An Introduction to Physical Attacks
– Application to Secret Specifications Algorithms –

Christophe Clavier

GEMALTO

Security Labs

SSTIC – Rennes – May 30, 2007

Outline

Introduction to Physical Attacks

What is Physical Security ?

Physical security $\neq$ Cryptanalysis

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

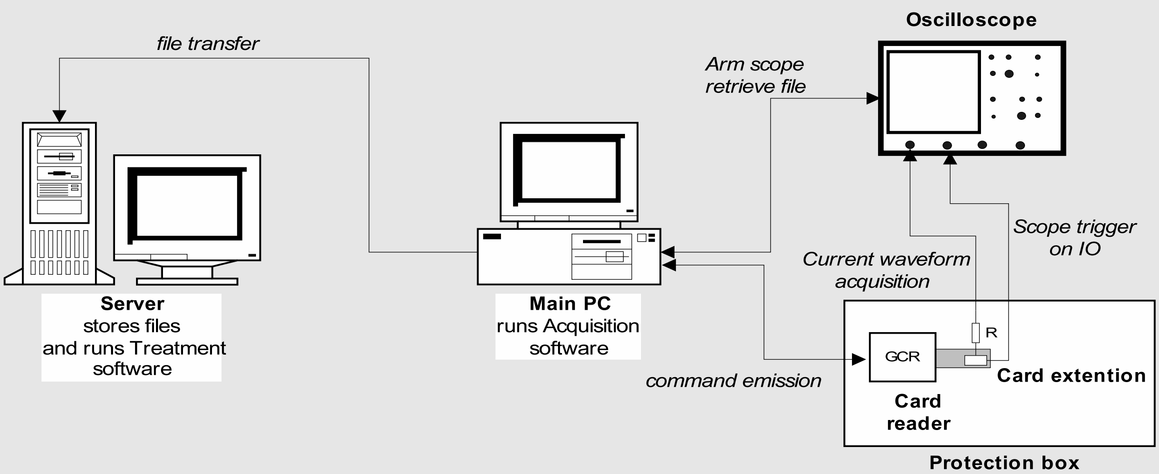
When applied to secure embedded devices such as smart cards, this may be performed by:

- Observing and analysing the duration of commands or operations
  (not covered in this presentation)
- Measuring the power consumption of the device 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)

Side Channel Analysis (content)

- Introduction to Power Analysis
  - Experimental equipment
  - Information leakage through the power
- Simple Power Analysis (SPA)
  - Against an RSA private exponentiation

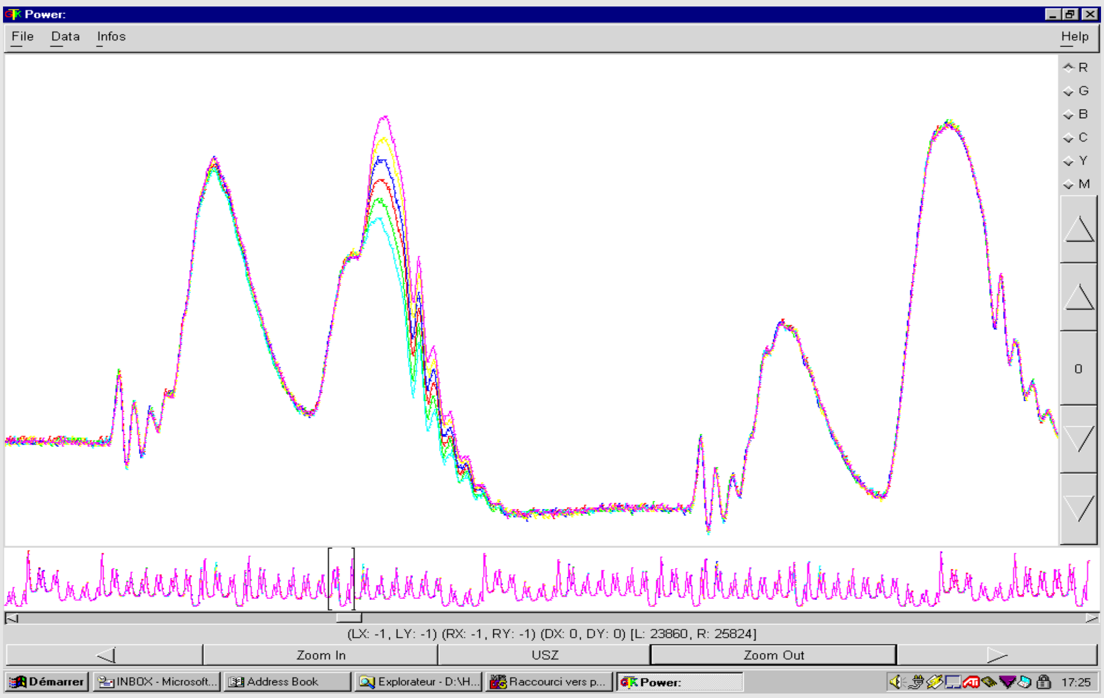
Experimental equipment



Information leakage

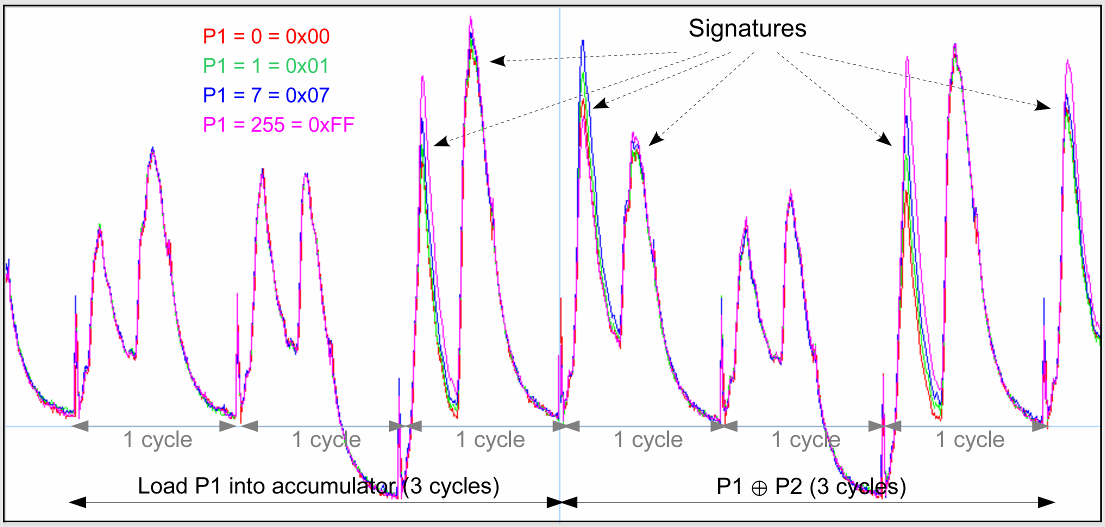
- The power consumption of a chip depends on:
  - The executed instruction
  - The manipulated data
- Leakage models
  - Hamming weight of whatever data put on the bus: data, address, opearation code, . . .
    - $W = a \cdot \mathrm{HW}(data) + b$
  - Hamming distance (bus transition weight) w.r.t.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

Information leakage



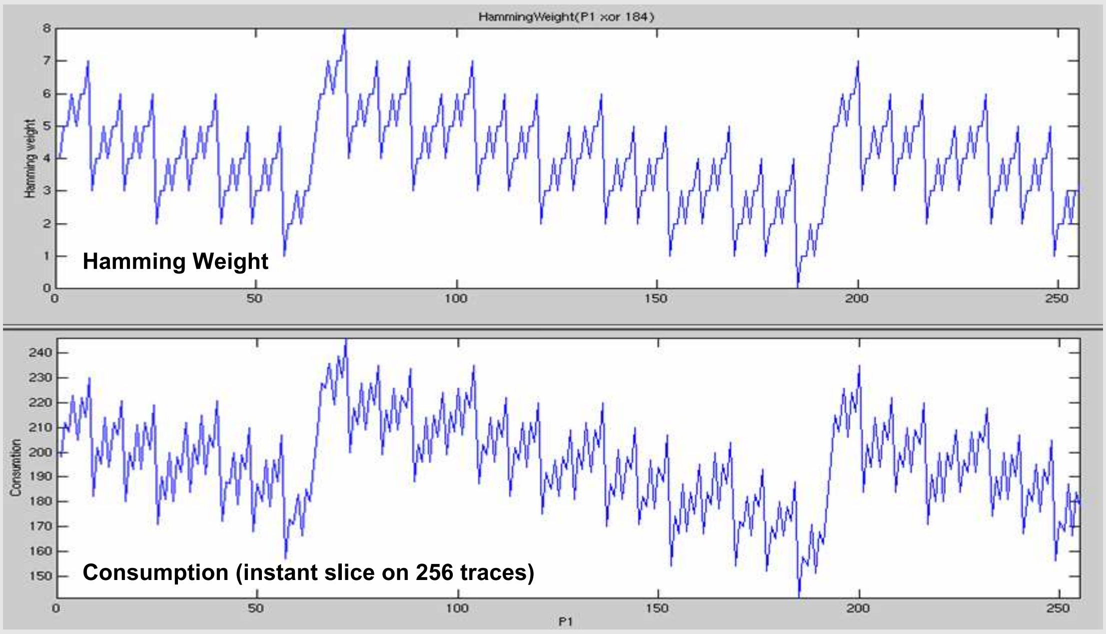
Information leakage

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



Information leakage

HW(P1 ⊕ 184) for P1 = 0, 1, 2, . . . , 255



SPA attack on standard RSA

- RSA signature computation requires arithmetic operations on large integer operands
- On some cryptoprocessors, the power consumption may depend on the type of (large integer) 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 1 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

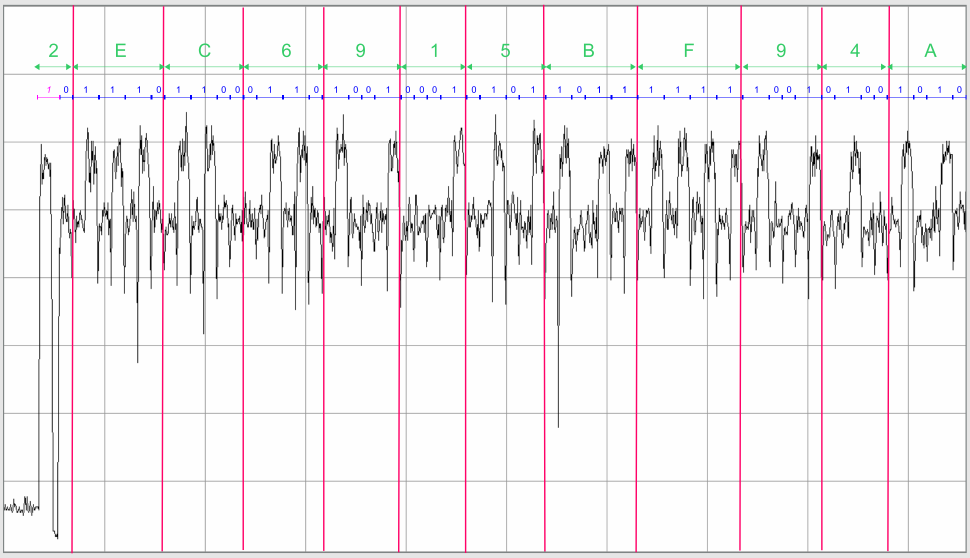
procedure SIGN(m)
  $s \leftarrow 1$
  for i from $k-1$ down to 0 **do**
    $s \leftarrow s * s \bmod n$
    if $d_i = 1$ **then**
      $s \leftarrow s * m \bmod n$
    end if
  end for
  return s
end procedure

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

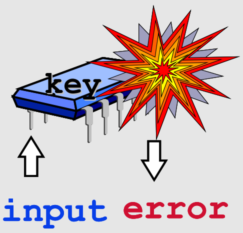
The power consumption directly reveals the private key!



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

Fault Analysis (content)

- Fault injection methods
  - Glitch attacks
  - Temperature variation
  - Light attacks



- Classification
  - Permanent faults
  - Transient faults
- Fault Analysis examples
  - Differential Fault Analysis (DFA) on DES
  - Collision Fault Analysis (CFA) on AES

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 miss

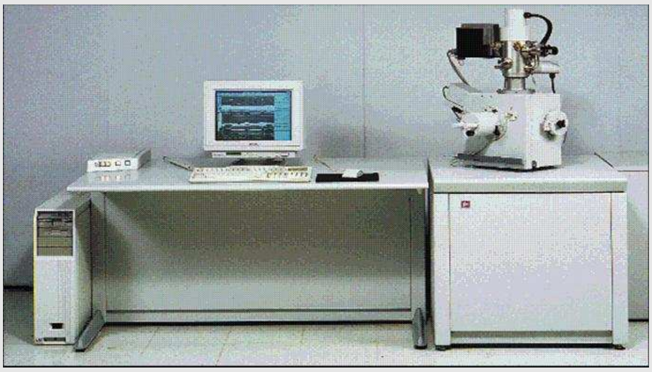
Temperature attacks

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

Fault injection methods

Light attacks

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

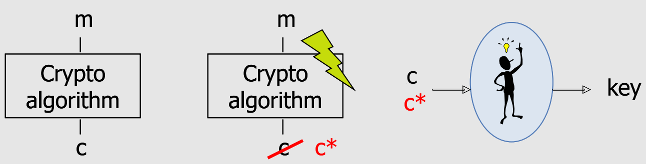


Type of faults

- 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, . . .)

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

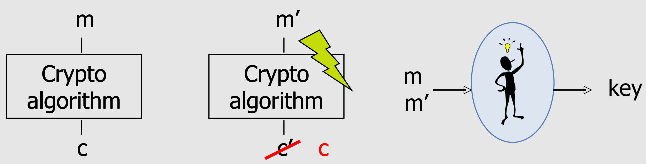


- When applied to DES (Biham & Shamir, 1996)
  - A fault is injected in the penultimate (15^{th}) round
  - The differential propagates and is observed after the last round
  - For each S-Box at last (16^{th}) round, eliminate subkeys incompatible with input/output differentials
- Also applies to other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

Collision Fault Analysis

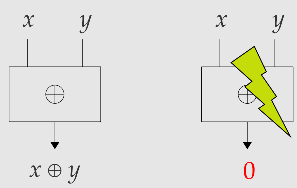
DFA aims at retrieving information about the key from a differential effect on the output.

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

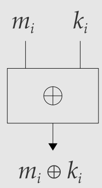


CFA on AES

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 stores the corresponding corrupt output $\overset{\rightsquigarrow}{C}$. ($\overset{\rightsquigarrow}{z_i} = 0$)

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 4096 normal executions.

Discussion

- All previous attacks implicitly assume that the cryptographic function (DES, AES, RSA, ...) is known from the attacker.
- As a security measure, keeping the cryptographic algorithm secret should make such physical attacks very difficult (impossible?).

Two questions

Reverse engineering Is it possible to reveal (part of) the specification of the algorithm by physical attacks?

Key recovery Without knowledge about the algorithm, is it yet possible to blindly recover the key?

Reverse Engineering of Unknown Algorithms

What is SCARE ?

Side Channel Analysis for Reverse Engineering

The side channel signal is exploited in order to reveal functional parts of unknown algorithms.

Appeared in 2003 [Nov03, Cla04] with an application to a secret A3/A8 algorithm.

In 2005, Daudigny *et al.* [DLMV05] also applied SCARE to recover *a priori* unknown details of the DES algorithm.

What is A3/A8 ?

A3/A8 is the generic appellation of the Authentication and Key Agreement algorithm used in GSM networks.

From a random challenge *RAND* (received from the network), and the user's secret key K_i (stored on the SIM card), A3/A8 derives:

- A3 An authentication tag (*SRES*) which proves the knowledge of the subscriber's key to the network,
- A8 A session key (K_c) later used for voice ciphering (A5) between the network and the mobile.

What is A3/A8 ?

A3/A8 is not fully specified, only its interface is:

- Inputs *RAND* and K_i must be 128 bits long,
- Output, from which are extracted *SRES* and K_c, also have 128 bits,
- Algorithm details are left to the operator.

Of course AES could be chosen ...

...but actually many operators prefer to use their own proprietary algorithm with undisclosed specifications.

What is recovered ?

In 2003, R. Novak [Nov03] (ANCS'03) first described a way to partially reverse engineer some actual instance of A3/A8:

- With little knowledge of the algorithm (the structure of the very beginning), he devised a way to recover the content of one substitution table (out of two).
- The knowledge of the other substitution table and the secret key K_i must though be known.

This attack has been improved in [Cla04] (ePrint report 2004/049):

- Both tables and the user's key are disclosed.
- The attack feasibility has been verified by a concrete implementation in black box conditions.

The attack principle

Side channel assumption

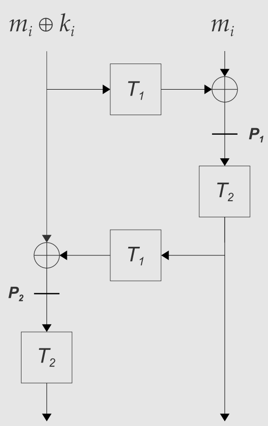
It is possible to detect whether intermediate values at two different instants (possibly on different curves) are identical.

Actual values remain unknown, but local collisions are detected.

Not so easy in practice:

- This assumption is not verified in the (perfect) Hamming weight model,
- Feasible under the Hamming distance model with simultaneous measurements with respect to several reference states.

Novack's attack



The attacker knows that the first computations consist in combining the random challenge $RAND = (m_i)_{i=0,\ldots,15}$ with the key $K = (k_i)_{i=0,\ldots,15}$ by means of 16 applications of the hereabout function.

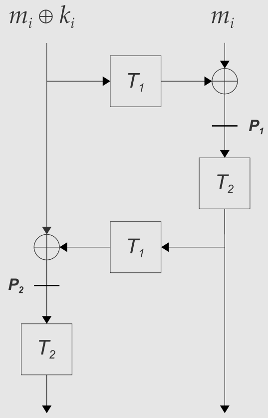
The rest of the algorithm does not matter.

T_1 and K are supposed to be known.

Local collisions at point P_2 are exploited.

Unknown T_2 is to be retrieved.

Novack's attack



A local collision at point P_2 implies:

$$T_1(T_2(x)) \oplus (m_i \oplus k_i) = T_1(T_2(x')) \oplus (m'_j \oplus k_j)$$

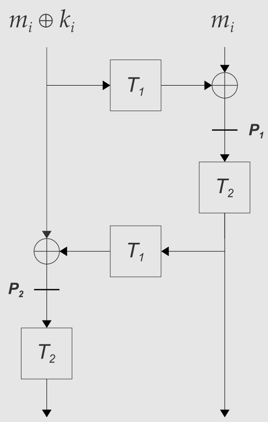
One thus collects relations like:

$$T_1(T_2(x)) \oplus T_1(T_2(x')) = d$$

with known values:

$$\begin{cases} x = T_1(m_i \oplus k_i) \oplus m_i \\ x' = T_1(m'_j \oplus k_j) \oplus m'_j \\ d = (m_i \oplus k_i) \oplus (m'_j \oplus k_j) \end{cases}$$

Novack's attack



$$T_1(T_2(x)) \oplus T_1(T_2(x')) = d$$

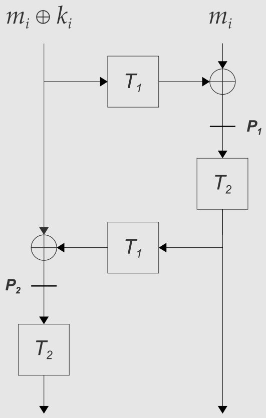
Each such relation links together two T_2 entries (for indices x and x').

By collecting and exploiting enough relations, all T_2 entries are determined relatively to each others.

T_2 is revealed up to the knowledge of $T_2(0)$.

The right valuation of the table is identifiable by DPA/CPA.

First improvement



Novak's attack drawback:

- Needs the knowledge of one substitution table (T_1) in order to retrieve the other (T_2).

It is possible to follow the same principle in order to recover T_1 with sole knowledge of the key.

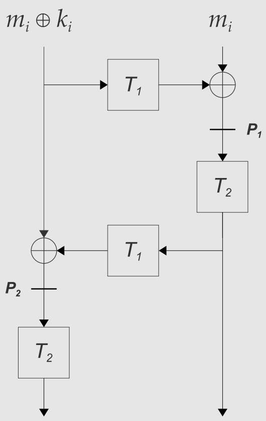
One exploits local collisions at point P_1:

$$T_1(m_i \oplus k_i) \oplus T_1(m'_j \oplus k_j) = m_i \oplus m'_j$$

T_1 is so retrieved up to the knowledge of $T_1(0)$.
(Right valuation identifiable by DPA/CPA)

Reverse Engineering of Unknown Algorithms | A SCARE attack against an A3/A8 algorithm

Second improvement



It is possible to retrieve T_1 without knowing the key!

Successive key bytes are progressively guessed.

Wrong guesses imply contradictions amongst constraints about T_1 and are eliminated.

T_1 is so revealed up to the knowledge of $T_1(0)$ and one key byte value (e.g. k_0).

(Correct values of $T_1(0)$ and k_0 are identified by DPA/CPA)

T_1 is retrieved from scratch ...
... as well as the secret key!

Lesson

It is possible to recover both substitution tables from no prior knowledge:

- First, retrieve T_1 and the key from scratch, (improved attack)
- Then, apply basic attack to retrieve T_2.

Lesson

Secret specifications may be jeopardized by side channel analysis (SCARE)

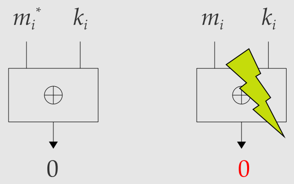
Other possible threat?

Fault Injection for Reverse Engineering (FIRE)

Key Recovery with Unknown Algorithms

A trivial example

A chosen message Collision Fault Analysis on AES allows a key recovery by causing output collisions:



$$\Longrightarrow \quad AES_K(M^*) = \overset{\leadsto}{AES}_K(M)$$

A crucial remark is that knowledge about what happens after the XOR is not needed for the attack to work.

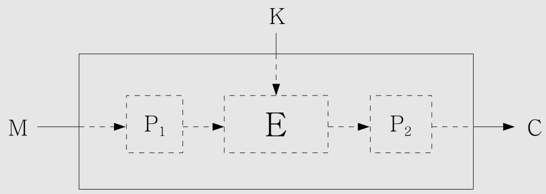
Consequence

This key recovery attack generically applies to any algorithm beginning with the $M \oplus K$ operation. Except if M is involved again afterwards.

Obfuscation to prevent from Fault Analysis

Any known transient Fault Analysis on a cryptographic algorithm requires the knowledge of either the input (CFA) or the output (DFA).

Designing a proprietary and secret algorithm could be achieved by *obfuscating* inputs and outputs of a given well known block cipher E (DES, AES, ...) :

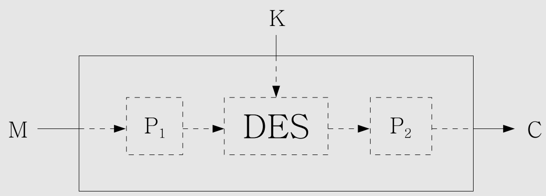


- P_1 and P_2 are two secret and deterministic one-to-one mappings.
- The design inherits its security from the core function E.
- Fault analysis should be prevented by the obfuscation layers P_1 and P_2 which hide inputs/outputs of E from the attacker.

The case of obfuscated DES

Fact [Cla07] (CHES'07)

An *obfuscated* DES is not secure against transient Fault Analysis.



The hereafter described attack allows to recover the DES key without any knowledge about P_1 and P_2.

- Also applies on obfuscated 3-DES as well
- Practically relevant since such constructions actually exist

The attack model

Fault model

When a fault is injected during an 8-bit XOR instruction, its output is zero whatever the inputs.

Attacker model

- The *obfuscated* DES is straightforwardly implemented in software on an 8-bit architecture.
- The attacker controls inputs of the algorithm, and knows its outputs.

The attack principle

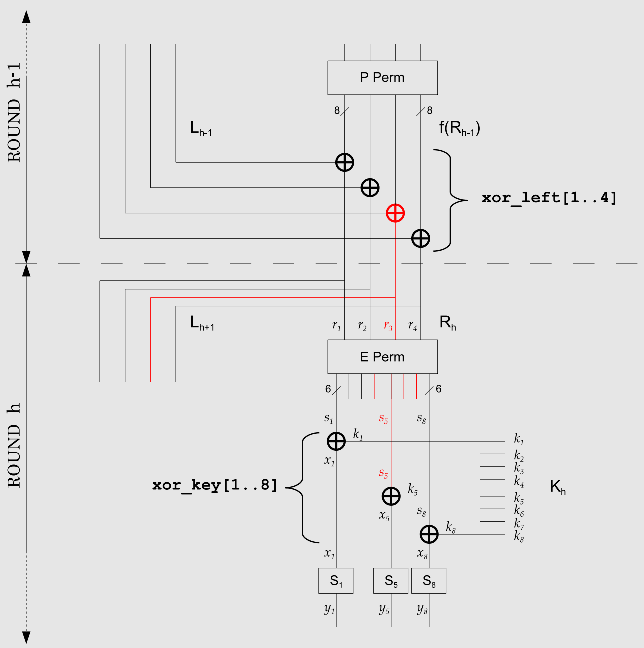
Fault as a probing tool

By comparing the outputs of two executions (one normal, one faulty) with same input, one infers whether the normal output of the faulted XOR is zero.

Putting together that the normal outputs of two related XOR instructions are simultaneously equal to zero, it is possible to infer some information about the key.

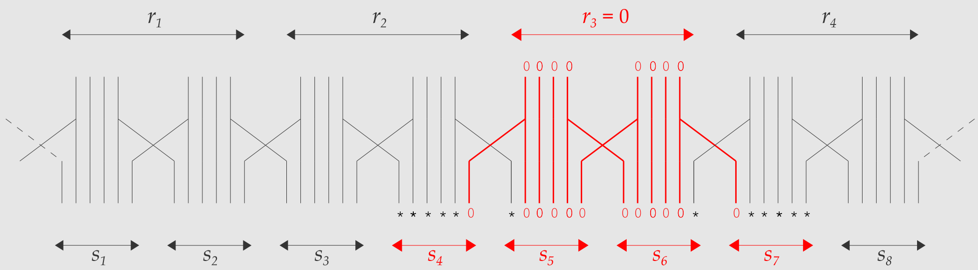
Remark: ‘simultaneously’ means *for the same input*, not *on the same execution*.

Indeed, the attacker does not need to inject ‘multi-faults’.



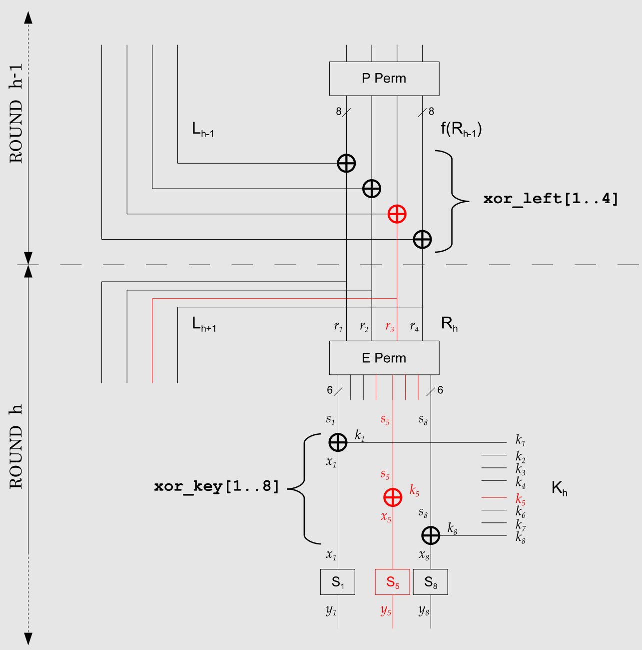
$M \xrightarrow{fault} \overset{\leadsto}{C}$

For some input M, observation that $C = \overset{\leadsto}{C}_{\texttt{xor\_left[3]}}$ implies that $r_3 = 0$



$r_3 = 0$ implies that s_5 and s_6 are almost zero after the expansive permutation.

Knowing that $s_5 \approx 0$, it may be interesting to known what happens when next XOR is also faulted: $\overset{\rightsquigarrow}{x_5} = s_5 \overset{\rightsquigarrow}{\oplus} k_5$.



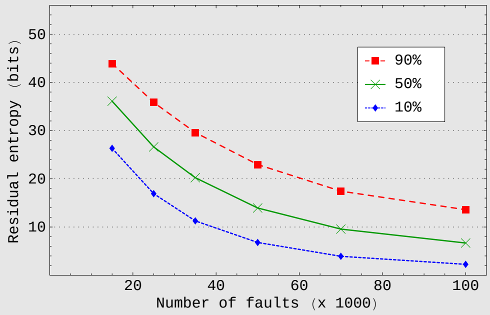
If for the same input M, one also observes that $C = \overset{\rightsquigarrow}{C}_{\texttt{xor\_key[5]}}$, then:

$$k_5 \in s_5 \oplus S_5^{-1}(0) \approx S_5^{-1}(0)$$

Each *double ineffective fault* gives some information bits about the round subkey.

Exploiting all these events (and others) all along the DES allows to (quasi fully) recover the key.

A drawback is the important number of fault injections that are needed:



This could be seen as the price to pay for the *magic property* that the key is retrieved without knowing anything about the two obfuscating secret shuffles P_1 and P_2.

Lesson

It is possible to retrieve a DES key by transient fault analysis, even when inputs/ouputs are unknown from the attacker.

Lesson

Secret specifications do not prevent from key recovery.

Open problems and conclusion

Open problems

- Is it possible to reverse engineer a fully secret algorithm by means of side-channel signal and/or transient faults exploitation?
- Is it possible to recover the key of a fully secret algorithm by means of side-channel signal and/or transient faults exploitation?
- Is it possible to do that in a generic way?

Conclusion

- *Security through obscurity* does not prevent from physical attacks.