Laser-induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-bit Microcontroller

**Brice Colombier[1], Alexandre Menu[2],
Jean-Max Dutertre[2], Pierre-Alain Moëllic[3],
Jean-Baptiste Rigaud[2] and Jean-Luc Danger[4]**

[1]Univ Lyon, UJM-Saint-Etienne, CNRS, Laboratoire Hubert Curien UMR 5516
[2]IMT, Mines Saint-Etienne, Centre CMP, Equipe Commune CEA Tech - Mines Saint-Etienne
[3]CEA Tech, Centre CMP, Equipe Commune CEA Tech - Mines Saint-Etienne
[4]LTCI, Télécom ParisTech , Institut Mines-télécom, Université Paris Saclay

Cryptarchi workshop
June 25th, 2019

Fault attacks on 32-bit microcontrollers

Fault attacks

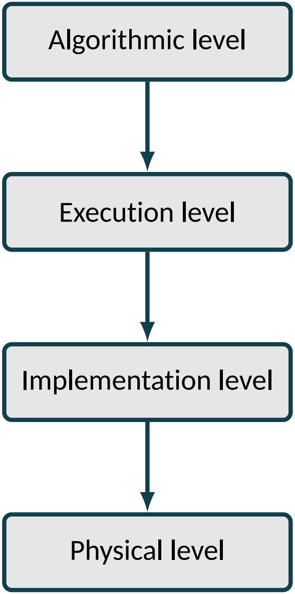
Definition: fault attacks

A fault attack consists in **disturbing the operating conditions** of a device to gain **privileged access** or **knowledge about the secret data** it handles.

Fault injection techniques

- Global
    - Clock glitches,
    - Supply voltage glitches,
    - Underpowering,
    - ...
- Local
    - Electromagnetic,
    - **Optical**,
    - ...

Understanding fault attacks: a timeline



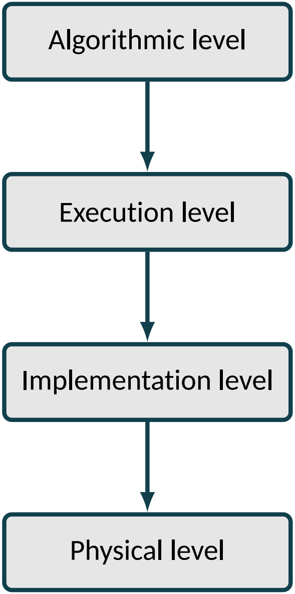
Algorithmic level

Execution level

Implementation level

Physical level

Understanding fault attacks: a timeline



Algorithmic level

Execution level

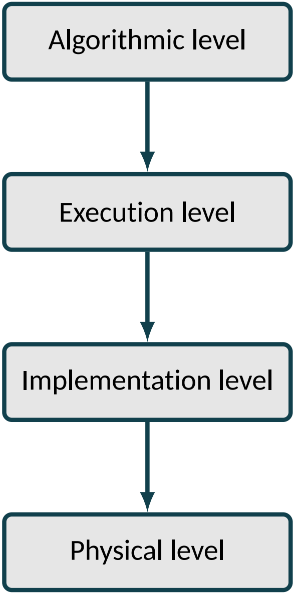
Implementation level

Physical level

8-bit understanding:

- attacks on cryptographic algorithms,
- register corruption and instruction skip,
- timing constraints violation.

Understanding fault attacks: a timeline



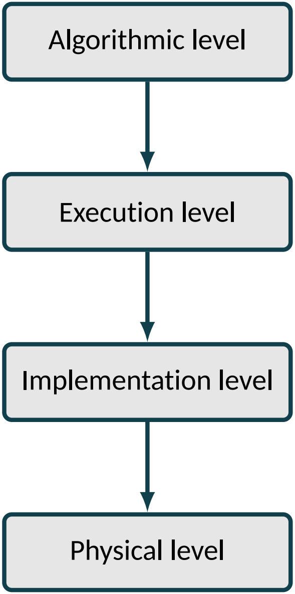
8-bit understanding:

- attacks on cryptographic algorithms,
- register corruption and instruction skip,
- timing constraints violation.

32-bit understanding:

- **Currently**: mostly algorithmic and execution level.

Understanding fault attacks: a timeline



Algorithmic level

Execution level

Implementation level

Physical level

8-bit understanding:

- attacks on cryptographic algorithms,
- register corruption and instruction skip,
- timing constraints violation.

32-bit understanding:

- **Currently**: mostly algorithmic and execution level.

32-bit challenges

- **Bigger**, more **complex** chips,
- **Micro-architecture**: pipeline, pre-fetch...
- Execution timing **variability**.

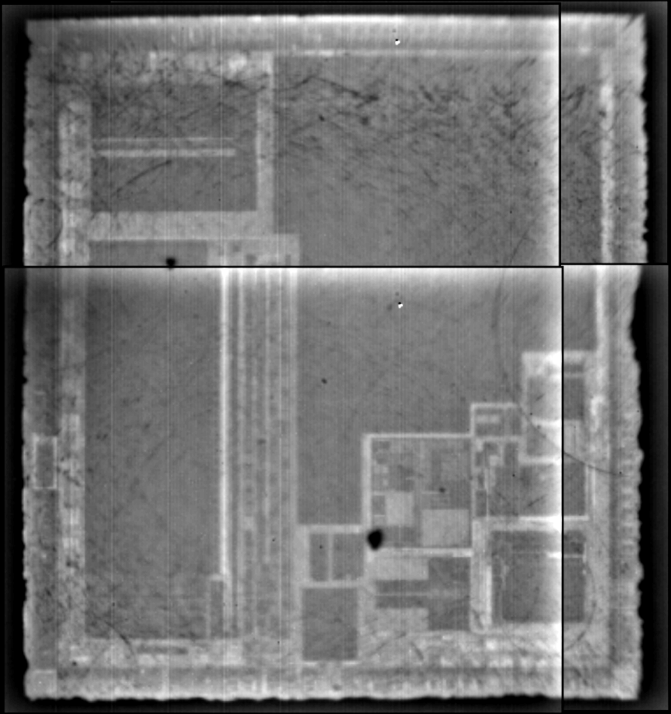
Experimental setup and preparatory work

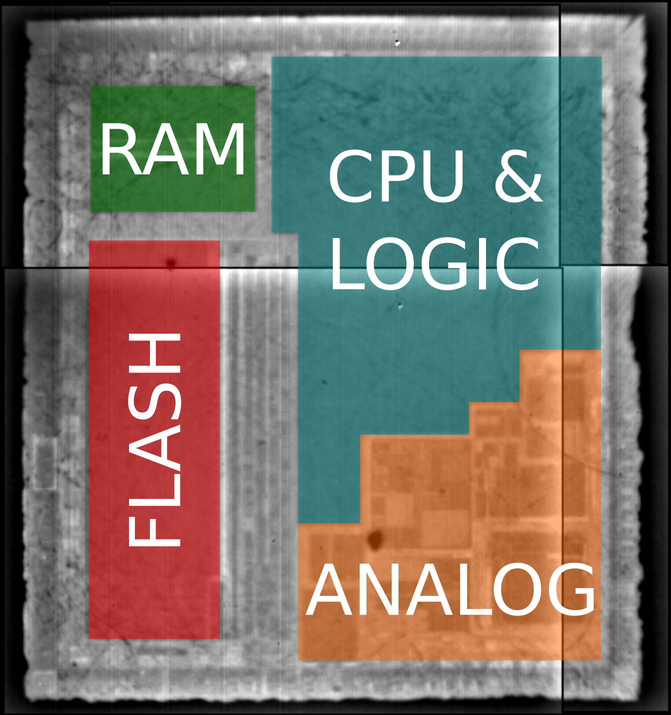
Hardware target

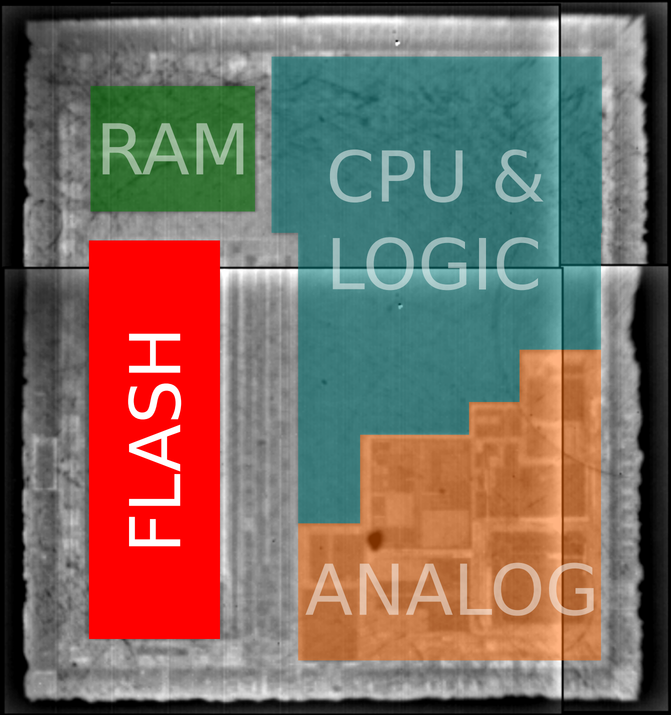
A **32-bit** microcontroller:

- 2.5 x 2.5 mm.
- ARM Cortex-M3 core,
- 90 nm technology node,
- 128 kB of Flash memory,

The C source code is compiled into the **Thumb-2** instruction set.



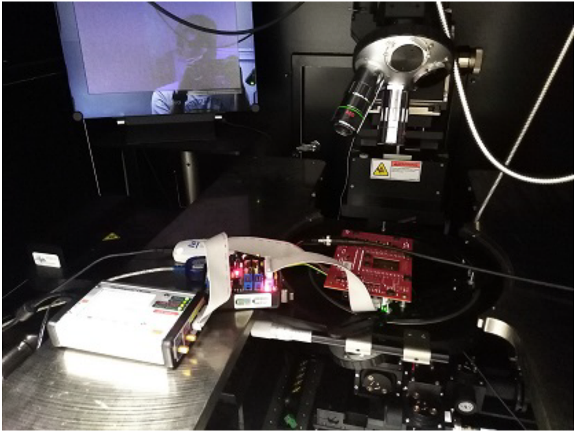

RAM
CPU &
LOGIC
FLASH
ANALOG


RAM
CPU & LOGIC
FLASH
ANALOG

Experimental setup

Laser bench characteristics

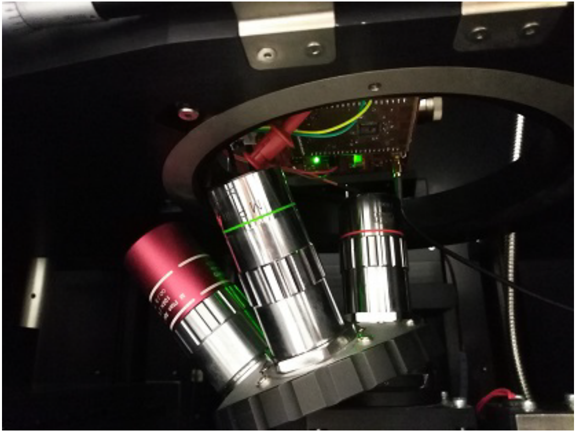
- Infrared (1064 nm) for **back-side** injection,
- >30 ps,
- 0-3 W,
- 3 objective lenses:
  - x5 (20 µm),
  - **x20 (5 µm),**
  - x100 (1 µm).



Experimental setup

Laser bench characteristics

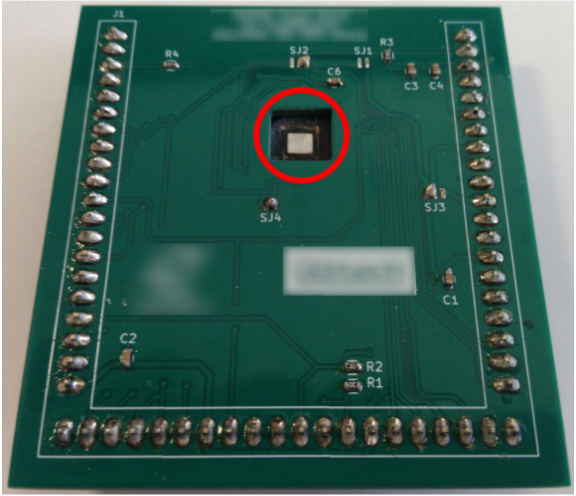
- Infrared (1064 nm) for **back-side** injection,
- >30 ps,
- 0-3 W,
- 3 objective lenses:
  - x5 (20 µm),
  - **x20 (5 µm),**
  - x100 (1 µm).



Experimental setup

Laser bench characteristics

- Infrared (1064 nm) for **back-side** injection,
- >30 ps,
- 0-3 W,
- 3 objective lenses:
  - x5 (20 µm),
  - **x20 (5 µm),**
  - x100 (1 µm).



Perparatory work

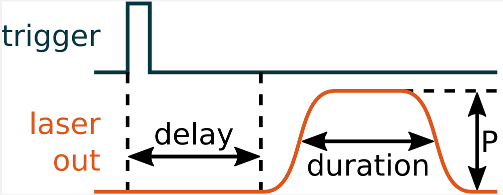
Preparatory work (4-5 months)

- ✓ Design of a **custom** ChipWhisperer target board:
  - ✓ **Front-side** access,
  - ✓ **Back-side** access.
- ✓ Target **preparation**: **decapsulate** the chip to see the die,
- ✓ **Mechanical setup** on the laser injection bench,
- ✓ Faults **mapping**:
  - ✓ x-position,
  - ✓ y-position,
  - ✓ power,
  - ✓ duration,
  - ✓ delay,
  - ✓ **type** of fault: instruction skip, bit-set, bit-reset, bit-flip...

Perparatory work

Preparatory work (4-5 months)

- ✓ Design of a **custom** ChipWhisperer target board:
  - ✓ **Front-side** access,
  - ✓ **Back-side** access.
- ✓ Target **preparation**: **decapsulate** the chip to see the die,
- ✓ **Mechanical setup** on the laser injection bench,
- ✓ Faults **mapping**:
  - ✓ x-position,
  - ✓ y-position,
  - ✓ power,
  - ✓ duration,
  - ✓ delay,
  - ✓ **type** of fault: instruction skip, bit-set, bit-reset, bit-flip...



Characterisation results

Characterisation code

```
test_data:
.word 0x00000000
NOP
NOP
NOP
NOP
NOP
NOP
LDR R0, test_data ←
NOP
NOP
NOP
NOP
NOP
NOP
# Reading back R0
```

- Write a **test data** at a specific address in Flash memory,
- **Store** this value in a **known register**,
- **Read back** the register.

Characterisation code

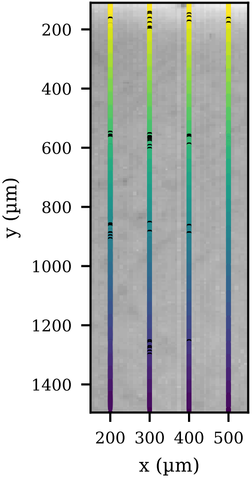
```
test_data:
.word 0x00000000
NOP
NOP
NOP
NOP
NOP
NOP
LDR R0, test_data ←
NOP
NOP
NOP
NOP
NOP
NOP
# Reading back R0
```

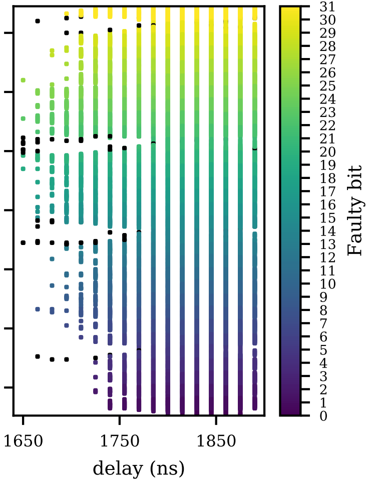
- Write a **test data** at a specific address in Flash memory,
- **Store** this value in a **known register**,
- **Read back** the register.

Choice of test data

- 0x00000000: bit-sets,
- 0xFFFFFFFF: bit-resets,
- 0x55555555
  0xAAAAAAAA: bit-flips.

Faulty bit wrt. x/y position and delay





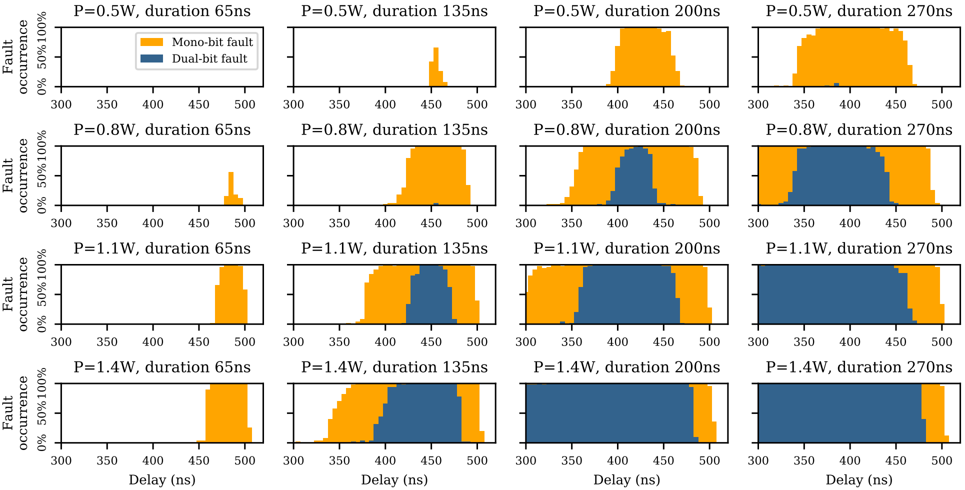
Fault model

Monobit-set on fetched data.

Parameters dependency

Faulty bit depends on **y** position.

Faulty bit wrt. power and duration



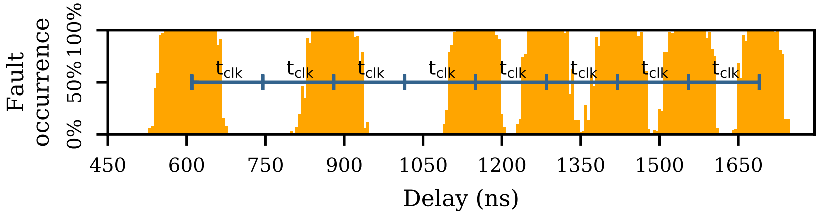
Observation

Increasing the **energy** allows to fault more bits.

Characterisation code 2

```
# Initialising registers
# R0, R1, R4, R5, R6, R8
# and R9 to 0xFFFFFFFF
NOP
NOP
MOVW R0, 0x0000 ←
MOVW R1, 0x0000 ←
MOVW R4, 0x0000 ←
MOVW R5, 0x0000 ←
MOVW R6, 0x0000 ←
MOVW R8, 0x0000 ←
MOVW R9, 0x0000 ←
NOP
NOP
# Reading back the registers
```

Fault periodicity (faulty bit wrt. delay)



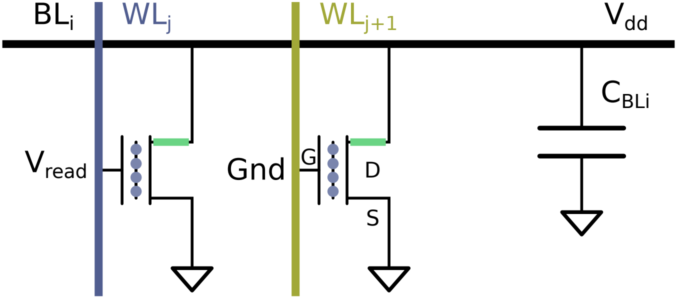
```
MOVW R0, 0x0000 ←
MOVW R1, 0x0000 ←
MOVW R4, 0x0000 ←
MOVW R5, 0x0000 ←
MOVW R6, 0x0000 ←
MOVW R8, 0x0000 ←
MOVW R9, 0x0000 ←
```

Observations

- **Each** instruction can be faulty,
- The occurence **always** reaches **100%**,
- The delay between two optimal injection timings is always a **multiple of the clock period**
- The delay between two optimal injection timings is **not constant**.

Physical explanation

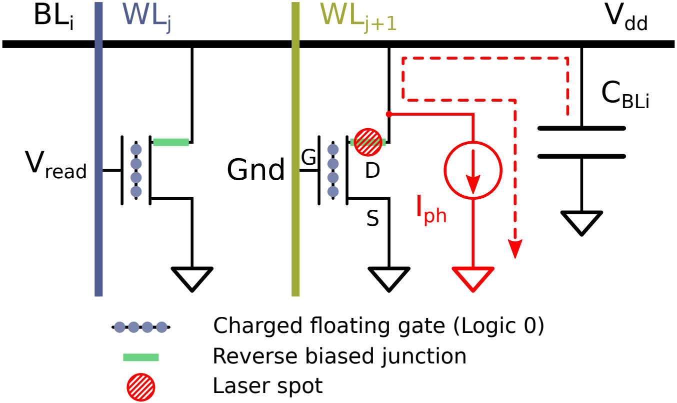
Physical explanation for the y-dependency



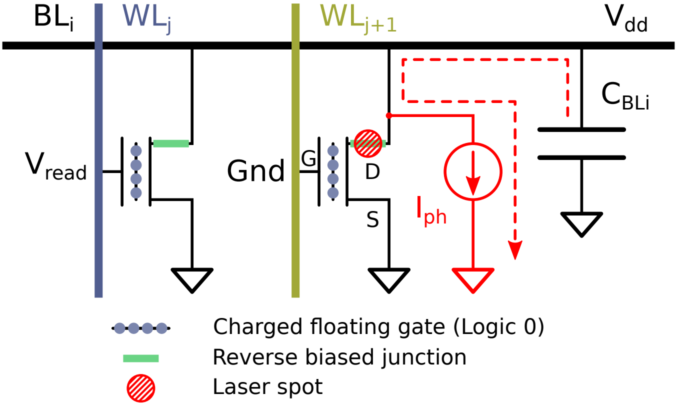
●●●● Charged floating gate (Logic 0)

🟩 Reverse biased junction

Physical explanation for the y-dependency



Physical explanation for the y-dependency



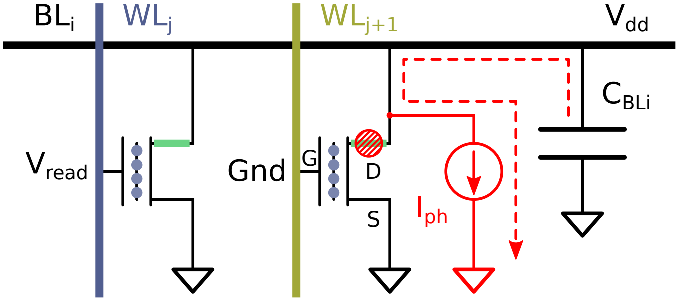
Moving along the x-axis

- Transistors of the same **BL**.
- **Same** faulty bit.

Moving along the y-axis

- Transistors of the same **WL**.
- **Successive** faulty bits.

Physical explanation for the asymmetry



●●●● Charged floating gate (Logic 0)

🟩 Reverse biased junction

▨ Laser spot

Without laser shot

- **with** charges: BL to $\mathbf{V_{dd}}$
- **without** charges: BL to **GND**

With laser shot

- **with** charges: BL to **GND**
- **without** charges: BL to **GND**

Applications

Example of instruction corruption: 32-bit `MOVW`

`MOVW`: store a 16-bit value in the lower half of a 32-bit register.

bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	1 2	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions:

`MOVW`	1	1	1	1	0	i	1	0	0	1	0	0	imm4				0	imm3			Rd				imm8							
`MOVW, R0, 0`	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Data corruption:

`MOVW, R0, 4`	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0

Register corruption:

`MOVW, R1, 0`	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0

Opcode corruption:

`MOVT, R0, 0`	1	1	1	1	0	0	1	0	**1**	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PIN verification algorithm

Constant-time implementation with **hardened booleans**:
No **simple** side-channel attack and TRUE=0x5555, FALSE=0xAAAA.

```
trials = 3
ref_PIN[4] = {1, 2, 3, 4}
procedure VerifyPIN(user_PIN[4])
    authenticated = FALSE
    diff = FALSE
    dummy = TRUE
    if trials > 0 then
        for i ← 0 to 3 do
            if user_PIN[i] != ref_PIN[i] then
                diff = TRUE
            else
                dummy = FALSE
            end if
        end for
        if diff == TRUE then
            trials = trials - 1
        else
            authenticated = TRUE
        end if
    end if
    return authenticated
end procedure
```

PIN verification algorithm

Constant-time implementation with **hardened booleans**:
No **simple** side-channel attack and `TRUE=0x5555`, `FALSE=0xAAAA`.

```
trials = 3
ref_PIN[4] = {1, 2, 3, 4}
procedure VerifyPIN(user_PIN[4])
    authenticated = FALSE
    diff = FALSE
    dummy = TRUE
    if trials > 0 then
        for i ← 0 to 3 do
            if user_PIN[i] != ref_PIN[i] then
                diff = TRUE
            else
                dummy = FALSE
            end if
        end for
        if diff == TRUE then
            trials = trials - 1
        else
            authenticated = TRUE
        end if
    end if
    return authenticated
end procedure
```

```
if (trials > 0)
{
  ...
}
```

```
CMP R3, 0
BLE address
```

Fault injection on the CMP instruction

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

Generic `CMP`	0	0	1	0	1	Rd			imm8							
`CMP R3, 0`	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0

Fault injection on the CMP instruction

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

Generic `CMP`	0	0	1	0	1	Rd			imm8							
`CMP R3, 0`	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0

Perform a bit-set on the **10th** bit of the instruction: `R3` → `R7`.

By design, `R7` stores the *frame-pointer*, **always positive**.

Fault injection on the `CMP` instruction

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

Generic `CMP`	0	0	1	0	1	Rd			imm8							
`CMP R3, 0`	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0

Perform a bit-set on the **10th** bit of the instruction: `R3` → `R7`.

By design, `R7` stores the *frame-pointer*, **always positive**.

Register corruption

						↓										
`CMP R7, 0`	0	0	1	0	1	**1**	1	1	0	0	0	0	0	0	0	0

Fault injection on the `CMP` instruction

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

Generic `CMP`	0	0	1	0	1	Rd			imm8							
`CMP R3, 0`	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0

Perform a bit-set on the **10th** bit of the instruction: `R3` → `R7`.
By design, `R7` stores the *frame-pointer*, **always positive**.

Register corruption

						↓										
`CMP R7, 0`	0	0	1	0	1	**1**	1	1	0	0	0	0	0	0	0	0

Outcome

trials is **never compared** → **unlimited** number of trials.

AES-128 encryption: last `AddRoundKey`

```
procedure AddRoundKey
    for i ← 0 to 3 do
        for j ← 0 to 3 do
            $S_{i,j} = S_{i,j} \oplus K^{10}_{i,j}$
        end for
    end for
end procedure
```

AES-128 encryption: last AddRoundKey

```
procedure AddRoundKey
    for i ← 0 to 3 do
        for j ← 0 to 3 do
            $S_{i,j} = S_{i,j} \oplus K^{10}_{i,j}$
        end for
    end for
end procedure
```

```
for (int i=0; i<4; i++)
{
  for (int j=0; j<4; j++)
  {
    ...
  }
}
```

```
MOV R0, 0
addr_i:
MOV R1, 0
addr_j:
...
ADD R1, 1
CMP R1, 3
BLE addr_j
ADD R0, 1
CMP R0, 3
BLE addr_i
```

Fault injection on the `ADD` instruction

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

<table>
<tr><td>Generic <code>ADD</code></td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td colspan="3">Rd</td><td colspan="8">imm8</td></tr>
<tr><td><code>ADD R0, 1</code></td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
</table>

Fault injection on the ADD instruction

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

Generic `ADD`	0	0	1	1	0	Rd			imm8							
`ADD R0, 1`	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1

Perform a bit-set on the **2nd** bit of the instruction.

Add **5** instead of **1** to the **loop variable**.

Fault injection on the ADD instruction

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

Generic `ADD`	0	0	1	1	0	Rd			imm8							
`ADD R0, 1`	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1

Perform a bit-set on the **2$^{\text{nd}}$** bit of the instruction.
Add **5** instead of **1** to the **loop variable**.

Data corruption

`ADD R0, 5`	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1

Fault injection on the ADD instruction

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Reference instructions

Generic `ADD`	0	0	1	1	0	Rd			imm8							
`ADD R0, 1`	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1

Perform a bit-set on the **2nd** bit of the instruction.

Add **5** instead of **1** to the **loop variable**.

Data corruption

`ADD R0, 5`	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1

Outcome

For loop exit after **one** execution only.

Fault on the *for* loops

Faulty ciphertext byte: $\tilde{C}_{x,y} = C_{x,y} \oplus K^{10}_{x,y}$

Fault on the **inner** for loop on its **first** execution.

$C_{0,0}$	$C_{1,0}$	$C_{2,0}$	$C_{3,0}$
$\tilde{C}_{0,1}$	$C_{1,1}$	$C_{2,1}$	$C_{3,1}$
$\tilde{C}_{0,2}$	$C_{1,2}$	$C_{2,2}$	$C_{3,2}$
$\tilde{C}_{0,3}$	$C_{1,3}$	$C_{2,3}$	$C_{3,3}$

Fault on the *for* loops

Faulty ciphertext byte: $\tilde{C}_{x,y} = C_{x,y} \oplus K^{10}_{x,y}$

Fault on the **inner** for loop on its **first** execution.

$C_{0,0}$	$C_{1,0}$	$C_{2,0}$	$C_{3,0}$
$\tilde{C}_{0,1}$	$C_{1,1}$	$C_{2,1}$	$C_{3,1}$
$\tilde{C}_{0,2}$	$C_{1,2}$	$C_{2,2}$	$C_{3,2}$
$\tilde{C}_{0,3}$	$C_{1,3}$	$C_{2,3}$	$C_{3,3}$

Fault on the **outer** for loop.

$C_{0,0}$	$\tilde{C}_{1,0}$	$\tilde{C}_{2,0}$	$\tilde{C}_{3,0}$
$C_{0,1}$	$\tilde{C}_{1,1}$	$\tilde{C}_{2,1}$	$\tilde{C}_{3,1}$
$C_{0,2}$	$\tilde{C}_{1,2}$	$\tilde{C}_{2,2}$	$\tilde{C}_{3,2}$
$C_{0,3}$	$\tilde{C}_{1,3}$	$\tilde{C}_{2,3}$	$\tilde{C}_{3,3}$

10^{th} round-key recovery

Faulty ciphertext byte: $\tilde{C}_{x,y} = C_{x,y} \oplus K^{10}_{x,y}$

$C_{0,0}$	$C_{1,0}$	$C_{2,0}$	$C_{3,0}$
$\tilde{C}_{0,1}$	$C_{1,1}$	$C_{2,1}$	$C_{3,1}$
$\tilde{C}_{0,2}$	$C_{1,2}$	$C_{2,2}$	$C_{3,2}$
$\tilde{C}_{0,3}$	$C_{1,3}$	$C_{2,3}$	$C_{3,3}$

$\oplus$

$C_{0,0}$	$\tilde{C}_{1,0}$	$\tilde{C}_{2,0}$	$\tilde{C}_{3,0}$
$C_{0,1}$	$\tilde{C}_{1,1}$	$\tilde{C}_{2,1}$	$\tilde{C}_{3,1}$
$C_{0,2}$	$\tilde{C}_{1,2}$	$\tilde{C}_{2,2}$	$\tilde{C}_{3,2}$
$C_{0,3}$	$\tilde{C}_{1,3}$	$\tilde{C}_{2,3}$	$\tilde{C}_{3,3}$

=

10^{th} round-key recovery

Faulty ciphertext byte: $\tilde{C}_{x,y} = C_{x,y} \oplus K^{10}_{x,y}$

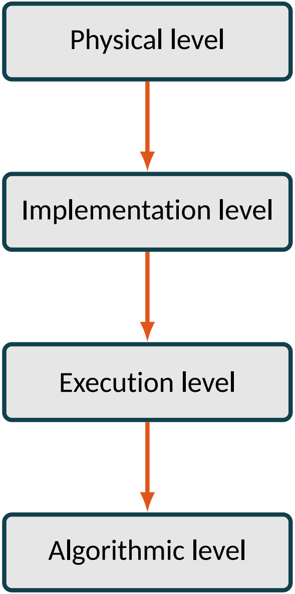
$$\begin{array}{|c|c|c|c|}\hline C_{0,0} & C_{1,0} & C_{2,0} & C_{3,0} \\ \hline \tilde{C}_{0,1} & C_{1,1} & C_{2,1} & C_{3,1} \\ \hline \tilde{C}_{0,2} & C_{1,2} & C_{2,2} & C_{3,2} \\ \hline \tilde{C}_{0,3} & C_{1,3} & C_{2,3} & C_{3,3} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|}\hline C_{0,0} & \tilde{C}_{1,0} & \tilde{C}_{2,0} & \tilde{C}_{3,0} \\ \hline C_{0,1} & \tilde{C}_{1,1} & \tilde{C}_{2,1} & \tilde{C}_{3,1} \\ \hline C_{0,2} & \tilde{C}_{1,2} & \tilde{C}_{2,2} & \tilde{C}_{3,2} \\ \hline C_{0,3} & \tilde{C}_{1,3} & \tilde{C}_{2,3} & \tilde{C}_{3,3} \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 0 & K^{10}_{1,0} & K^{10}_{2,0} & K^{10}_{3,0} \\ \hline K^{10}_{0,1} & K^{10}_{1,1} & K^{10}_{2,1} & K^{10}_{3,1} \\ \hline K^{10}_{0,2} & K^{10}_{1,2} & K^{10}_{2,2} & K^{10}_{3,2} \\ \hline K^{10}_{0,3} & K^{10}_{1,3} & K^{10}_{2,3} & K^{10}_{3,3} \\ \hline \end{array}$$

What then?

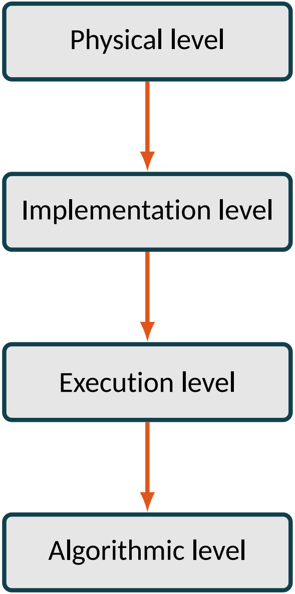
Only **one byte** of the 10^{th} round-key, must be **brute-forced**.

Conclusion

Summary

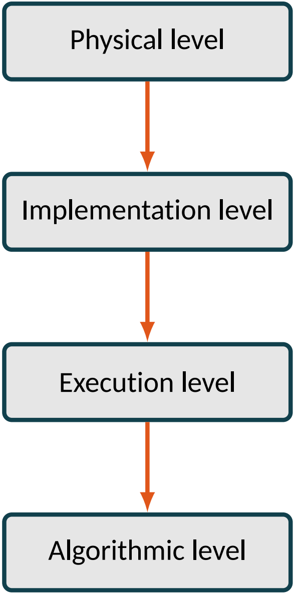

Physical level
Implementation level
Execution level
Algorithmic level

Summary



Force storage transistors to **conduct** in Flash memory.

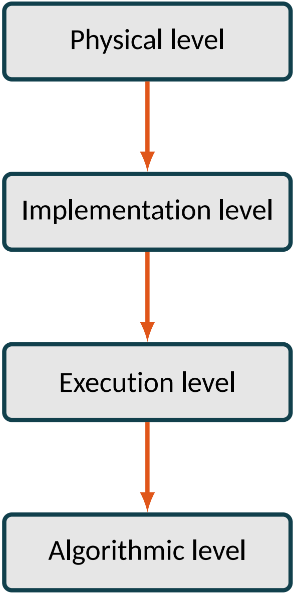
Summary



Force storage transistors to **conduct** in Flash memory.

Perform a **bit-set** on a **chosen single** bit of the instruction.

Summary

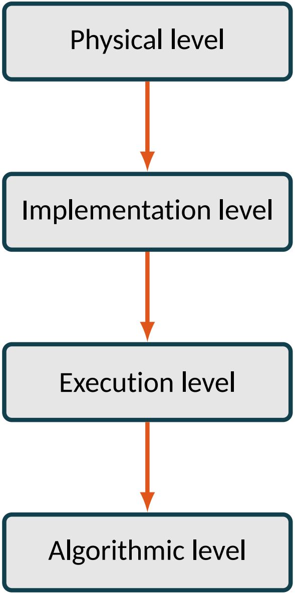


Force storage transistors to **conduct** in Flash memory.

Perform a **bit-set** on a **chosen single** bit of the instruction.

Always take the first *if* branch. **Prematurely exit** the *for* loops.

Summary



Physical level

Implementation level

Execution level

Algorithmic level

Force storage transistors to **conduct** in Flash memory.

Perform a **bit-set** on a **chosen single** bit of the instruction.

Always take the first *if* branch.
Prematurely exit the *for* loops.

Unlimited trials on the VerifyPIN.
AES last `AddRoundKey` alteration.

Perspectives

Possibilities

- **Bit-set** on Flash data,
- Security level **lowering**.

Limitations

- **Contiguous** bits only,
- **Control-flow** alteration mostly.

Perspectives

Possibilities

- **Bit-set** on Flash data,
- Security level **lowering**.

Limitations

- **Contiguous** bits only,
- **Control-flow** alteration mostly.

Perspectives:

- Try on **other application** codes,
- Try on **protected** codes,
- Try on **other microcontrollers**,
- **Multispot** laser:
  - More **possibilities** of corruption,
  - Disable **error-detection/correction** capabilities.
- Develop **countermeasures**

Perspectives

Possibilities

- **Bit-set** on Flash data,
- Security level **lowering**.

Limitations

- **Contiguous** bits only,
- **Control-flow** alteration mostly.

Perspectives:

- Try on **other application** codes,
- Try on **protected** codes,
- Try on **other microcontrollers**,
- **Multispot** laser:
  - More **possibilities** of corruption,
  - Disable **error-detection/correction** capabilities.
- Develop **countermeasures**

— Questions? —