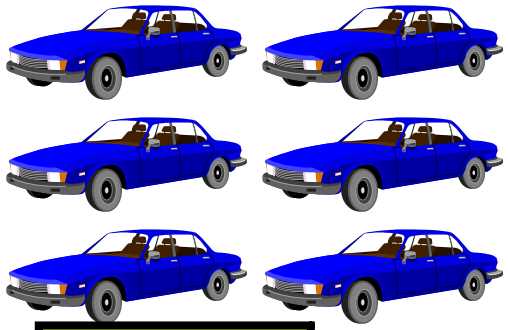
The Sorcerer's Apprenctice's Guide to Fault Attacks

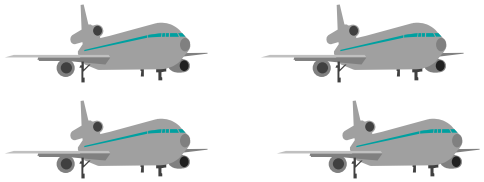
Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, Claire Whelan

WHAT IS THIS ABOUT?

Broken toys are not charged to our clients



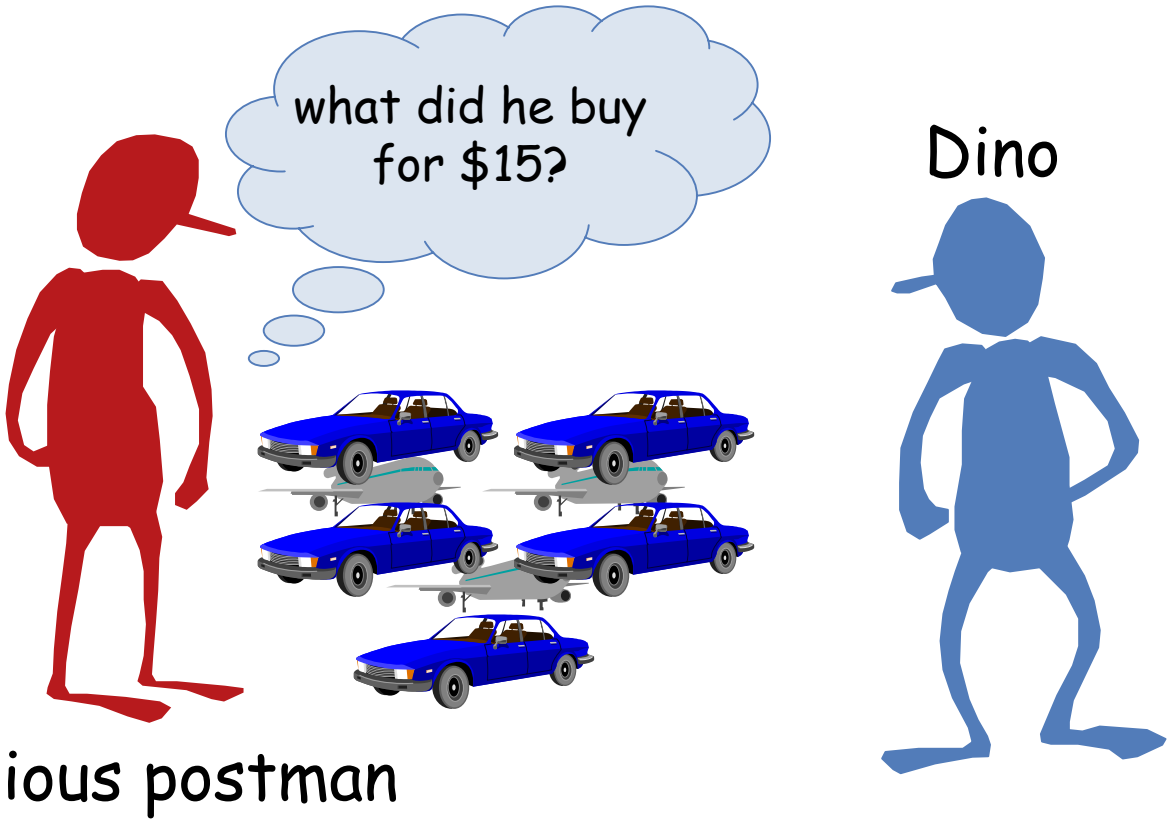
car = $3



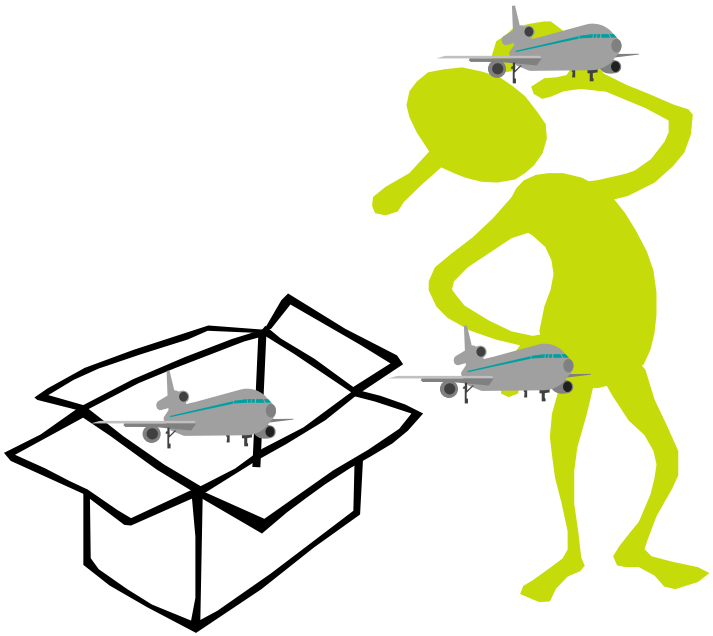
plane = $5

Dino buys toys from Jack

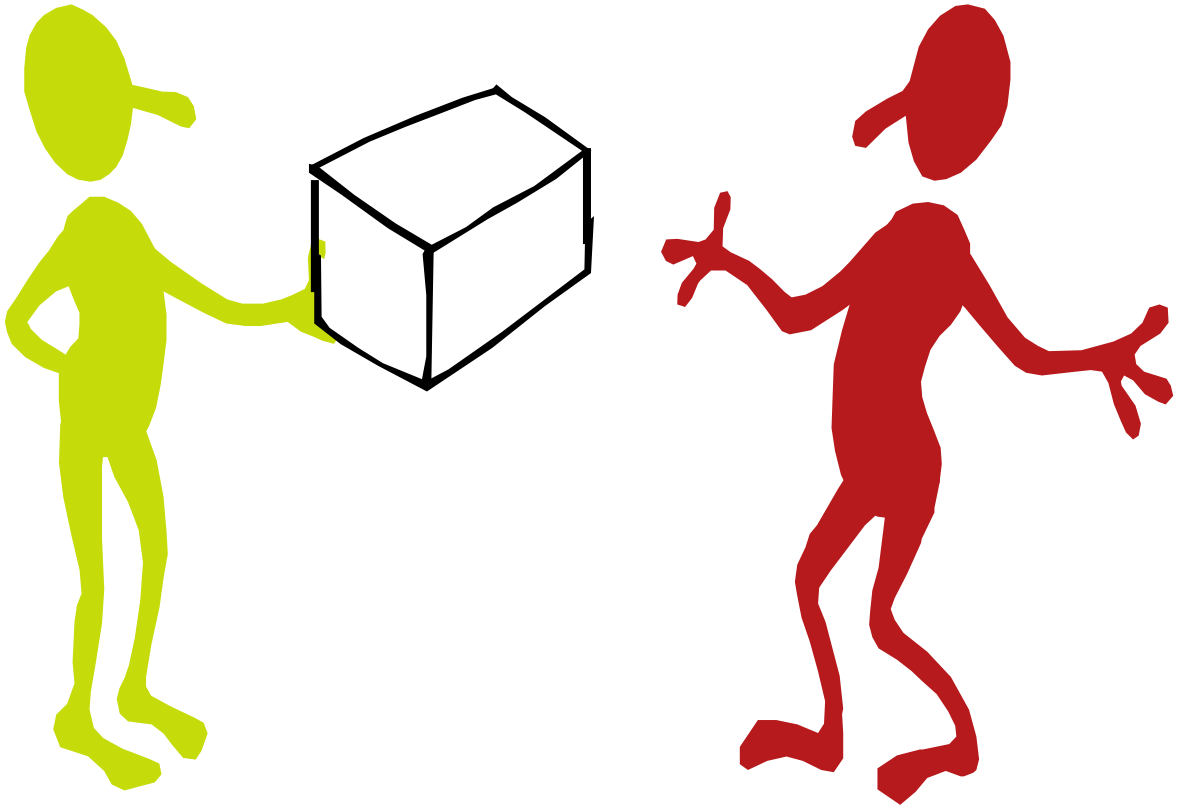
The postman wants to know what Dino bought for $15



In the meanwhile Jack prepares the DHL

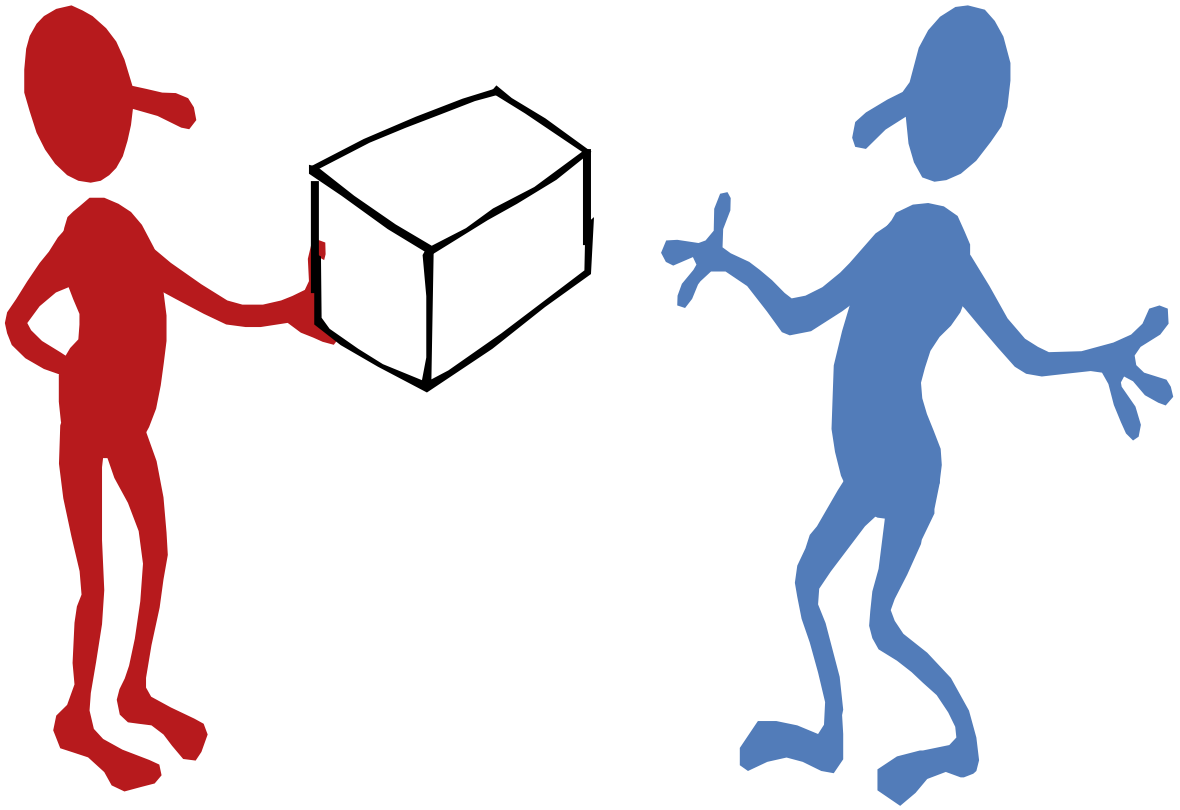


and gives it to the postman



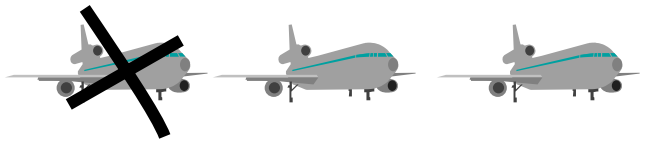
Who kicks it strong enough to break one toy

and gives it to Dino



a week later he monitors Dino's postal order...

= 4 × 3 = $12



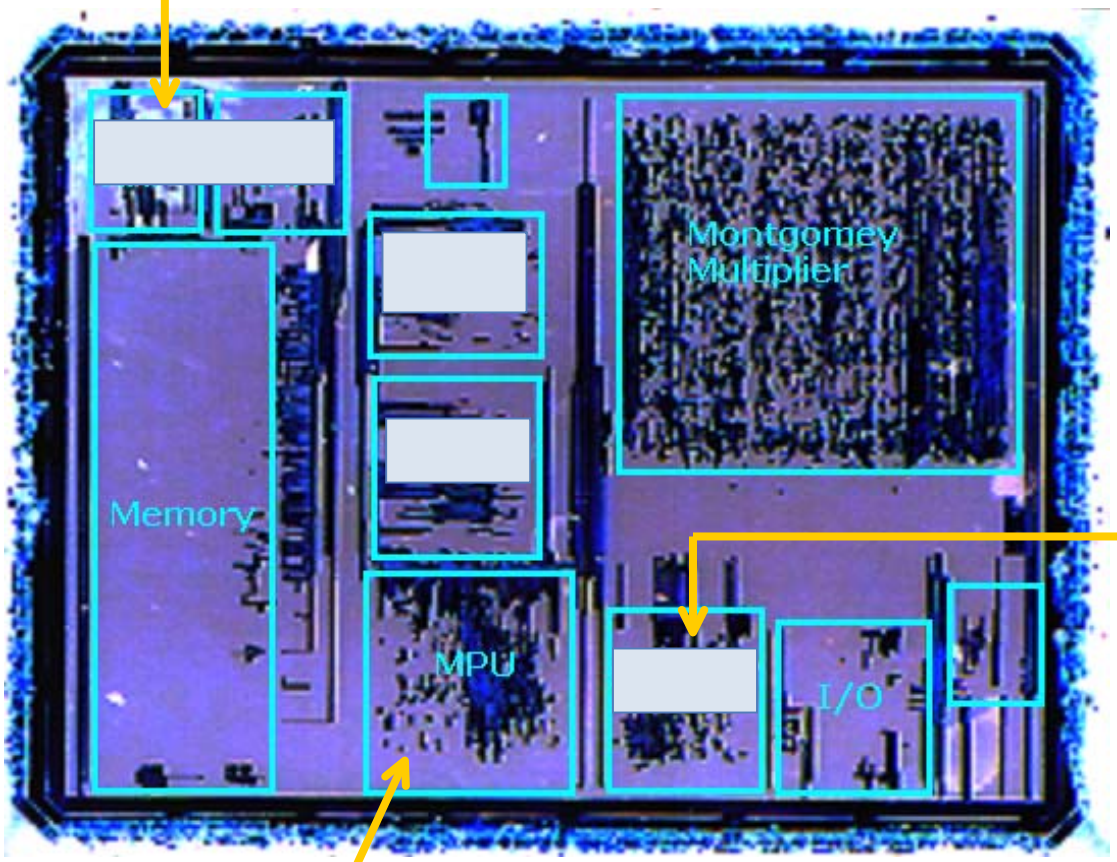
= 2 × 5 = $10

Lesson learned: **Fault attacks** can also extract secrets from tokens!

Hardware faults can have various sources:
voltage glitches, light beams, laser beams...

How is this done experimentally?

An Experimental Chip



Synchronous µP

Clocked processor

Secure µP

Asynchronous processor

Dual Rail Implementation

Memory Encryption Unit

The Target

- Currently used in
  - *Pagers, cordless phones, automotive electronics, radio systems,…*
- General Purpose 16-bit RISC processor
- 2 stage pipeline
- Very basic instruction set
- Experimental version with 4 usable registers

Expected Behaviour

- Asynchronous Processor
  - Low EM emitter => *immune against EMA*
  - Can work at very low voltages => *resistant to glitches*

- Dual Rail coding with RTZ
  - Constant Hamming weight data
  - Same number of transitions

  } *counteracts any PA or EMA*

  - Unused 1-1 state is propagated and causes circuit to deadlock => *resistant against Light fault injection*

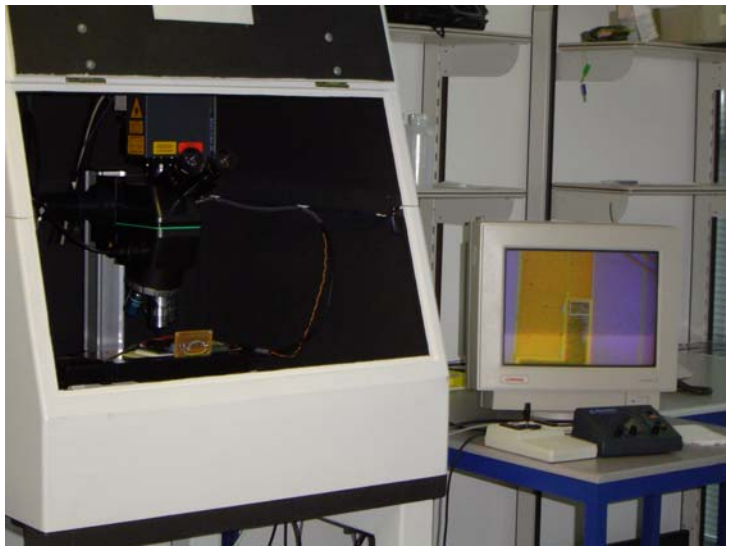
Aim of Injecting light / laser

- Test the effectiveness of the *dual-rail* encoding of the asynchronous design
- Set-up used is similar to the previous one
- We run a short piece of code which is synchronised with the light/laser shoot
- The time of the light injection and the behaviour of the processor are monitored by
    - by monitoring the code execution via the power curves themselves
    - by observing the effects on the results of the XOR

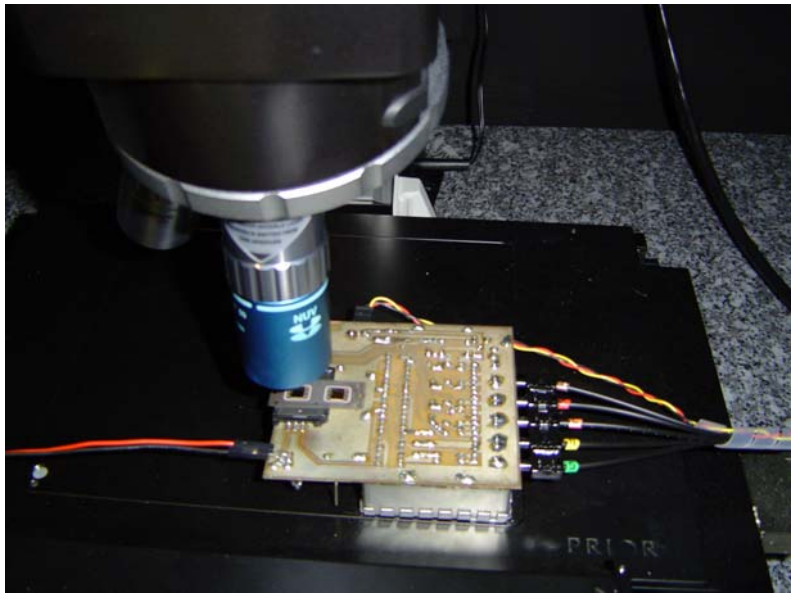
Injecting white light

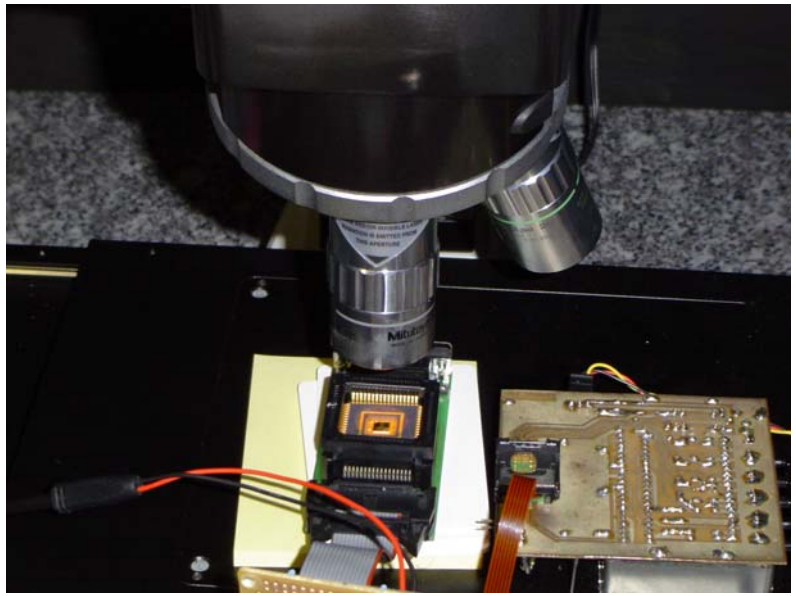
- We injected pulses of white light onto the entire chip.
- No particular effect was observed except for punctual increases in the power consummed at the instants the light pulses were injected.
- This could be explained by
  - 1° The weakness of our light source
  - 2° The layer of metal filings that, according to us, filters out the light injected

Laser

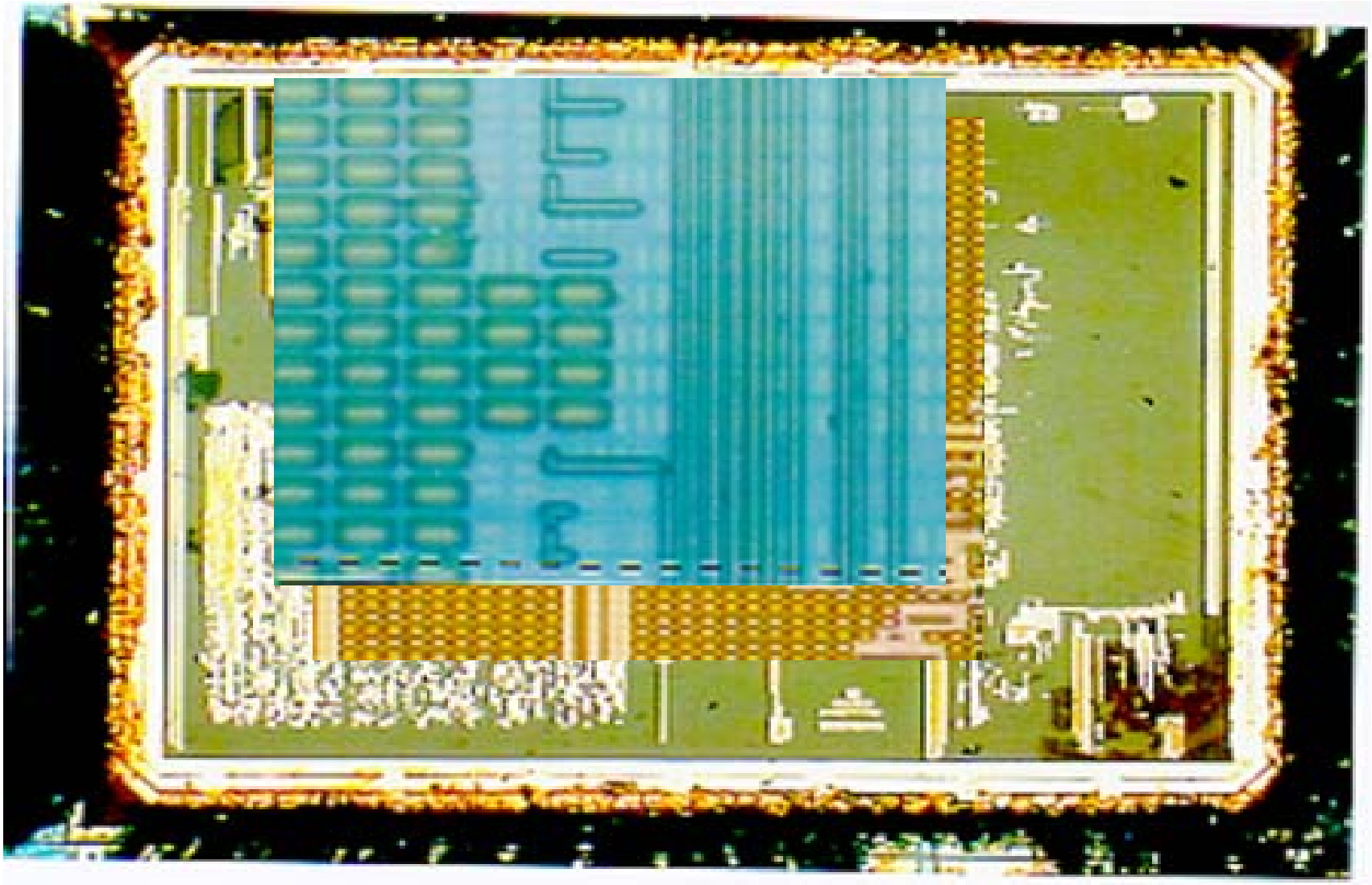


Laser

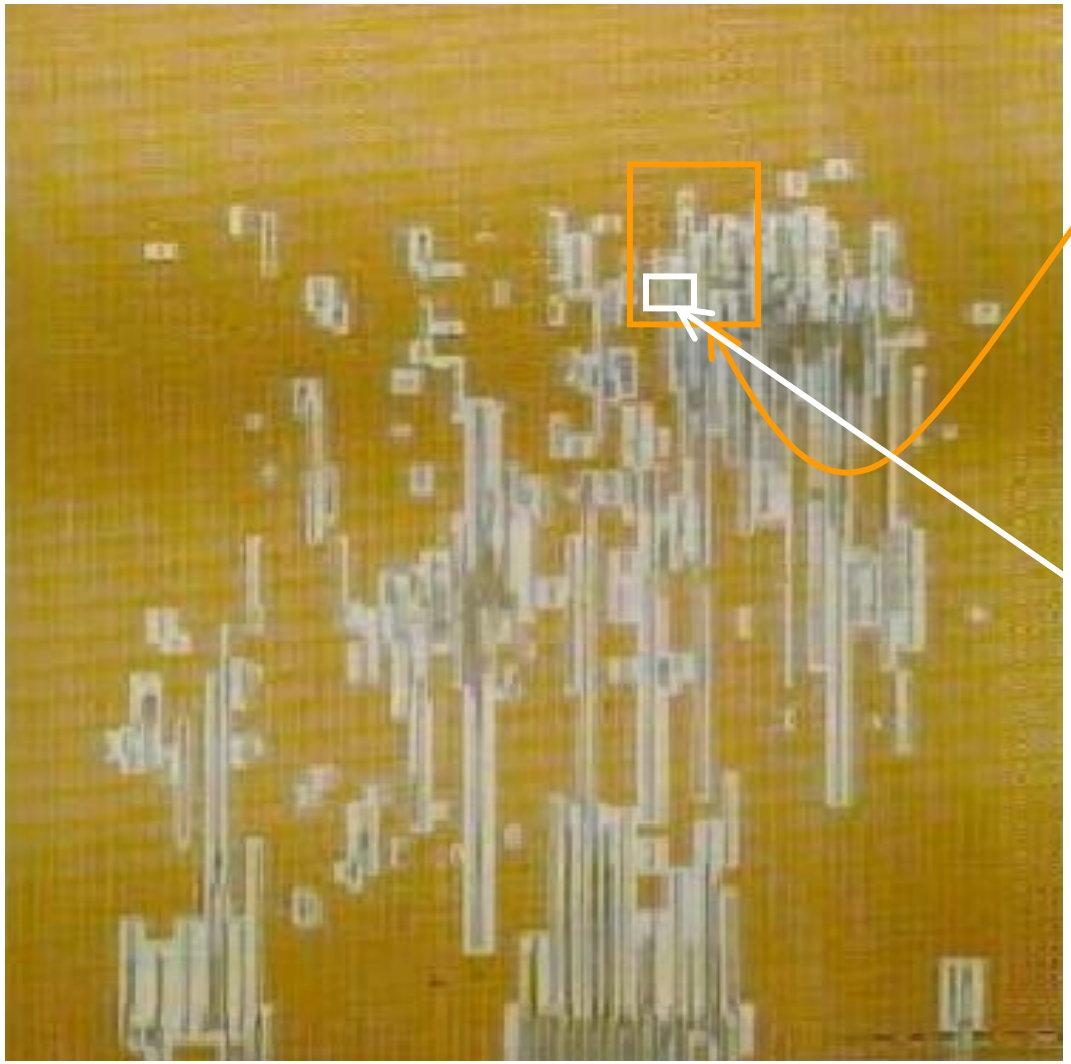




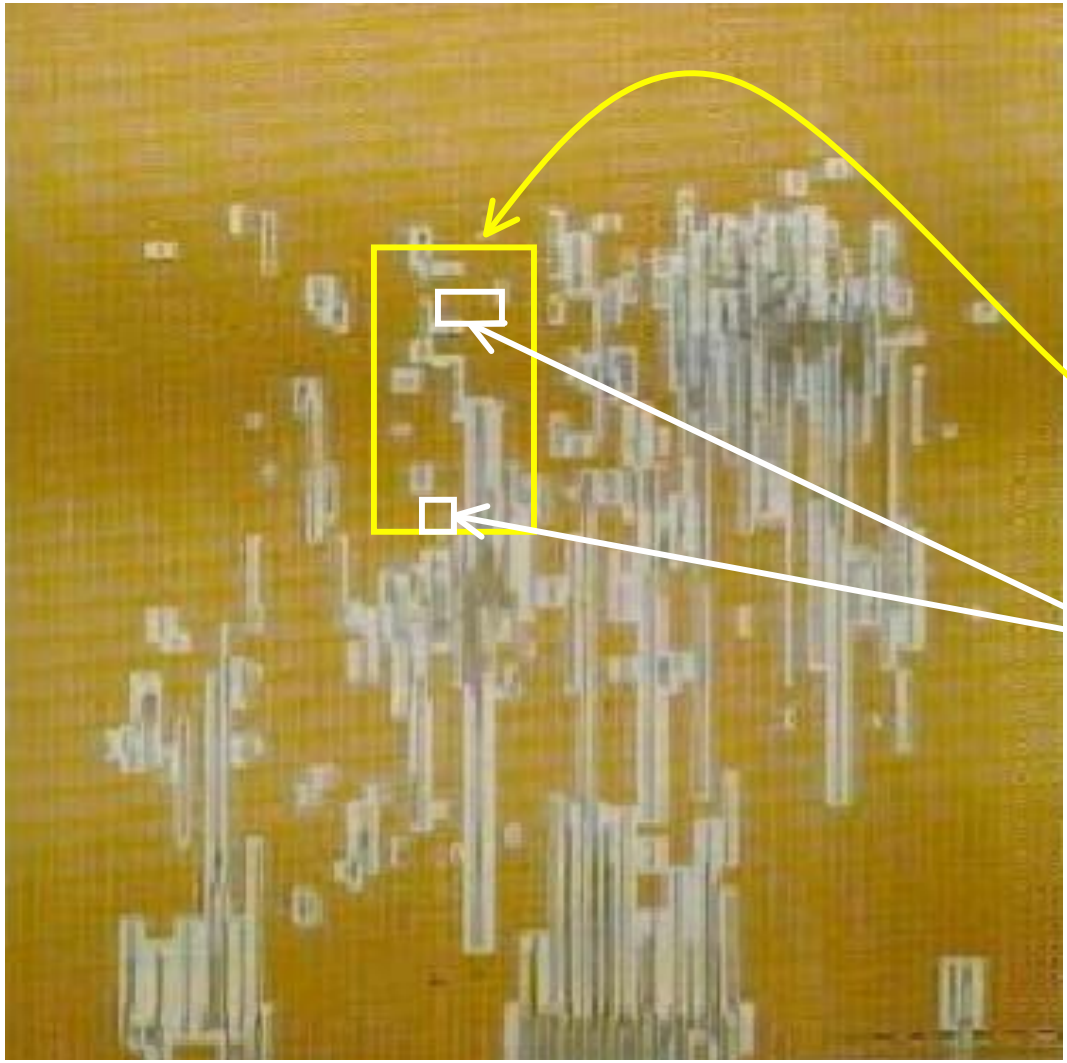
Target Chip



Laser on the AL-AH registers



Laser on the ALU

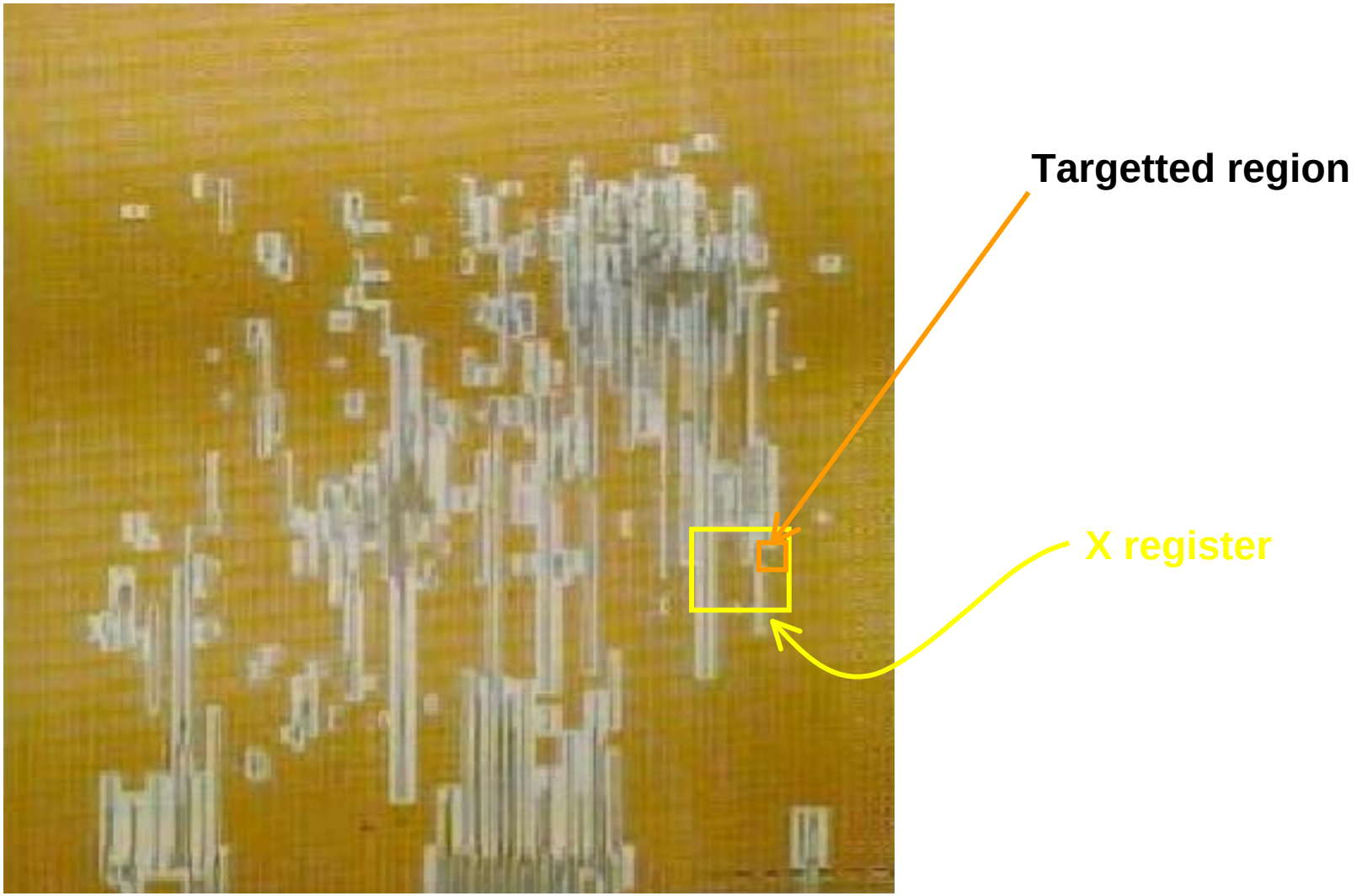


Regions where faulty behaviours were obtained

Laser on the ALU

- Faulty behaviours occurred only during the execution of the XOR (time T3)
- Two ‘families’ of effects could be observed
  - The result of the XOR is false, e.g.
    - `0x0001 xor 0x0013 giving 0x0025`
    - `0x0001 xor 0x0010 giving 0x0023`
    - `0x0001 xor 0x002D giving 0x0059`
    - `0x0001 xor 0x0028 giving 0x0053`
  - The result of the XOR is always `0x0001`

Laser on the X register



Laser on the X register

- No matter when we shot the laser (T1-T4), the result returned for the XOR execution never corresponded to the arguments passed
- In our program, the X register contains the base address at which the data are loaded from and stored to.
- The X register is expected to contain `0x12`
- A memory dump showed that data at addresses like `0x52`, `0x92` or `0x112` were unnecessarily modified !
- We corrupted the value read from register X

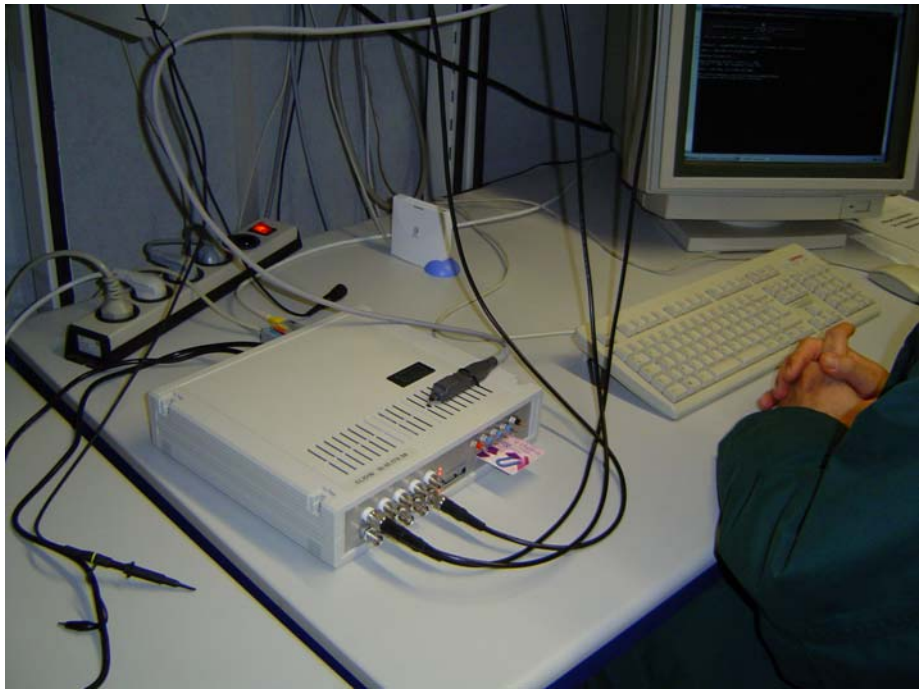
Laser attack on *Secure µP*

- A careful positionning of the laser beam gave exploitable faulty behaviours
- The apparent poor resistance of the registers is due to single flip-flop implementation
  - => It’s not enough to have buses only in dual rail!
- The behaviour of the ALU, which is in dual rail, has not been explained.

Short Glitches on the Secure µP

- Kept the same program as executed for the laser experiments
- Used glitches where power drops from 1.8V to 0V for *d* ns
- For short *d*, the processor would just stop and just resume normal execution when power is restored
- We monitored the instant the fault was injected: falling edge of glitch was kept constant (just after 1st IO) and we moved the rising edge (by varying *d*)

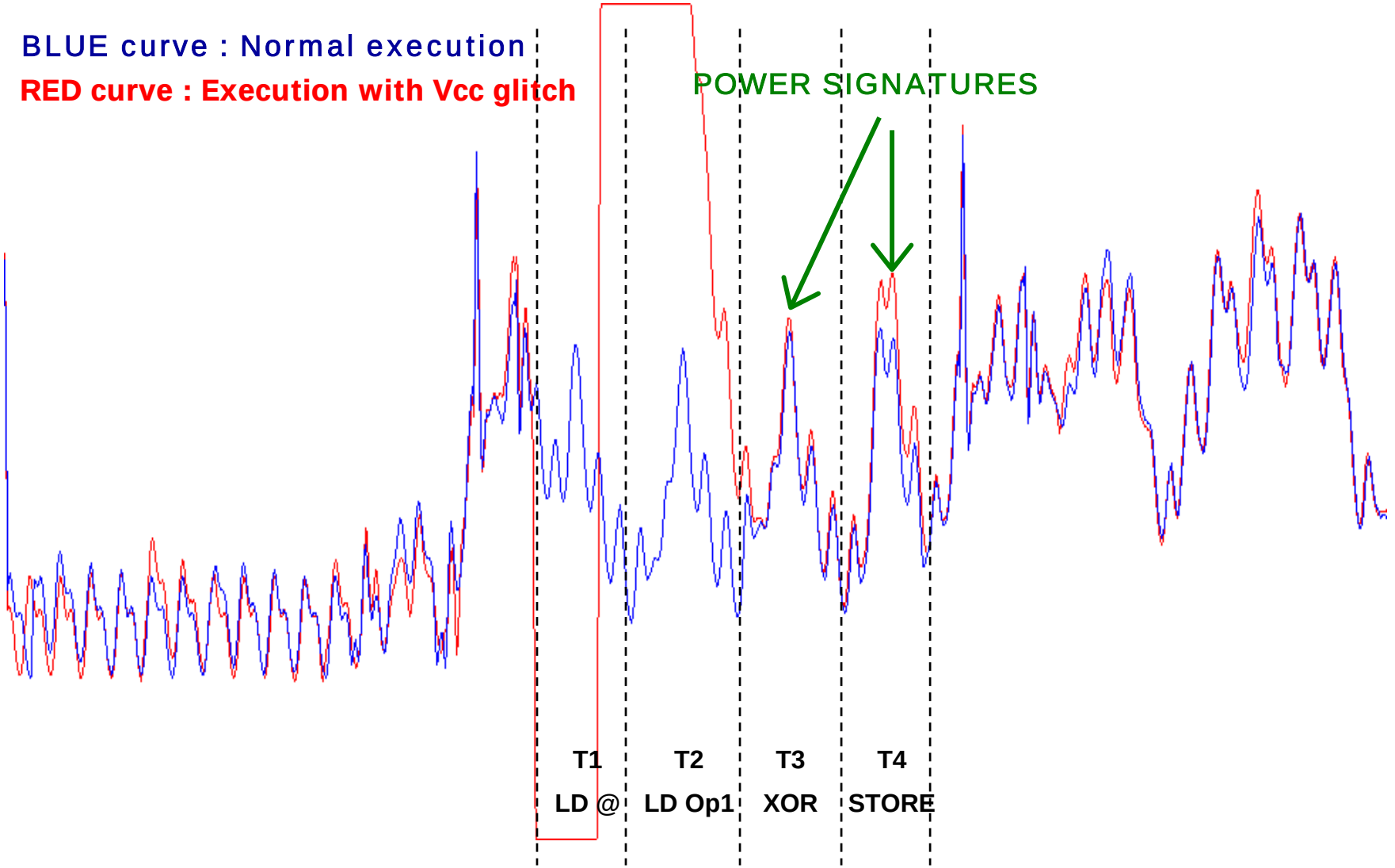
CLIO Glitch Injector



Corrupting LOAD of 1st operand

- Rising edge of glitch occurs at instant T2
- The result of the XOR operation was the logical inverse of second operand
- The XOR operation executed was between the second operand and 0xFFFF as first operand !

Corrupting 1st operand



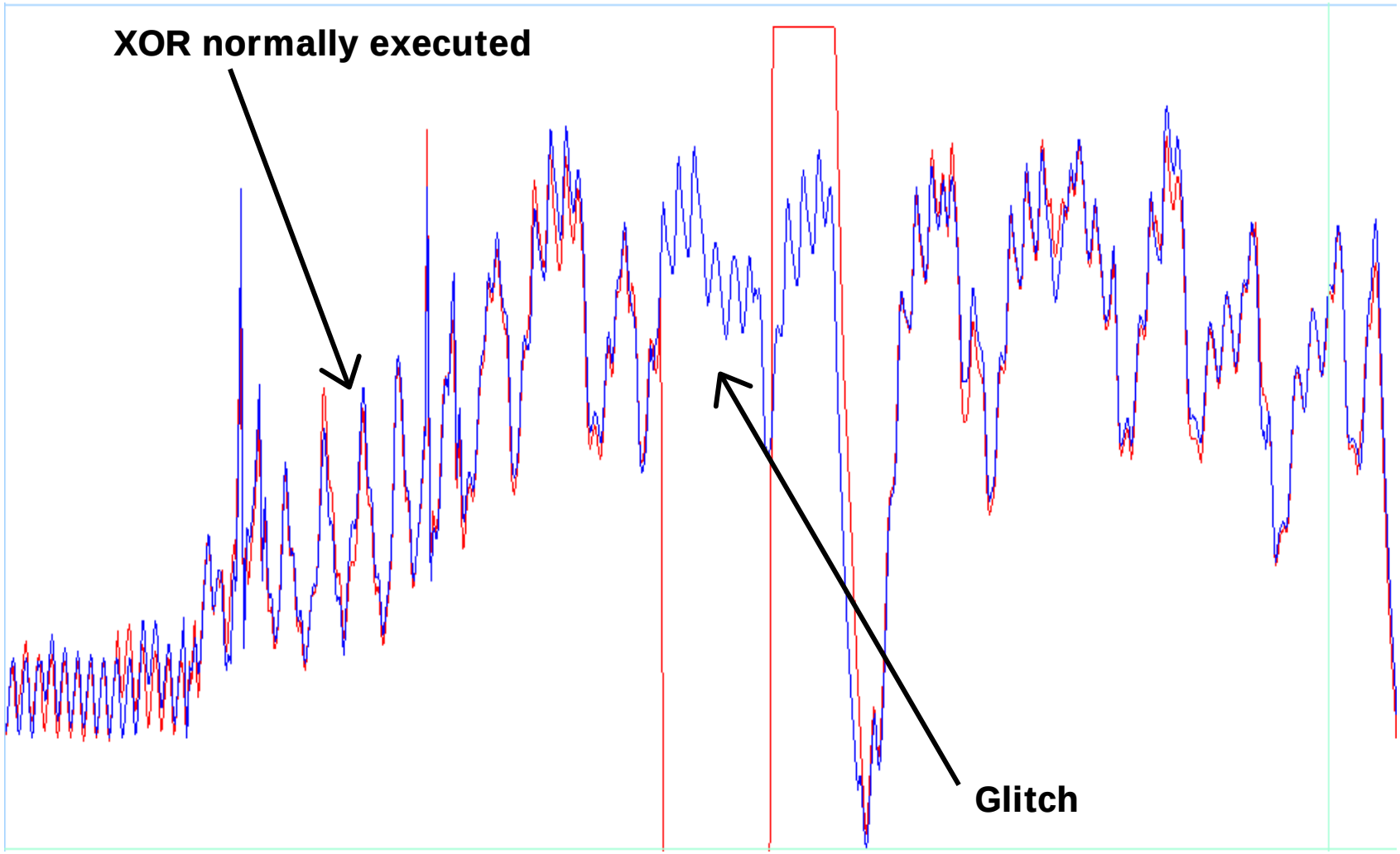
Modifying STORE of result

- Likewise, we managed to make the rising edge of the glitch coincide with the STORE instruction (T4)
- As a result, the memory content corresponding to the @ at which the result should be contained was never ‘updated’...
- We corrupted the execution of the STORE
- If we looked only at the result, it’s as if the XOR operations never took place

Dumping data memory

- By increasing to the max the duration of the glitch, we dumped the data memory on the Dual-Rail XAP
- Instead on sending and displaying 3 words from the data memory, we had **51** consecutive 16-bit words from the data memory

Dumping data memory



Glitches on the *Asynchronous* µP

- We put the first operand to 0xFFFF
- We short-circuited the STORE of the result
- We modified the value and the quantity of the data sent on the UART
- For the non protected-µP we dumped 51 words from data memory
  - The Sec µP and the non protected -µP having the same nature, there is no reason for not reproducing the same effect on the Sec-µP: the glitch generated did not happen at the right time, the Sec-µP being slower than the non protected-µP

Gemplus' Internal CQP

```
loop on CEVA commands and on hit t's
    {
    dichotomy loop on Laser intensity / glitch amplitude
        {
        XY table loop on X
            {
            XY table loop on Y
                {test normal behaviour}}}}
```

simplified fault qualification campaign takes around 4 weeks
thorough campaign may take a couple of months
if the campaign fails ⇨ internal CSP process

What Is The State of The Art?

Card manufacturers and chip manufacturers are not evenly protected

Nearly no protections in newcomers' chips (all remains to do)

Protection by blindly stacking protections (ineffective)

Uneven investment level by manufacturers

Academia is taking over… ☺

Experimental
Differential
Fault
Attack
on RSA

Fault injection step

- In the RSA experiment, the fault injection mean was the laser

- The appropriate set-up must be performed for the following parameters:
  - space localisation (x1,y1)
  - Fire window size (x2,y2)
  - Light intensity
  - Light wavelength

- Finding the proper injection parameters is quite difficult

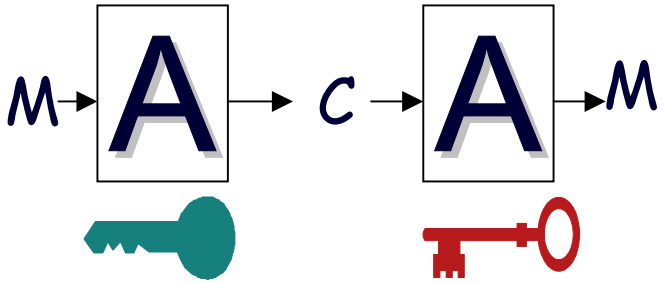
Fault exploitation step

- The target of the attack is the RSA CRT algorithm

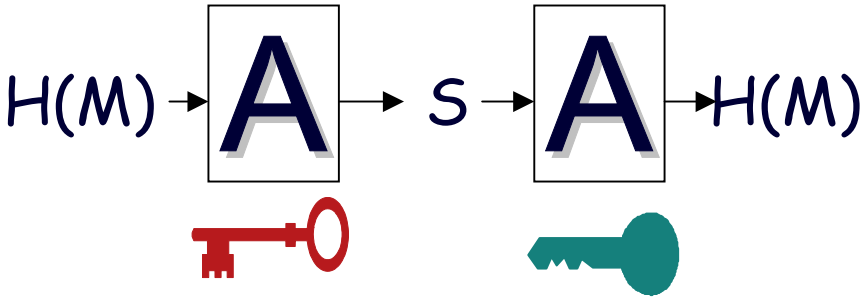
- The fault exploitation scheme is well known as the Lenstra attack.

RSA

- Two modes:
  - Encryption:
    - One for encryption (public)
    - One for decryption (secret)



  - Signature:
    - One for signature (secret)
    - One for verification (public)



Chinese Remaindering Remainder

- The Chinese Remainder Theorem is used in RSA in order to speed up exponentiation.

- Exponentiation is performed in three steps
  - $s_p = m^d \bmod p$ is computed
  - $s_q = m^d \bmod q$ is computed
  - the signature is recombined with CRT as

$$s = a.s_p + b.s_q \bmod n,$$

- The constants a and b are precomputed such that

$$a = 1 \bmod p, \qquad b = 0 \bmod p,$$
$$a = 0 \bmod q, \qquad b = 1 \bmod q.$$

Attack on CRT exponentiation

- This attack was first published by Lenstra.

- Hypothesis:
  - *s*, signature of a message *m* is known.
  - a fault is injected in the exponentiation mod *p.*

- Due to error injection, s_p becomes s_p'

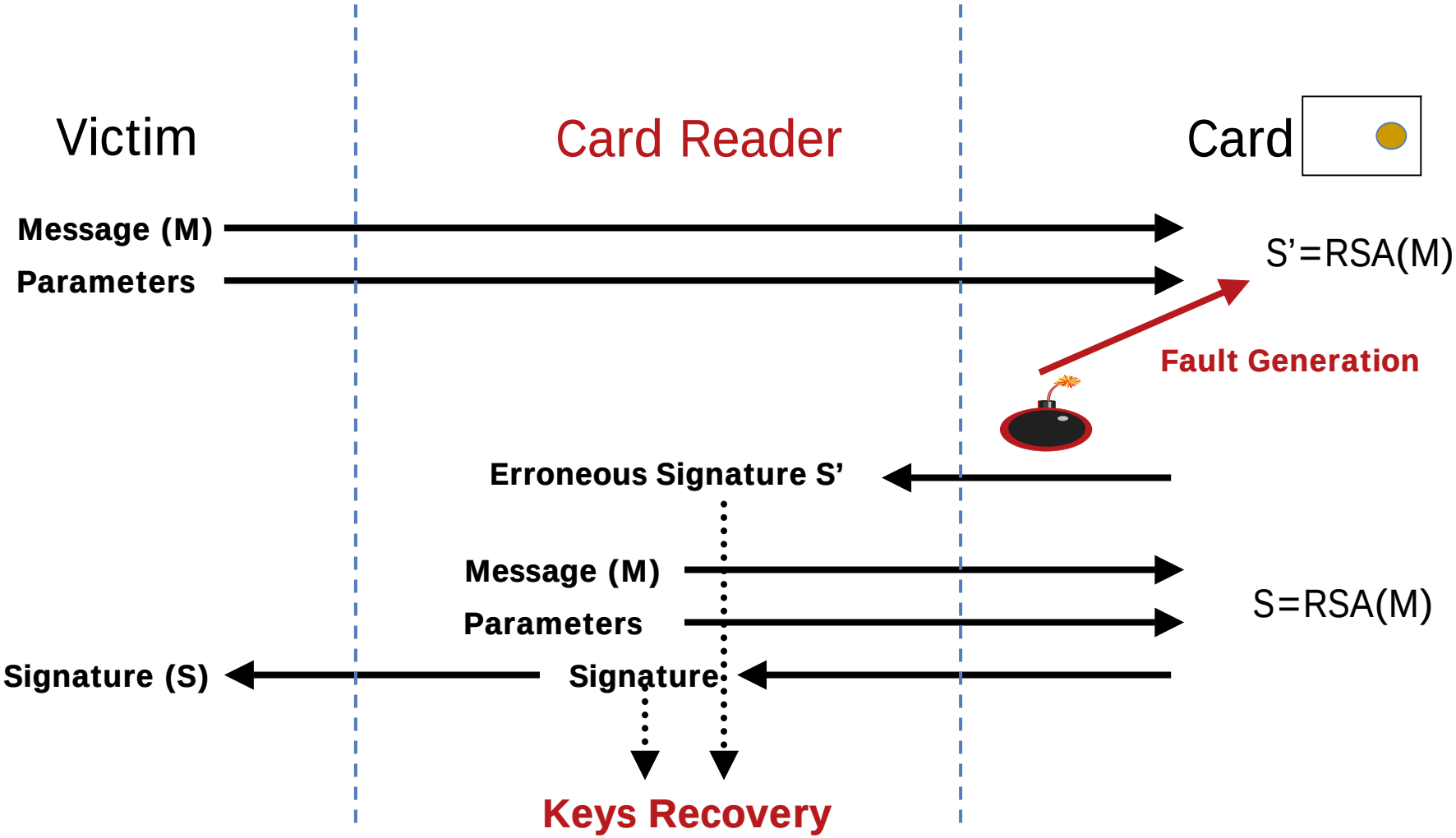
$$s' = a.s_p' + b.s_q$$
$$s'-s = (a.s_p' + b.s_q) - (a.s_p + b.s_q)$$
$$s'-s = a.(s_p' - s_p)$$

the prime *q* divides *a* and can be retrieved by GCD

Tools

- Public key (N,e)
- One execution during target algorithm to have a right signature S
- One execution with the fault injected during one of the two CRT exponentiation calculation to get the wrong signature S’

Operating mode



Experiment 1: Insecure design

- Attack performed on RSA CRT: the fault is introduced during one of the two exponentiations
- The fault is not detected and therefore it is exploitable by the external world
- The fault directly lead to the secret recovery by a simple GCD computation
- Fault attack was successful because:
  - Component (HW) is sensitive regarding the fault injection mean (laser)
  - The fault injection equipment have been properly set-up
  - RSA CRT implementation (SW) is sensitive to differential fault analysis, fault is injected at appropriate instant in time

Experiment 2: Secure design

- Attack performed on RSA CRT: the fault is introduced during one of the two exponentiations
- An error status word 92 00 is returned by the card
- The fault is detected and the false signature is not returned to the external world
- The fault is not exploitable
- Fault attack was thwarted because:
  - A software detection mechanism has detected the false signature
  - The false signature is not returned to the external world
  - The fault injection step is successful but the exploitation step is not possible anymore

General Defense Strategy Used

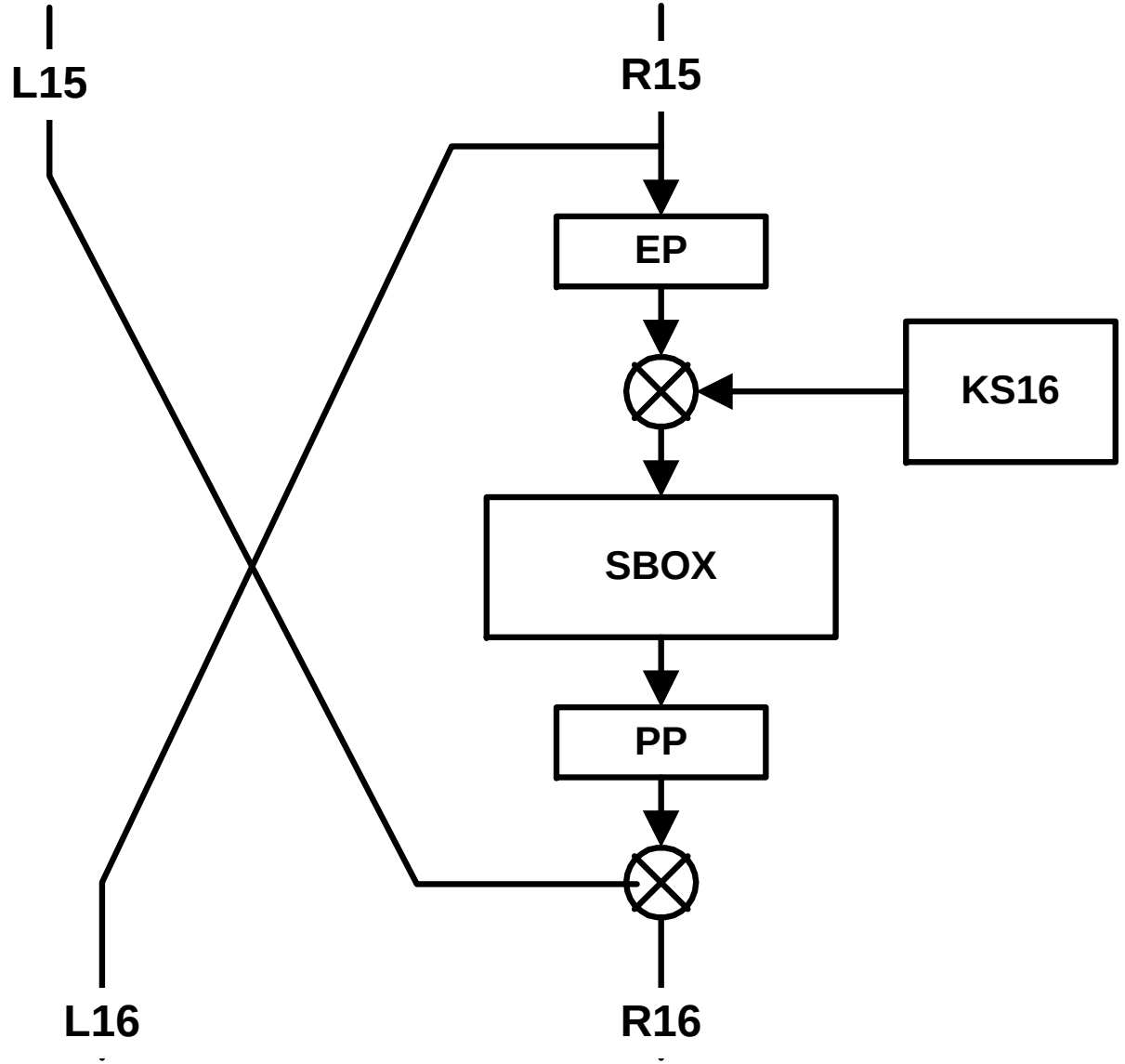
Resist - Detect - React

Experimental
Differential
Fault
Attack
on DES

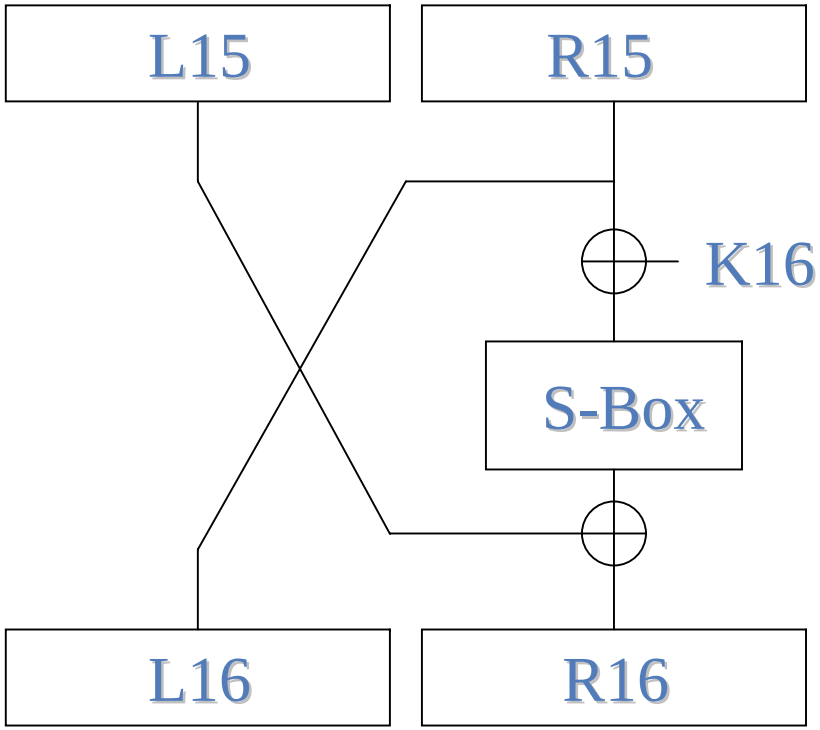
Fault exploitation (DFA) in a regular DES/3DES algorithm

Fault exploitation (DFA) in a DPA protected DES using randomised SBOX stored in RAM

DES Round



15th round DPA



- The Last round of the DES

  Transfom of [L15,R15] to [L16,R16] using K16

  Permutations are ignored for convenience

$$L16 = R15$$

$$R16 = S(R15 \oplus K16) \oplus L15$$

15th round DPA

$$L16 = R15$$

$$R16 = S(R15 \oplus K16) \oplus L15$$

- If R15 is changed to R15', without changing L15

$$L16' = R15'$$

$$R16' = S(R15' \oplus K16) \oplus L15$$

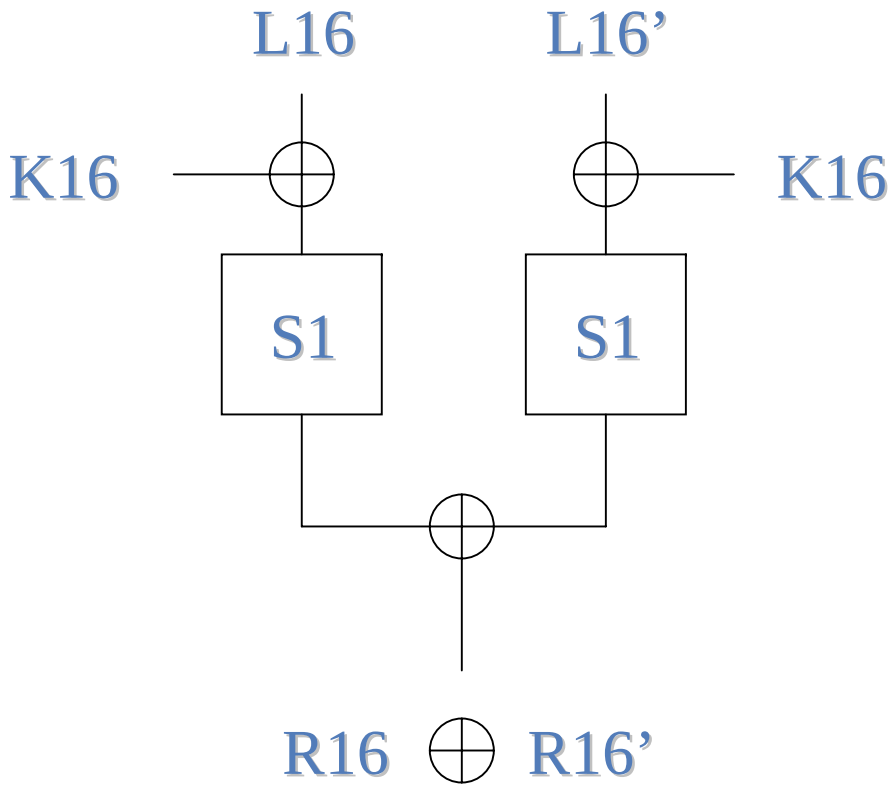
- Then

$$R16 \oplus R16' = S(R15 \oplus K16) \oplus L15 \oplus S(R15' \oplus K16) \oplus L15$$
$$= S(L16 \oplus K16) \oplus S(L16' \oplus K16)$$

- where S(x) is the S-box function

15th round DPA

- For each S-box, verifying:



- Gives a list of possible key values 2^{26}
- Leads to an exhaustive search

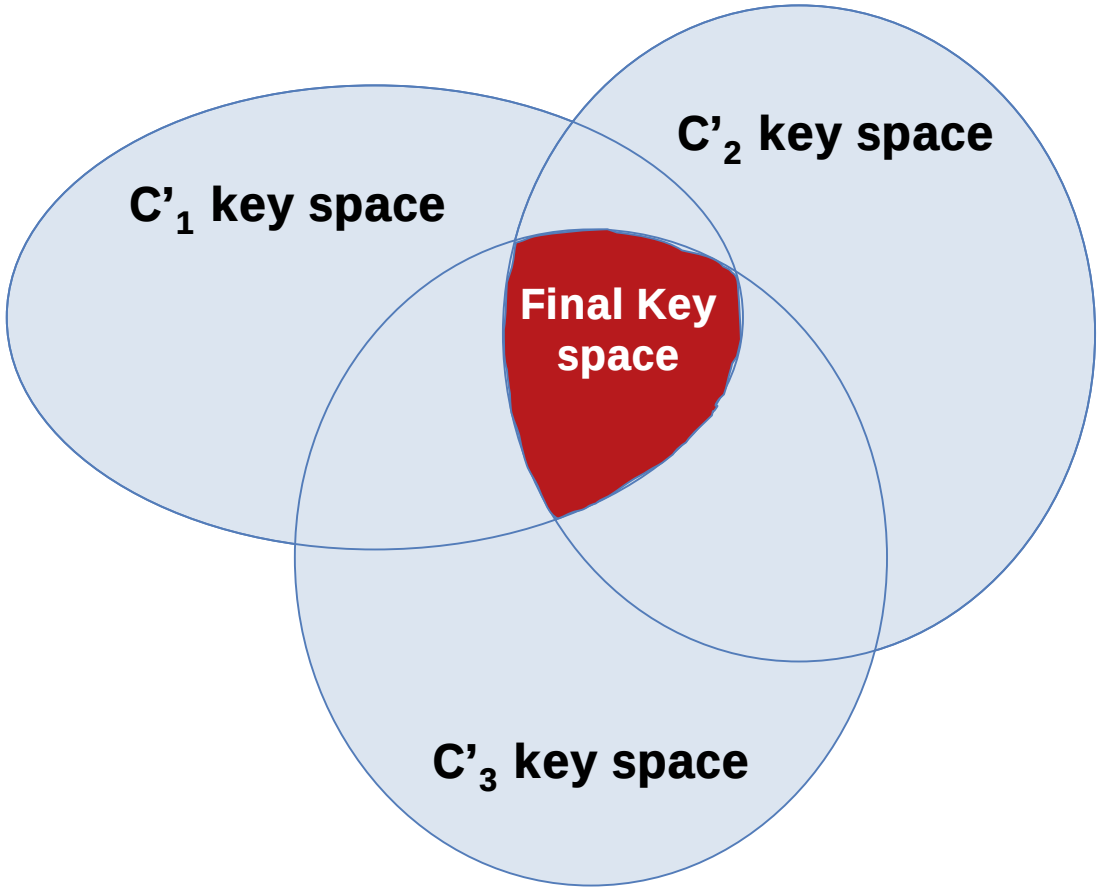
15^{th} round DPA

- The number of hypothesis' given for each six bits of the key can be found using the tables, described in, "Differential Cryptanalysis of DES-like Cryptosystems" by Biham and Shamir

- More faulty ciphertexts rapidly decreases the number possible keys
- Best Results by attacking expansive permutation in fifteenth round

- The same technique can not reasonably be applied to higher rounds (differences in R14 leads to an exhaustive search of 2^{55} DES executions)

15th round DPA



- Multiplying the number of faulty ciphertext directly reduces the final key space size and therefore the complexity of the exhaustive search

Applied to 3DES

- If faults are generated during the fifteenth round of the last DES, and the fifteenth round of the second DES.
- For every hypothesis of the first key we have a list of hypotheses for the second key.
- Sets hypotheses with an impossible differential can be ignored.
- With one fault in the last round and one fault in the second round we have an exhaustive search of 2^{51}

Applied to 3DES

- With two faults in each DES the search can be reduced to DES executions 2^{15}
- The same process can be applied to 3DES with three different keys.
- One fault per key giving a search of 2^{75}
- With two faults per DES key giving a search of 2^{20}

Experimental Implementation

- With glitch injection
- Attack takes less than 2 minutes on a PC.

For More Information, Remember