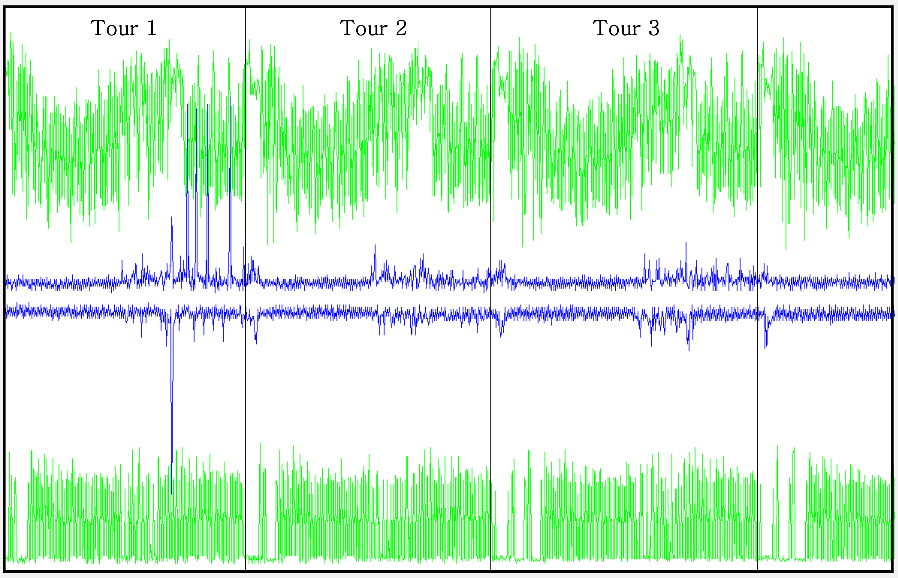
## All in one slide



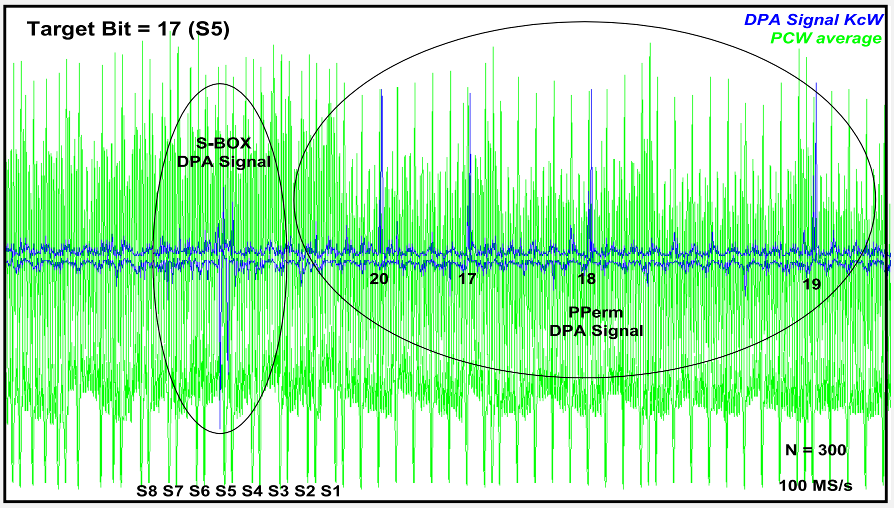





## Example of DPA curve on DES



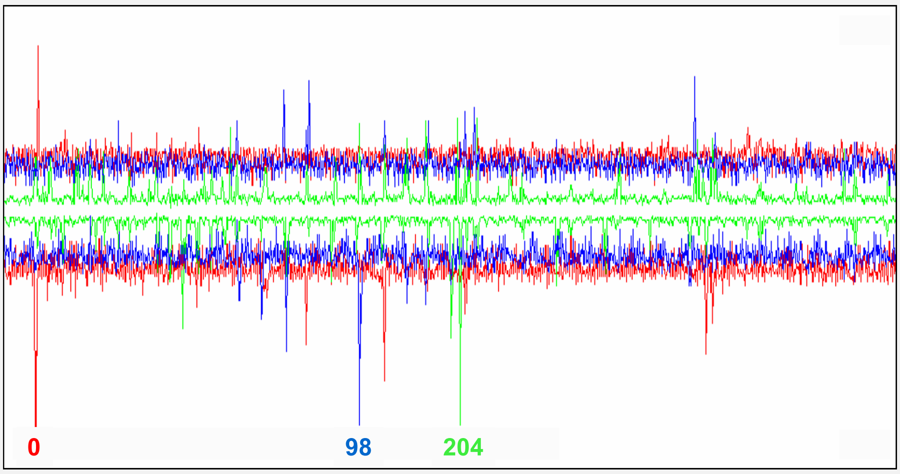
## Example of DPA curve on DES (zoom)








## Example of DPA curve on AES



## Implicit assumptions...

### Implicit assumptions

**Word** The contribution of the non-targeted bits is independent of the targeted bit value

- Their means in each trace set is the same
- The attacker does not need to care about these bits

**Key** The predicted value of the targeted bit for any wrong guess is independent of its value for the correct guess

**Time** At each time the targeted bit is not explicitly handled, the consumption is independent of its value

Under these assumptions:

- For any wrong guess about the subkey, the DPA trace is flat (statistically equal to zero) all along the trace
- For the correct guess, the DPA trace shows a positive DPA peak (statistical difference between the two wave set averages) at that time the targeted bit is handled





## ... and what reality is!

### Facts

- For the correct guess, DPA peaks appear also when the targeted bit is not explicitly handled
- More problematic, DPA peaks also appear for wrong guesses (ghost peaks)
- The true DPA peak may be smaller (or even null or negative!) than some ghost peaks

### Main reasons

- The bit-oriented consumption model (higher consumption if bit equal 1 than 0) is too simplifying
- The values of the non-targeted bits in the same word also contribute to the power consumption → they should not be ignored

## Correlation Power Analysis

- It has been observed that good models of the power consumption (as a function of the data) are

  Bit values $W = a \cdot \mathrm{HW}(data) + b$

  Bit transitions $W = a \cdot \mathrm{HD}(data, RS) + b = a \cdot \mathrm{HW}(data \oplus RS) + b$

  where RS is the (usually) constant reference bus state

- We can take advantage of the linear relationship between the measured consumptions and the Hamming weights (or Hamming distances) of the actual manipulated data

- The *Pearson correlation coefficient* is the most suitable tool to evaluate the linear fit between the measured consumptions ($W$) and the Hamming weights ($H$) of the predicted (under key guess) data

$$\rho_{H,W}(t) = \frac{\mathrm{Cov}(H, W(t))}{\sqrt{\mathrm{Var}(H)\,\mathrm{Var}(W(t))}}$$

- The CPA trace for the correct guess shows the highest peak





## Correlation Power Analysis

For each guess $g$ about the subkey, one computes a CPA trace:

- For each $i = 1, \ldots, N$, given the known plaintext $M_i$ (or ciphertext $C_i$) one can compute the predicted intermediate value $v_i = f(M_i, g)$ and then the predicted Hamming weight $h_i = \mathrm{HW}(v_i)$

- The CPA trace is generated by correlating (for each $t = 1, \ldots, T$) the series of Hamming weights $h_i$ with the series of measured consumptions $w_i(t)$:

$$\begin{aligned}
\rho_{H,W}(t) &= \frac{\mathrm{Cov}(H, W(t))}{\sqrt{\mathrm{Var}(H)\,\mathrm{Var}(W(t))}} \\
\mathrm{Cov}(H, W(t)) &= \frac{1}{N}\sum_{i=1}^{N}(h_i - \overline{h})(w_i(t) - \overline{w(t)}) \\
\mathrm{Var}(H) &= \frac{1}{N}\sum_{i=1}^{N}(h_i - \overline{h})^2 \\
\mathrm{Var}(W(t)) &= \frac{1}{N}\sum_{i=1}^{N}(w_i(t) - \overline{w(t)})^2
\end{aligned}$$

## Comparison between DPA and CPA

### What is common between DPA and CPA

- Statistical techniques allowing a hypothesis test on a guess about part of the secret key
- Need many side channel traces with known and variable inputs
- Do not require knowledge about when relevant instruction is executed





## Comparison between DPA and CPA

### DPA specifics

- \+ Very weak assumption on the consumption function
  (different consumptions for 0 and 1)
- \+ When predicting a bit, other ones in the same word do not matter
  (hopefully)
- – Subject to 'ghost peaks' problem
- – Needs many samples (hundreds to thousands)

### CPA specifics

- – Needs a consumption model
- – Whole word must be predicted (this is often possible)
- \+ More discriminating as all available information is involved:
  - Quantitatively (all bits) and qualitatively (model)
  - \+ Quasi insensible to 'ghost peaks' problem
  - \+ Needs much less samples (few dozens to hundreds)

## Template Analysis

### Principle

Given the statistical distributions (template models) of the side-channel leakage for each value of a sensitive data (e.g. a key byte), compare the measured leakage of an attacked device with all templates in turn to find the one giving the best fit (in the sense of maximum a posteriori likelihood)

- The attack only focuses on instants when the relevant data is manipulated $\rightarrow$ selection of points of interest
- The noise is usually considered as normally distributed $\rightarrow$ templates are defined by the mean and variance (or mean vector and covariance matrix) of the Gaussian distribution of the leakage

Université de Limoges





## Template Analysis : two phases

### Building the templates base (off-line, model inference phase)

- This is a pre-computation phase which needs an open device: attacker must be able to change the key
- The leakage on the open device is assumed to perfectly mimic that of the attacked device
- Leakage is considered only on few interest points
- Each template is made by averaging a huge number of traces with same sensible intermediate value

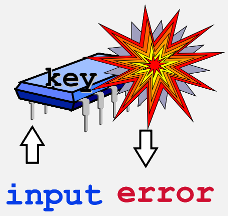
### Comparing a trace with the templates (on-line, attack phase)

- A trace is acquired on the attacked device and compared with each template
- The maximum likelihood of the template given the observed leakage (best fit) gives the value of the secret data

Université de Limoges

## Fault Analysis (content)

- Fault injection methods
  - Glitch attacks
  - Temperature variation
  - Magnetic pulses
  - Illumination attacks
- Classification
  - Permanent faults
  - Transient faults
- Fault models
  - Fault Analysis examples
    - Differential Fault Analysis (DFA) on DES
    - Collision Fault Analysis (CFA) on AES
  - Counter-measures








## Fault injection methods

### Glitch attacks

- Variations in supply voltage during execution may cause the processor to misinterpret or skip instructions
- Variations in the external clock may cause data misread or an instruction skips

### Temperature attacks

- Variations in temperature may cause:
  - random modification of RAM cells
  - erroneous read operations in NVMs

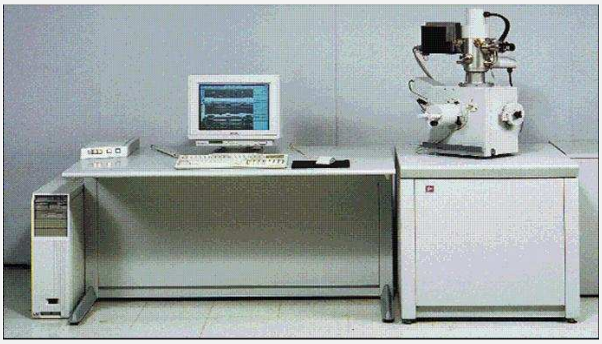
### Magnetic attacks

- Emission of a powerful magnetic pulse near the silicon (duration, power and location of the emission)

## Fault injection methods

### Illumination attacks

- Photoelectric effect (duration, power and location of the emission)
- White light (e.g. a flash camera)
  - cheap equipment
- Laser
  - allows to precisely target a circuit area







## Type of faults

- Permanent faults
  - Destructive effect
  - The value of a cell is definitely changed
    - data (EEPROM, RAM)
    - code (EEPROM)
- Transient faults
  - The circuit recovers its original behaviour after reset or when the fault's stimulus ceases
  - The code execution or a computation is perturbed:
    - instruction byte: a different instruction is executed (call to a routine skipped, test avoided, ...)
    - parameter byte: a different value or address is considered (operation with another operand, loop variable modified, ...)

## Transient fault models

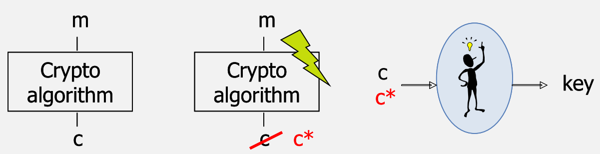
1. Precise bit errors
    - The attacker can cause a fault in a single bit
    - Full control over the timing and location of the fault
2. Precise byte errors
    - The attacker can cause a fault in a single byte
    - Full control over the timing but only partial control over the location of the fault (*e.g.* which byte is affected)
        - new faulty value can not be predicted
3. Unknown byte errors
    - The attacker can cause a fault in a single byte
    - Partial control over the timing and location of the fault
        - new faulty value can not be predicted
4. Random errors
    - Partial control over the timing and no control over the location





## Differential Fault Analysis

- Principle of Differential Fault Analysis (DFA)
    - Ask for a cryptographic computation twice
        - With any input and no fault (reference)
        - With same input, inject a fault during the cryptographic computation
    - Infer information about the key from the output differential




- When applied to DES (Biham & Shamir, 1996)
    - A fault is injected in the penultimate ($15^{th}$) round
    - The differential propagates and is observed after the last round
    - For each S-Box at last ($16^{th}$) round, eliminate subkeys incompatible with input/output differentials
- Also applies to other algorithms (RSA, AES, ...)

## Differential Fault Analysis on the DES

**Adversary Model**

We assume that the attacker is able to produce a computational error into the $15^{\text{th}}$ round of the DES

- The attack may be adapted to faults occurring in the $14^{\text{th}}, 13^{\text{th}}, \ldots$ round

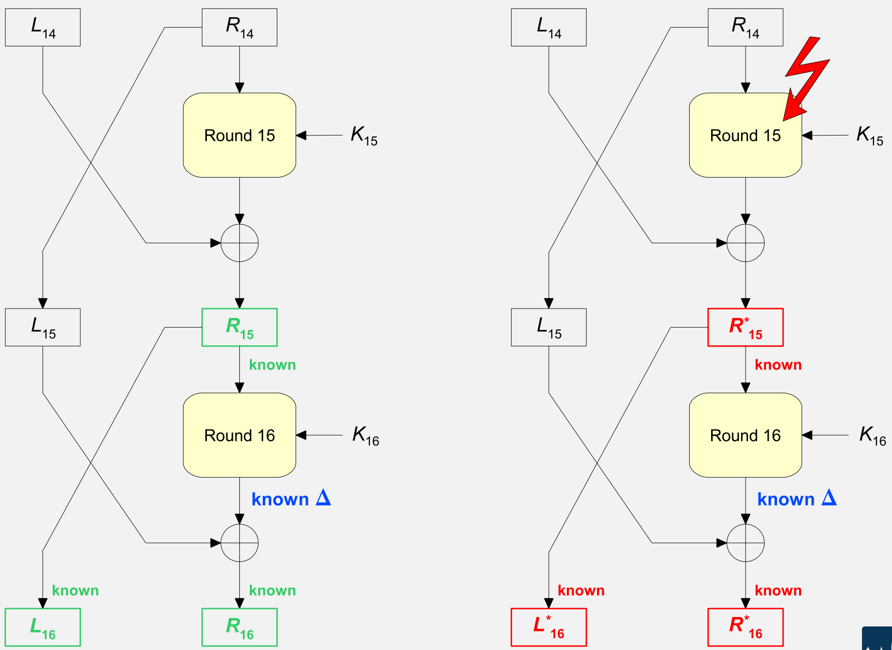
Given a pair of normal/faulty ciphertexts $(C, C^*)$ for the same entry $M$, one can trivially derive:

- the last round output $(L_{16}, R_{16})$ from the normal ciphertext $C$
  (simply inverse the final permutation)
- the last round output $(L^*_{16}, R^*_{16})$ from the faulty ciphertext $C^*$

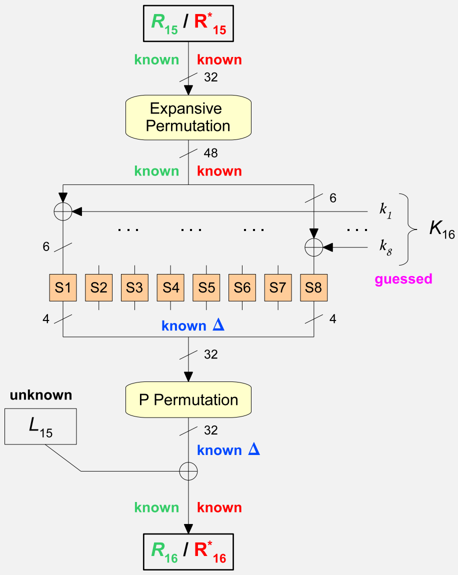




## Differential Fault Analysis on the DES



## Differential Fault Analysis on the DES



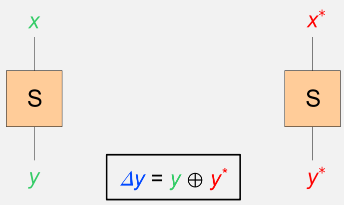

- $R_{15}$ and $R^*_{15}$ are known $\longrightarrow$
  - normal and faulty outputs of expansive permutation are known
- $R_{16}$ and $R^*_{16}$ are known, $L_{15}$ is unknown but not affected by the fault $\longrightarrow$
  - P permutation output differential is known
  - S-Box output differential is known

Each guess on a subkey leads to constraints on inputs/outputs of the S-Box





## How to retrieve $K_{16}$?




For each S-Box in the last round:

- Guess the 6-bit subkey $k$
- Compute the S-Box inputs $x$ and $x^*$, then S-Box outputs $y$ and $y^*$
- Check if known $\Delta y$ is equal to $y \oplus y^*$
- Invalidate the key guess if not

The number of remaining keys is expected to considerably reduce

Change the input $M$ repeat the process

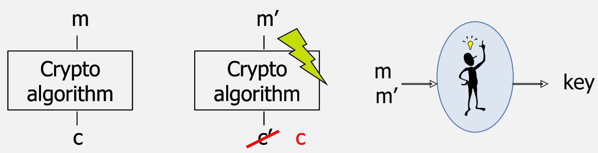
- Intersect the subkey spaces
- Only few messages allow the identify the correct subkey

Recover the 8 remaining unknown bits by exhaustive search

## Collision Fault Analysis

DFA aims at retrieving information about the key from a differential effect on the output.

With Collision Fault Analysis (CFA), information is obtained from two identical outputs.

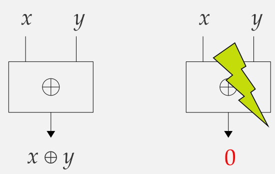




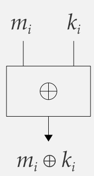



## Collision Fault Analysis on the AES

Assume the following (realistic) fault model:




First AES `AddRoundKey` implements 16 times:




Inject a fault when executing $z_i = m_i \oplus k_i$ and store the corresponding corrupt output $C'$. $(z_i' = 0)$

Exhaustively search for $m_i^*$ (without fault) until the same output is obtained. Then, $k_i = m_i^*$.

Whole key is retrieved within 16 faults and at most 4 096 normal executions.

## Counter-measures

- Hardware counter-measures
  - Sensors detecting an abnormal environmental condition
    - Light
    - Supply voltage
    - Temperature
    - Frequency
- Software counter-measures
  - Redundancy, duplication
    - Space or time
    - Simple or multiple
  - Blinding (data randomization)
  - Shuffling (code randomization)
  - Desynchronization (time randomization)





## Several Notions in This Lesson

- Physical security threats (timing analysis, power analysis, fault analysis)
- Timing analysis example (PIN code verification)
- Information leakage, power consumption model, hypothesis tests based attacks (DPA, CPA), reverse engineering
- Fault injection means, classification, models, DFA
- Many examples and counter-measures

THANK YOU FOR YOUR ATTENTION!

QUESTIONS?