



Evaluation by laser fault injection and photoemission of fault models on a RISC-V instructions set

Mael Coatanhay (IETR), **Edna Rocio Ferrucho Alvarez** (IETR), **Ludovic Claudepierre** (IETR), **Thomas Rubiano** (Inria), **Damien Hardy** (IRISA), **Laurent Pichon** (IETR), **Laurent Le Brizoual** (IETR)

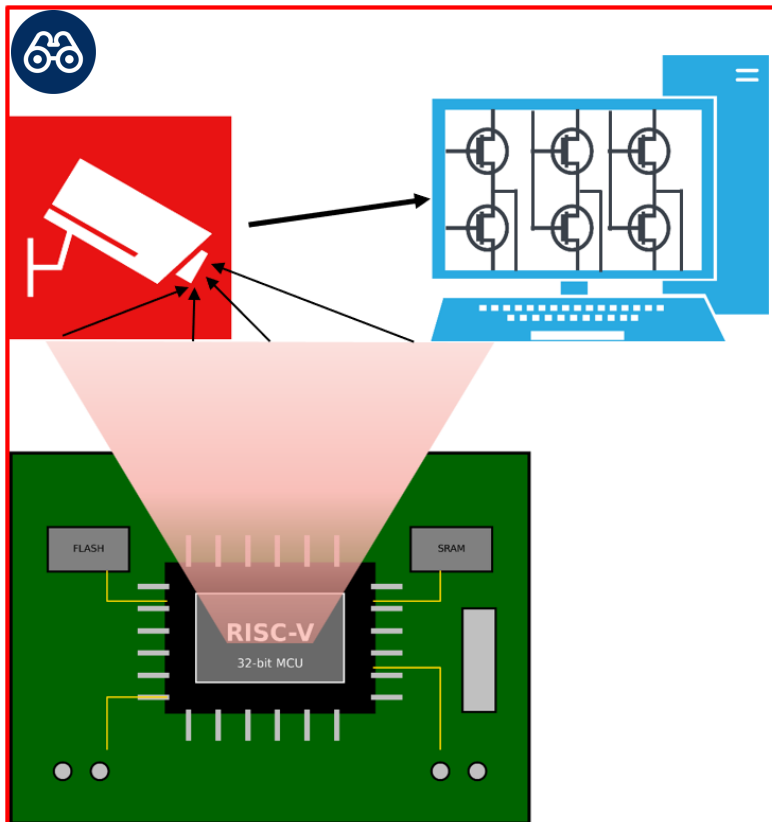
Objectives:

Main objective: Evaluating **RISC-V robustness** using **Photoemission** analysis and **Laser Fault Injection (LFI)**

Objectives:

Main objective: Evaluating **RISC-V robustness** using **Photoemission** analysis and **Laser Fault Injection (LFI)**

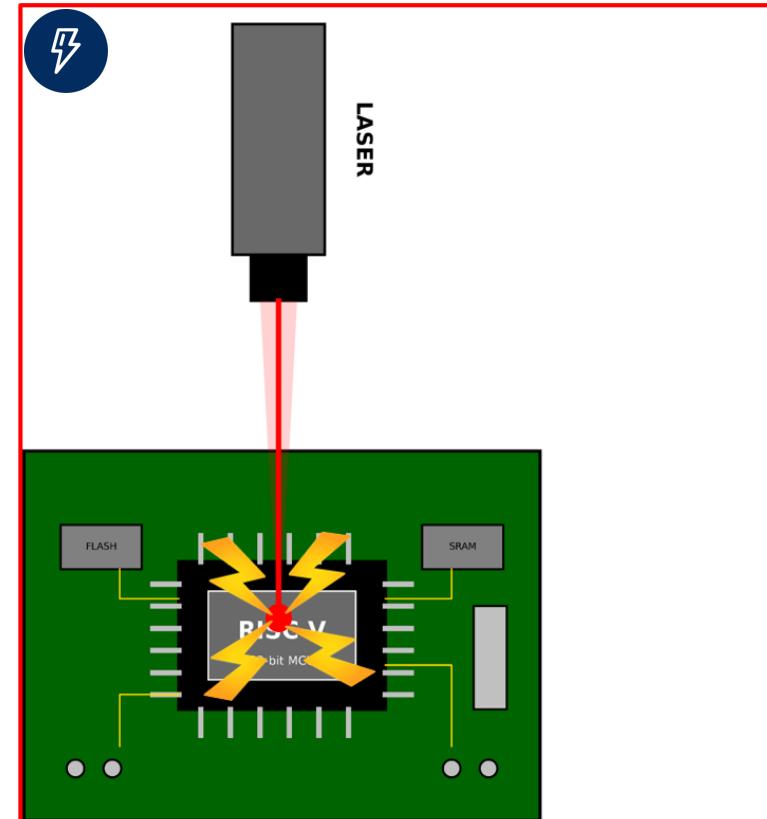
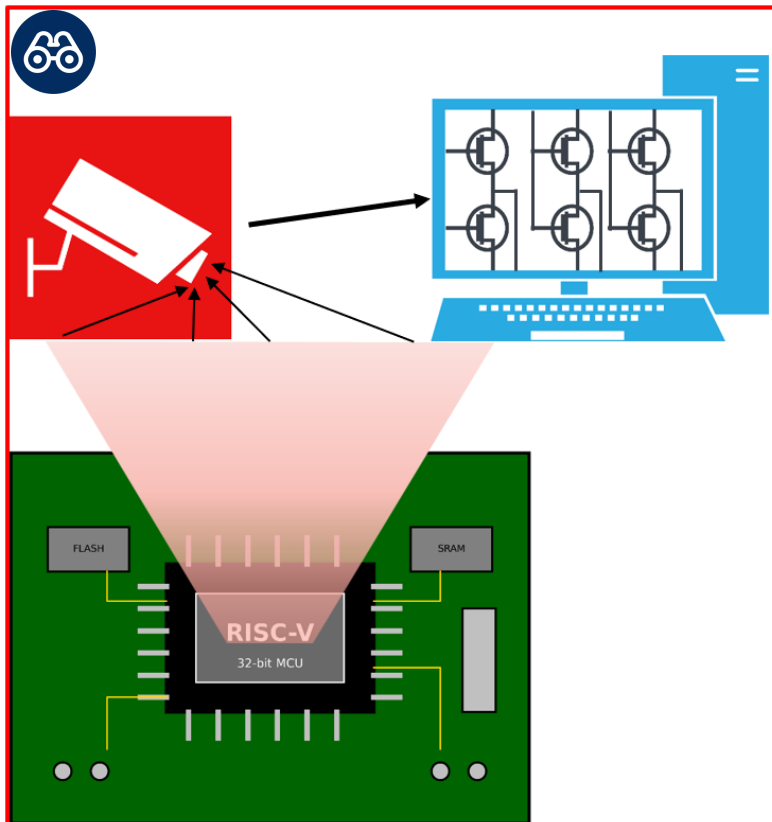
- Reverse engineering by photon emission microscopy (PEM)



Objectives:

Main objective: Evaluating **RISC-V robustness** using **Photoemission** analysis and **Laser Fault Injection (LFI)**

- Reverse engineering by photon emission microscopy (PEM)
- Laser Injection and Fault Modeling
- Evaluation of their exploitability in a security context



- **Instruction corruption:** The injection of physical defects can lead to single-bit or multi-bit instruction corruption.[Col19]
- **Persistent fault:** Some physical defects can persist across multiple executions and be exploited.[Zha20]
- **PEM & SRAM reverse engineering:** PEM enables SRAM structure identification and reverse engineering.[Lim24] Laser probing has also been investigated for SRAM reverse engineering.[Che18]
- **PEM-guided LFI:** PEM can identify points of interest to guide LFI.[Per24][Cas26]

Positioning: The goal is to **map logical addresses** to the **physical locations** of the bits in SRAM. This allows us to **study asynchronous memory corruptions** that is, those that can be induced independently of the processor's access to the targeted data. Eventually, the method should also be applicable to the cache.

[Lim24] R. S. Lima and al, DOI: 10.1109/PAINE62042.2024.10792702, 2024.

[Per24] H. Perrin and al, DOI: 10.1109/HOST64725.2025.11050055, 2024.

[Che18] S. Chef and al, DOI: 10.1109/IPFA.2018.8452604, 2018.

[Col19] B. Colombier and al, DOI: 10.1109/HST.2019.8741030, 2019.

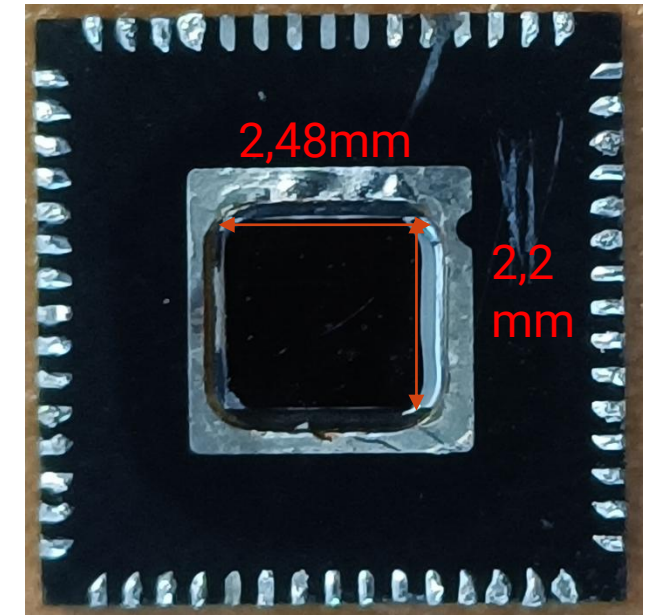
[Zha20] F. Zhang and al, DOI: <https://doi.org/10.13154/tches.v2020.i2.172-195>, 2020.

[Cas26] S. Casavecchia and al, DOI: 10.1109/HOST68814.2026.11604707, 2026.

- Processor: mixed architecture, combining ARM and **RISC-V Hazard3** cores
- Clock speed: **150 MHz** max
- SRAM: **520 KB**
- Flash memory: 16 MB external
- Cache: 16 KB
- Peripherals: 26 GPIO(UART, SPI, I2C, PWM, ADC, ...)
- Technology: **40 nm**
- Programming: MicroPython and C/C++.

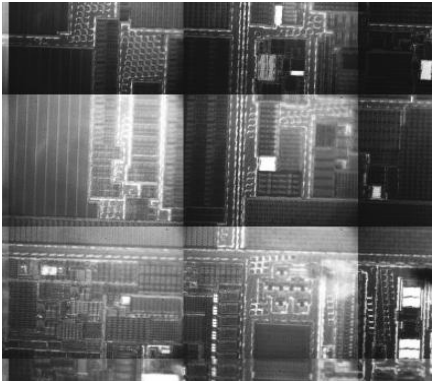
Hazard3 core configuration:

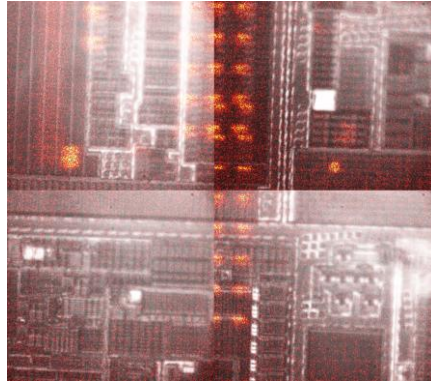
- 3-stage in-order pipeline: F / X / M
- RV32I
- Extension:
M,A,C,Zicsr,Zifencei,Zba,Zbb,Zbs,Zbkb,Zcb,Zcmp
- No branch prediction
- 32 registers
- Mode: Machine (M) et User (U)
- Physical Memory Protection (PMP)



1

Visual Analysis: Structure Identification



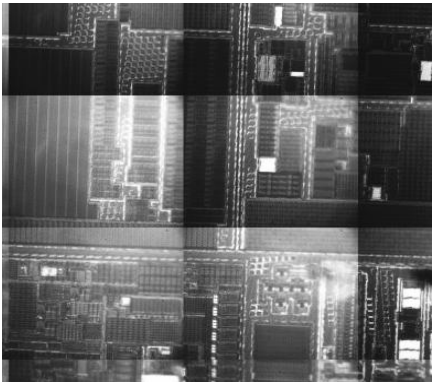


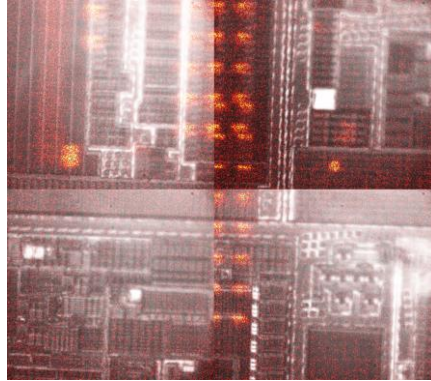
Photon Emission Microscopy (PEM): 
Reverse engineering of SRAM

1

2

Visual Analysis: 
Structure Identification





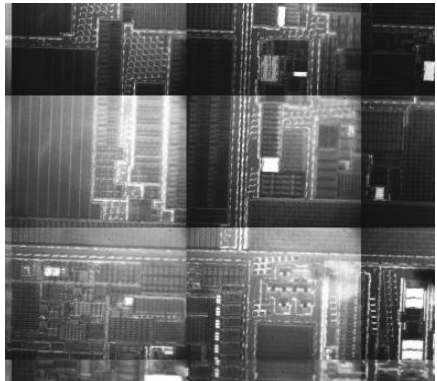
Photon Emission Microscopy (PEM): 
Reverse engineering of SRAM

1

2

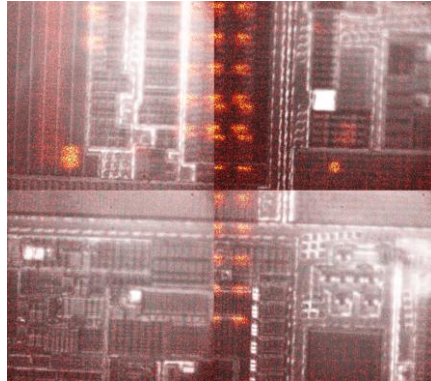
3

Visual Analysis: 
Structure Identification



LFI with ×20 and ×50 objectives: 
Fault Injection Reverse Engineering (FIRE) and classification of faults based on possible fault models .

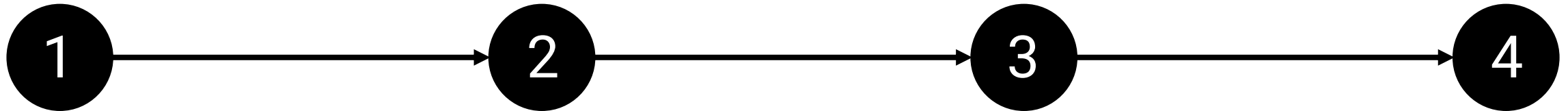




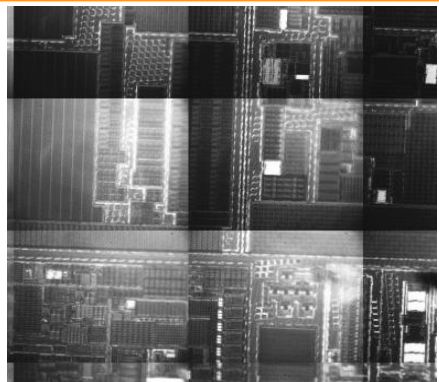
Photon Emission Microscopy (PEM): 
Reverse engineering of SRAM

```
fibonacci(int):
    addi    sp,sp,-32
    sw      ra,20(sp)
    sw      s0,24(sp)
    sw      s1,28(sp)
    addi    s0,sp,32
    sw      a0,-20(s0)
    lw      a4,-20(s0)
    li      a5,1
    bgt     a4,a5,.L12
    lw      a5,-20(s0)
    j       .L13
.L12:
    lw      a5,-20(s0)
    addi    a5,a5,-1
    mv      a0,a5
    call    fibonacci(int)
    mv      s1,a0
    lw      a5,-20(s0)
    addi    a5,a5,-2
    mv      a0,a5
    call    fibonacci(int)
    mv      a5,a0
    add     a5,s1,a5
.L13:
    mv      a0,a5
    lw      ra,20(sp)
    lw      s0,24(sp)
    lw      s1,28(sp)
    addi    sp,sp,32
    jr      ra
```

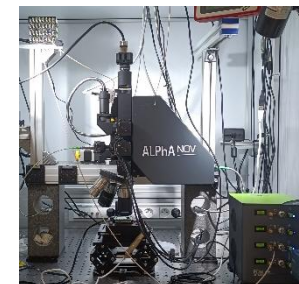
Implementation Attacks: 
Definition of attack scenarios



Visual Analysis: 
Structure Identification



LFI with ×20 and ×50 objectives: 
Fault Injection Reverse Engineering (FIRE) and classification of faults based on possible fault models .

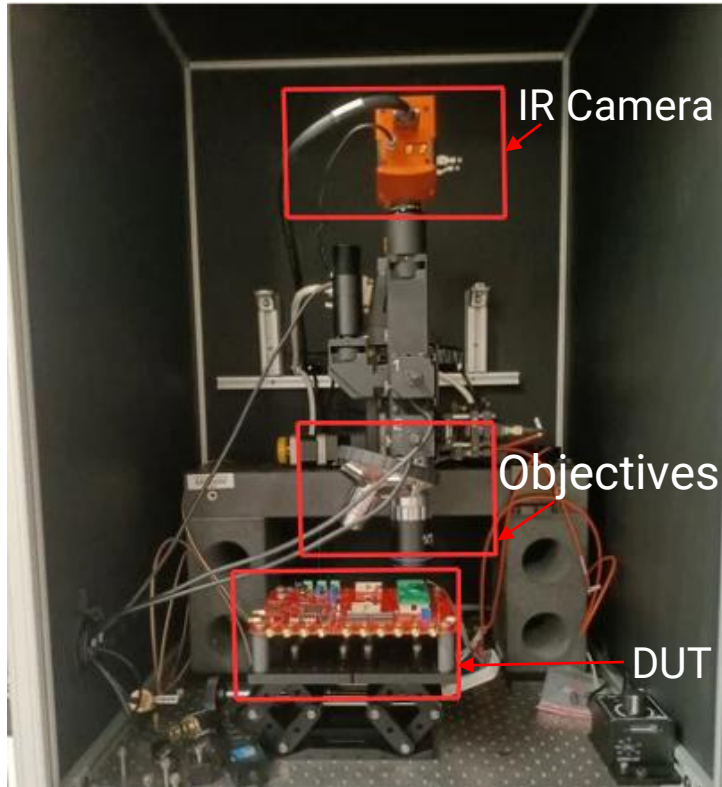


PEM – Reverse Engineering and Characterization of SRAM

- Step 1: Visual Analysis
- Step 2: Photon Emission Microscopy (PEM)

Photoemission Bench and SRAM localization

Photoemission setup:

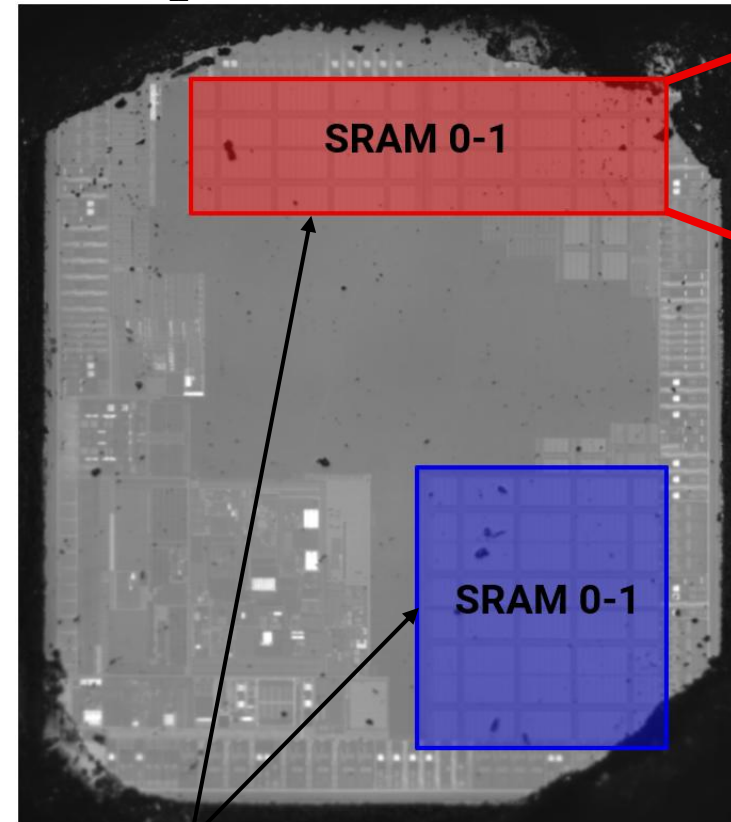


Photoemission Bench and SRAM localization

Photoemission setup:

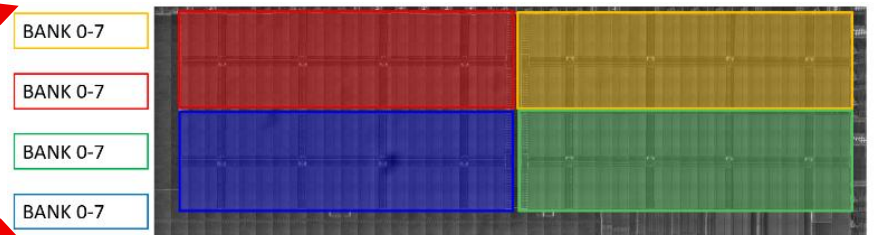


IR image of the circuit:

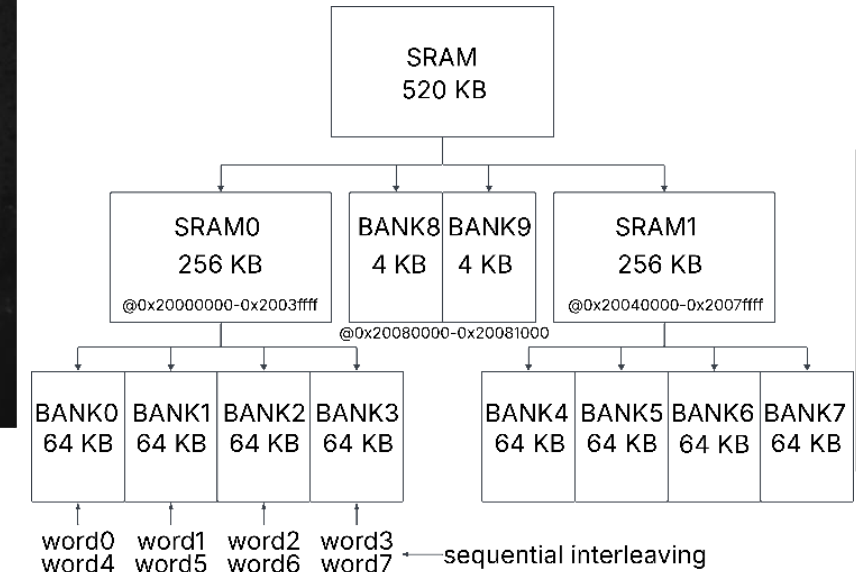


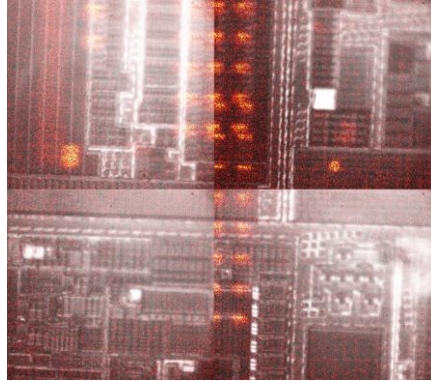
SRAM 0 or SRAM 1?

According to the 38th Chaos Communication Congress 2024:[Cul24]



SRAM Organization Based on the Documentation:



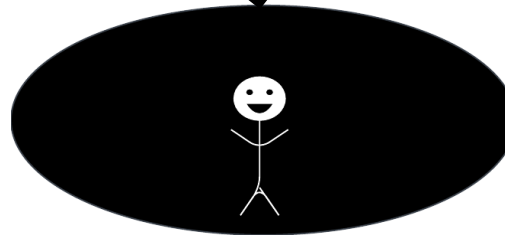
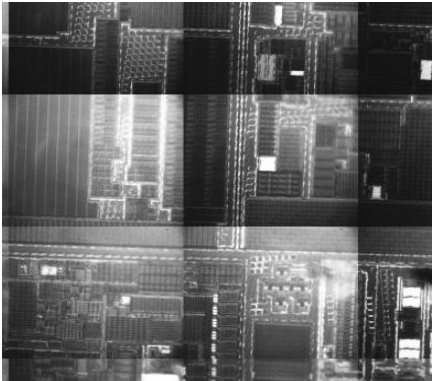


Photon Emission Microscopy (PEM): 
Reverse engineering of SRAM

1

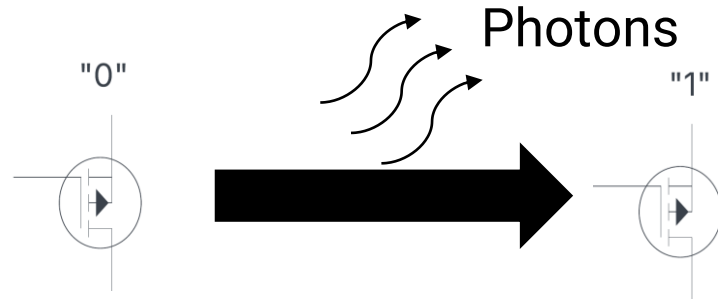
2

Visual Analysis: 
Structure Identification

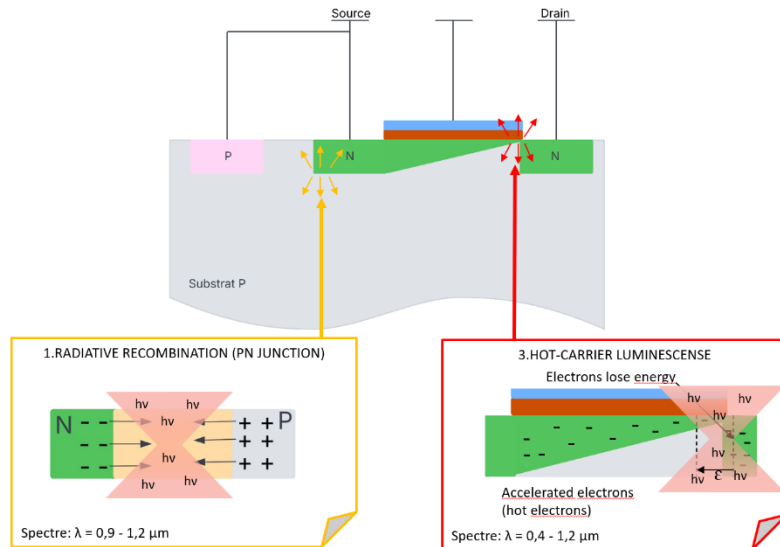


Principle

Principle of Photoemission:

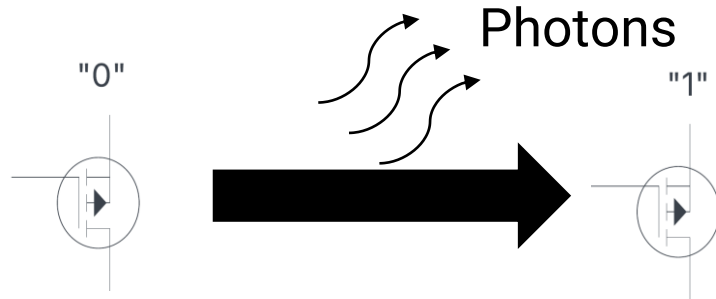


Physics of Photoemission:

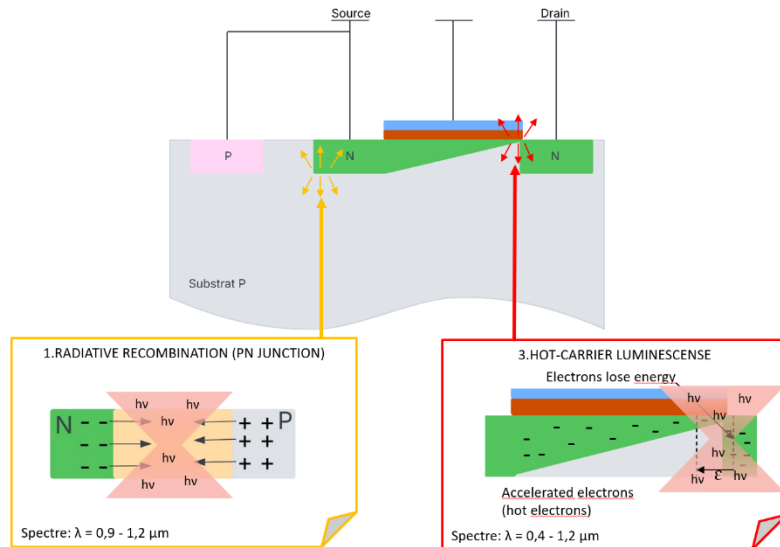


Principle

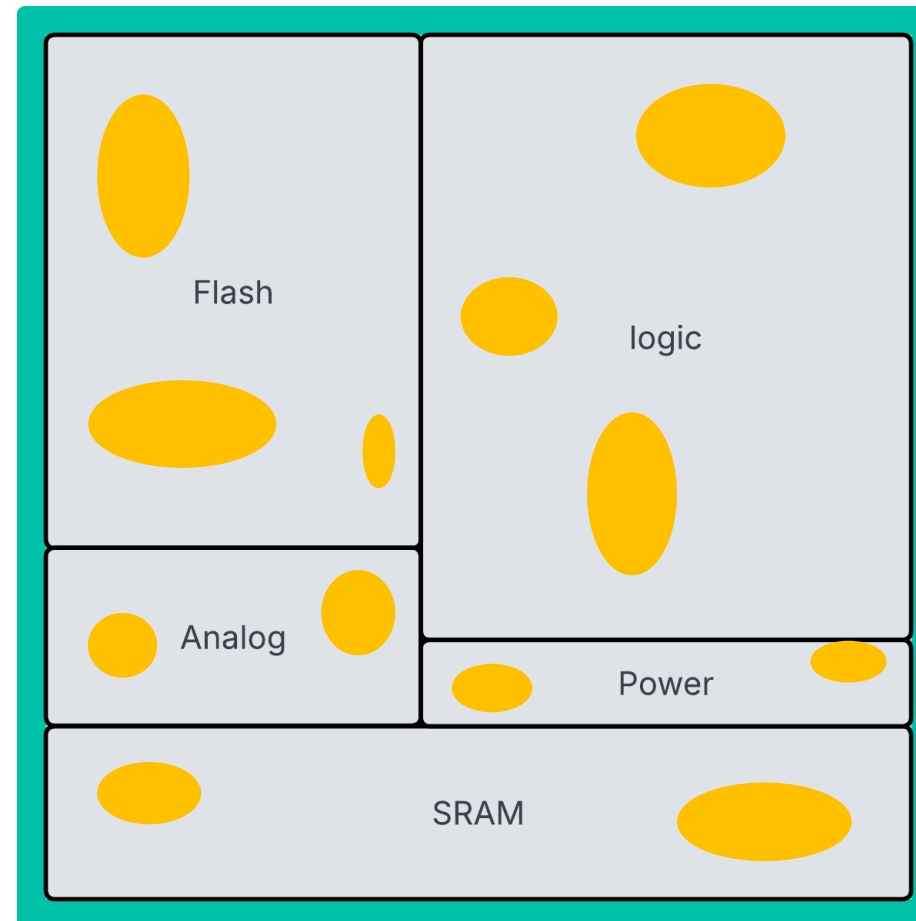
Principle of Photoemission:



Physics of Photoemission:



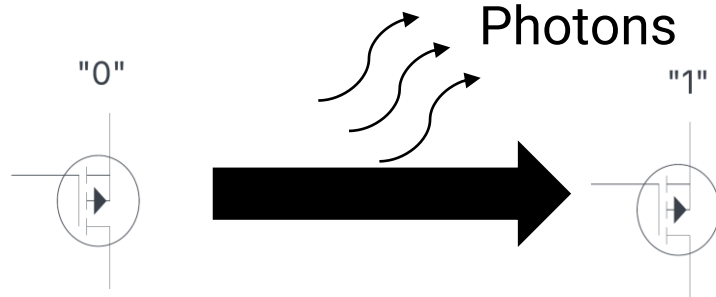
Using photoemission:



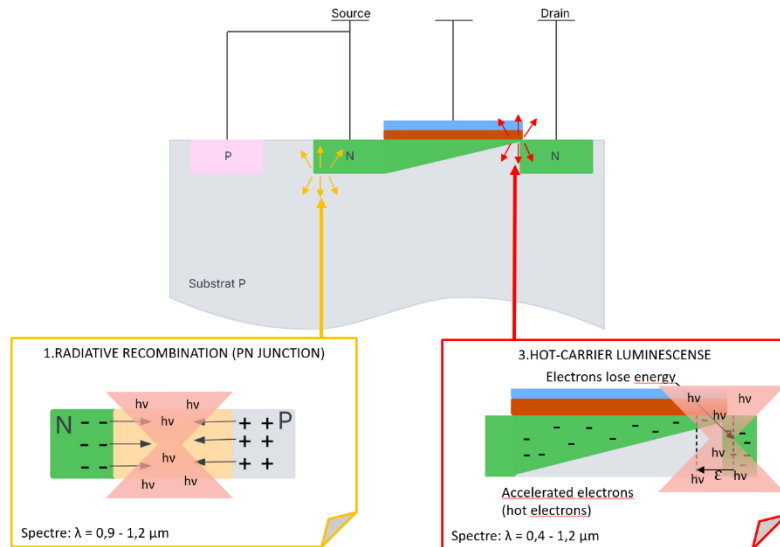
Photoemission zone

Principle

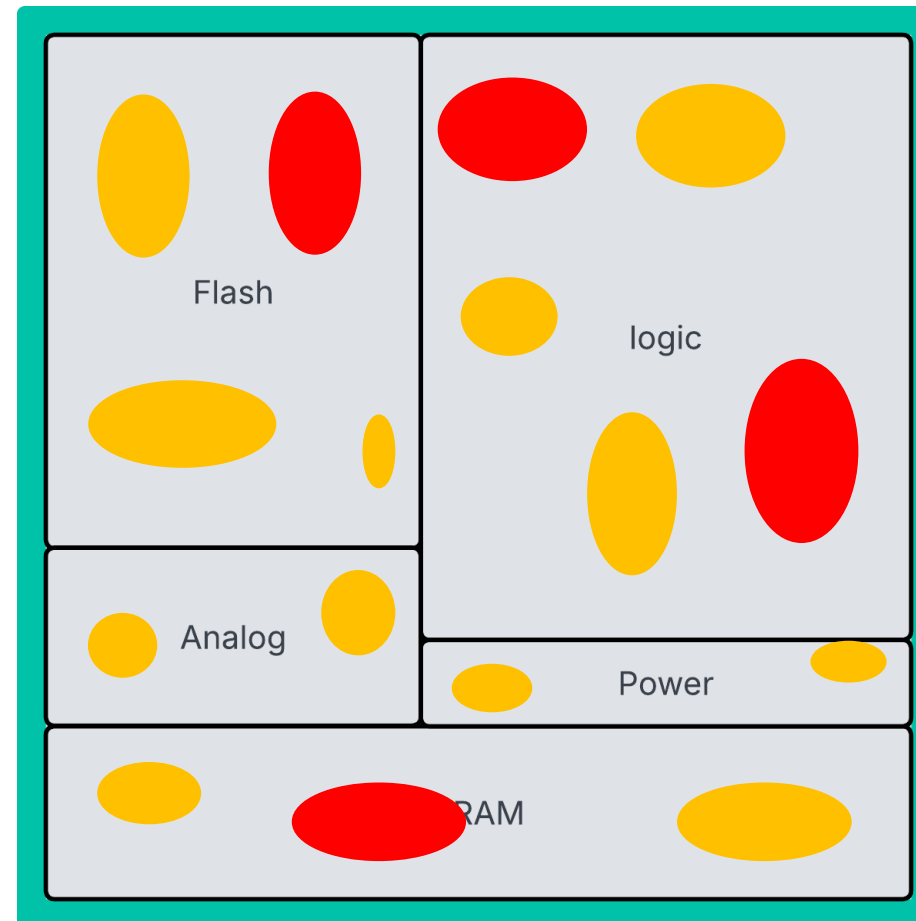
Principle of Photoemission:



Physics of Photoemission:



Using photoemission:



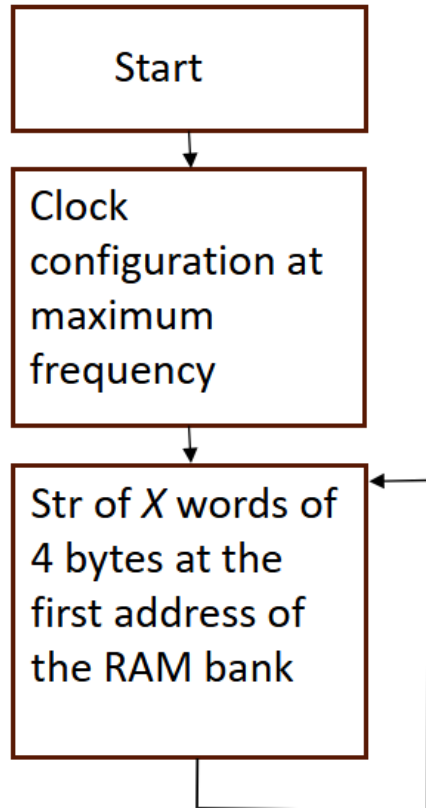
Photoemission zone

Zone of interest for laser fault injection

Reduced exploration of laser positioning on the component

Photoemission methodology

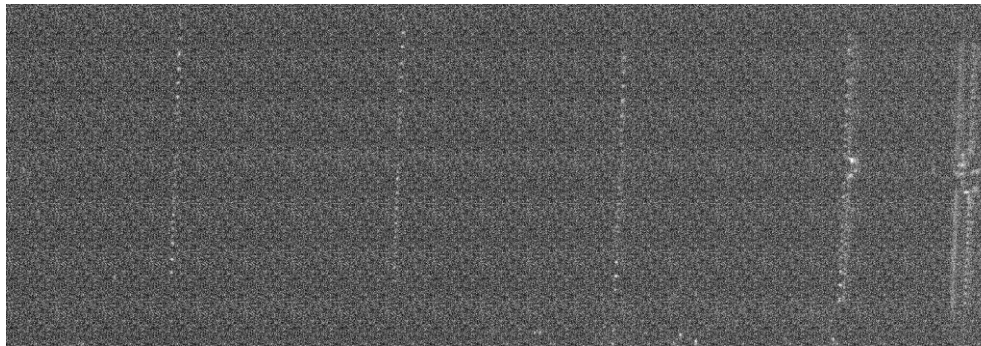
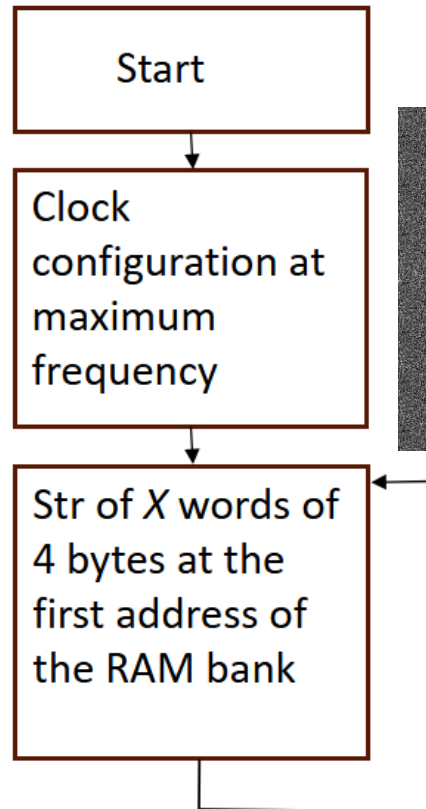
Experimental process:



Photoemission methodology

Experimental process:

Photoemission results for RAM writes:



Signal+power+noise

Photoemission parameters:

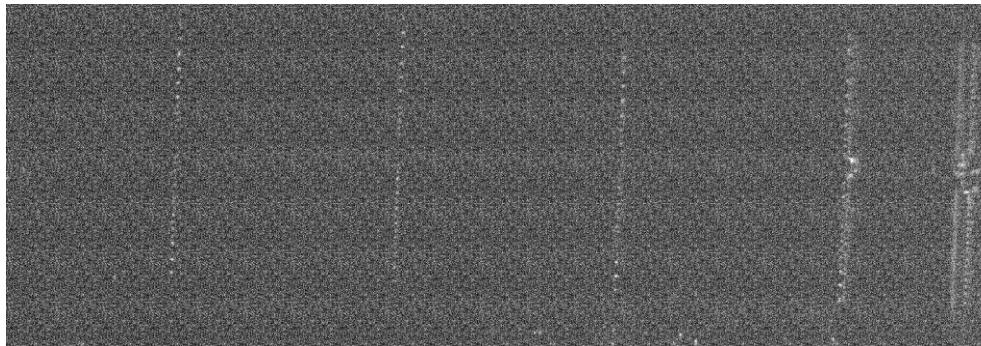
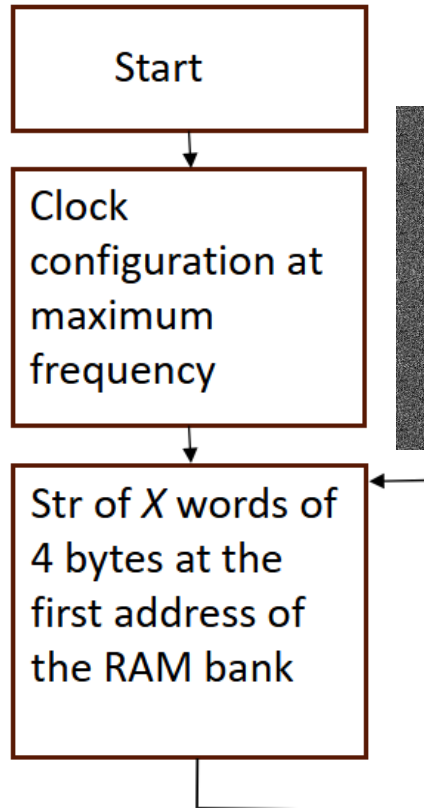
Objective x50 Exposure time: 2000 ms
Data: "0xAAAAAAAA" Gain: 31 dB

N° pictures taken: 16
Number of words: 64

Photoemission methodology

Experimental process:

Photoemission results for RAM writes:



Signal+power+noise

Photoemission parameters:

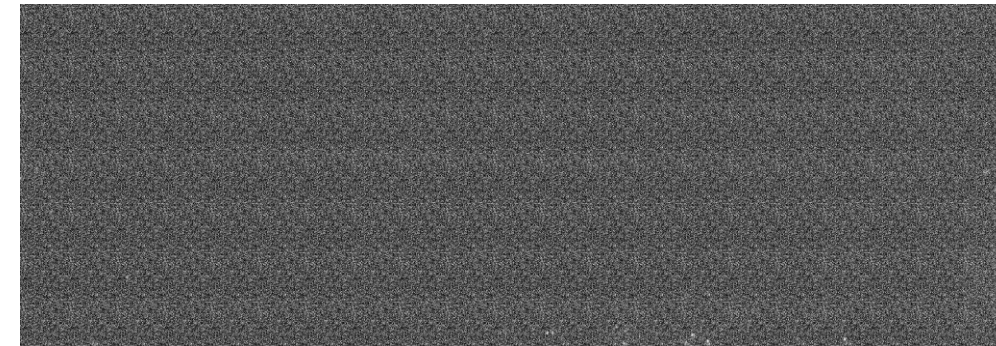
Objective x50

Exposure time: 2000 ms

Data: "0xAAAAAAAA"

Gain: 31 dB

—



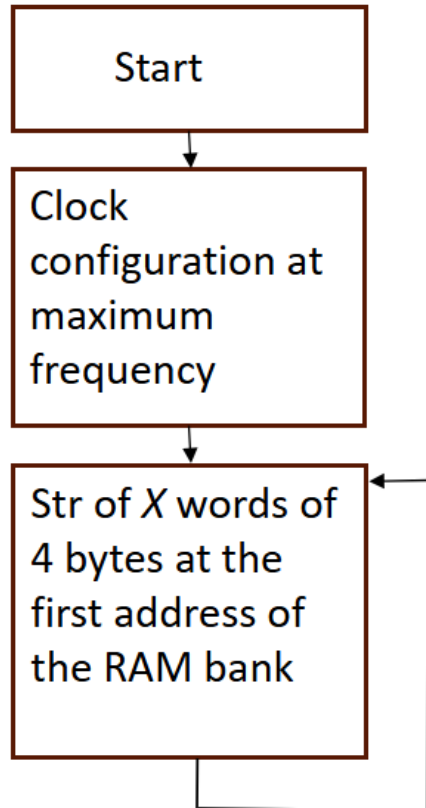
Power+noise

N° pictures taken: 16

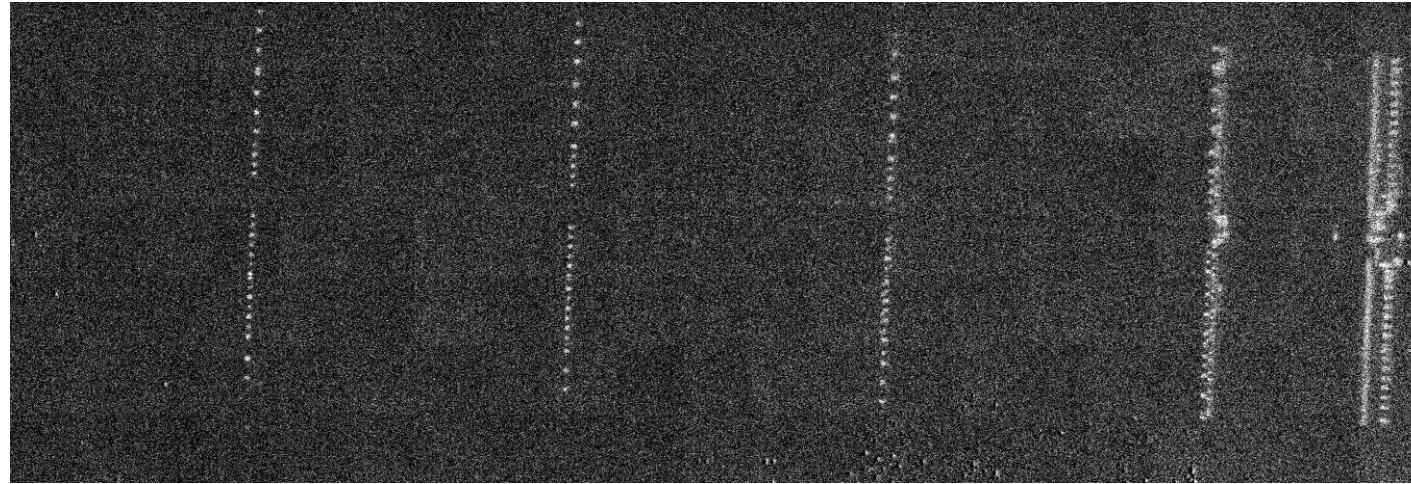
Number of words: 64

Photoemission methodology

Experimental process:



Photoemission results for RAM writes:



Result of the subtraction -> **Signal + residual noise**

Photoemission parameters:

Objective x50

Exposure time: 2000 ms

N° pictures taken: 16

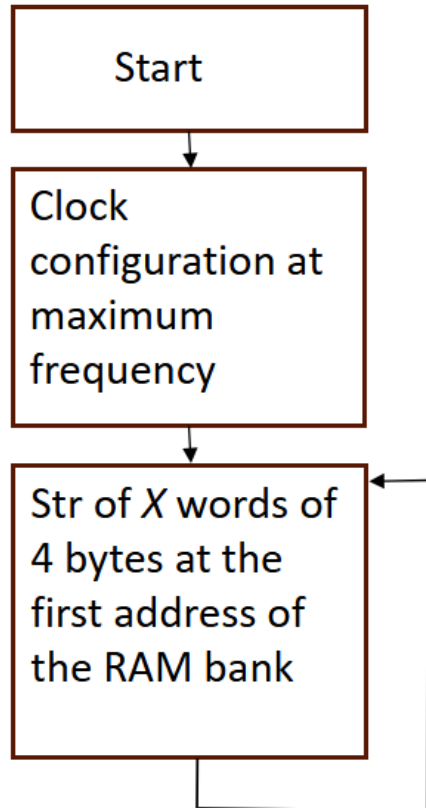
Data: "0xAAAAAAAA"

Gain: 31 dB

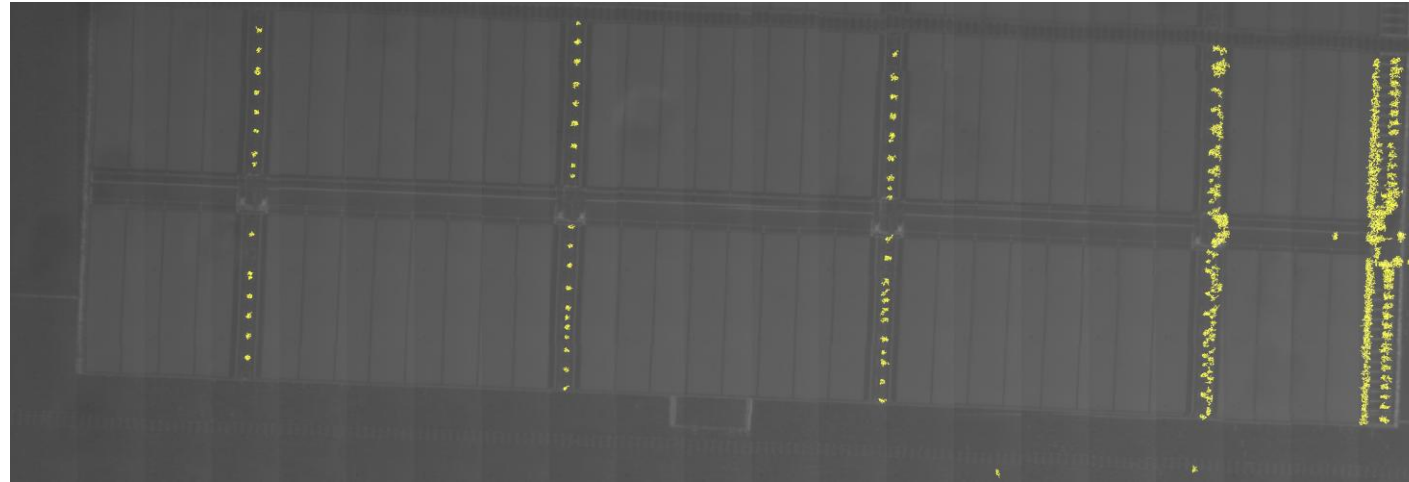
Number of words: 64

Photoemission methodology

Experimental process:



Photoemission results for RAM writes:



Overlay of the filtered photoemission image on an IR image

Grouping bright pixels into connected regions, then filtering them based on their size

Photoemission parameters:

Objective x50

Exposure time: 2000 ms

N° pictures taken: 16

Data: "0xAAAAAAAA"

Gain: 31 dB

Number of words: 64

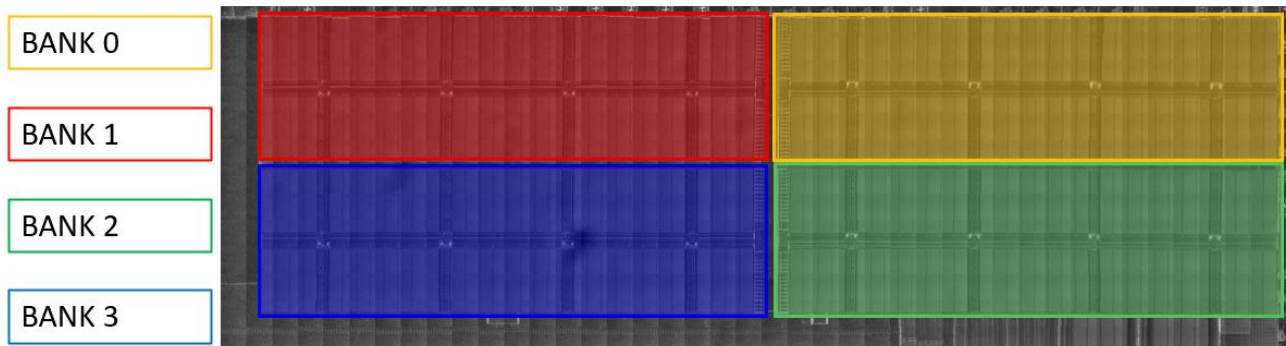
Results

First objective: Identify the BANK

Address	BANK
0x20000000	0
0x20000004	1
0x20000008	2
0x2000000c	3
0x20000010	0
0x20000014	1

Writing to a specific bank using the bank's first address

Write words into a single bank
-> first address+16



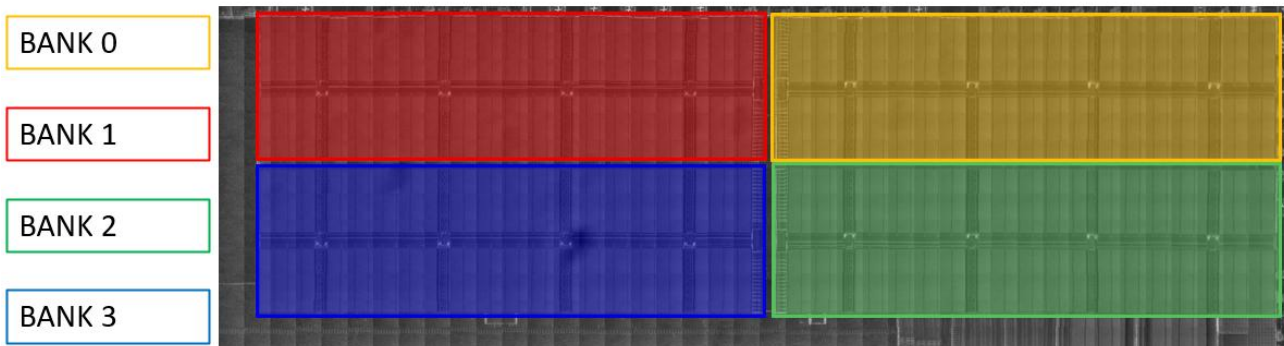
Results

First objective: Identify the BANK

Address	BANK
0x20000000	0
0x20000004	1
0x20000008	2
0x2000000c	3
0x20000010	0
0x20000014	1

Writing to a specific bank using the bank's first address

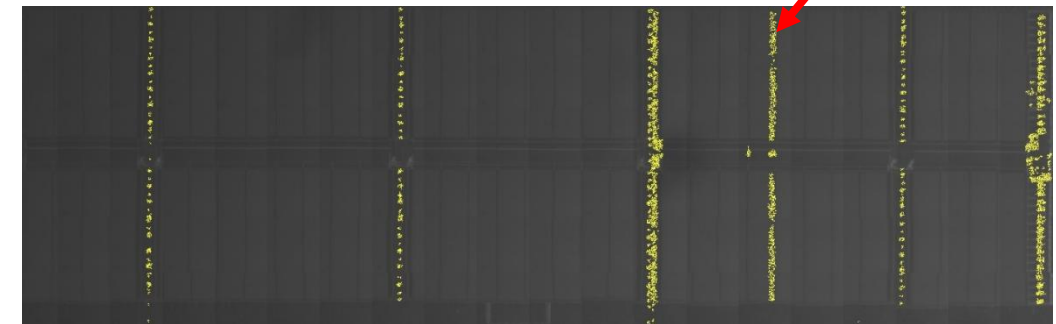
Write words into a single bank
-> first address+16



Second objective: Identifies the dataline



Data line

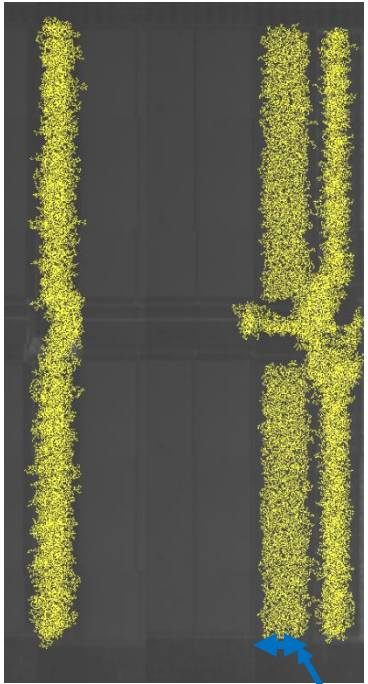


Photoemission data line from different positions in the BANK

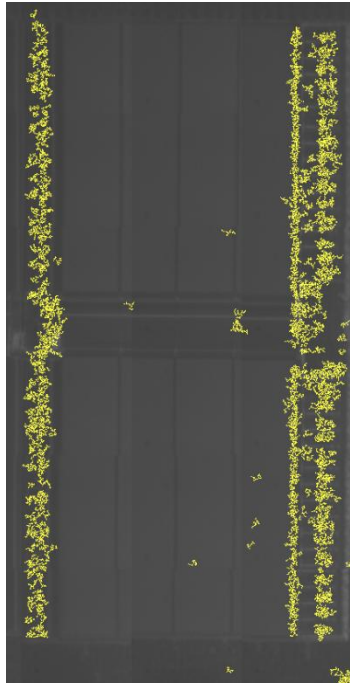
Results

First address 0x20000000

Writing more words activates more wordlines



512 word(32 bits)



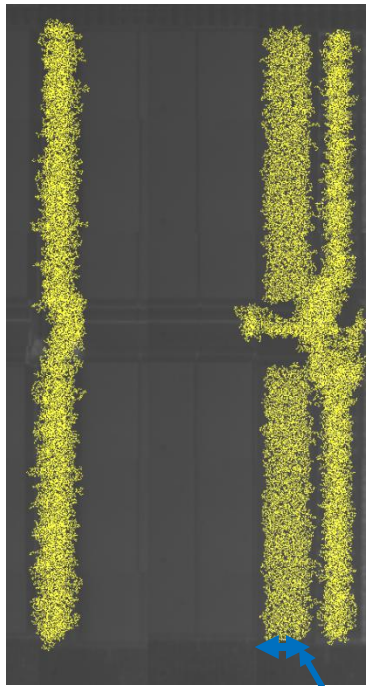
1word(32 bits)

First Hypothesis: Word line

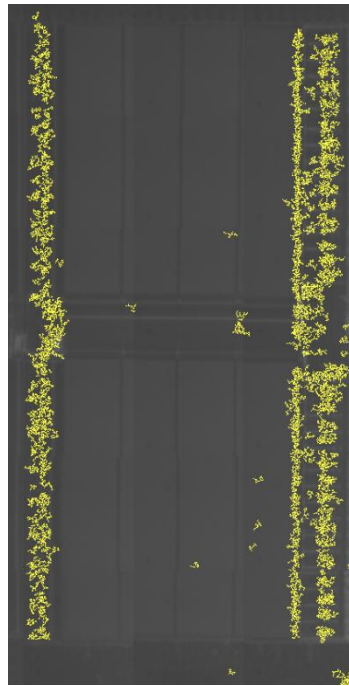
Results

First address 0x20000000

Writing more words activates more wordlines



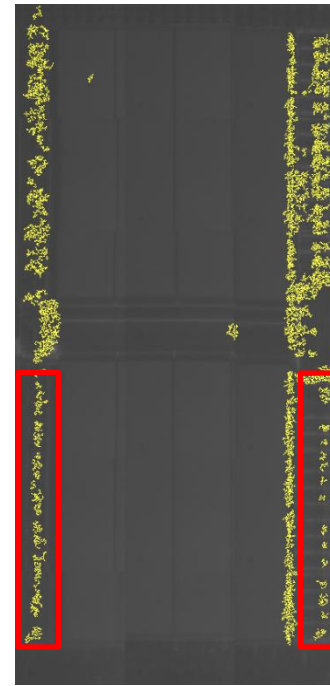
512 word(32 bits)



1word(32 bits)

First Hypothesis: Word line

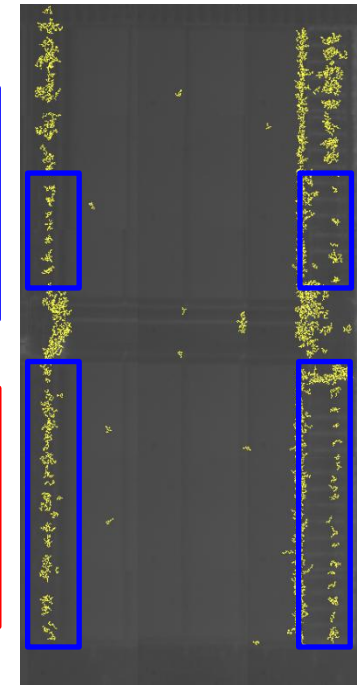
Writing a half and a byte to enable fewer Bitlines and \Bitline



1 half(16 bits)

Reduced by a factor of 4 compared to 1 word

Reduced by a factor of 2 compared to 1 word



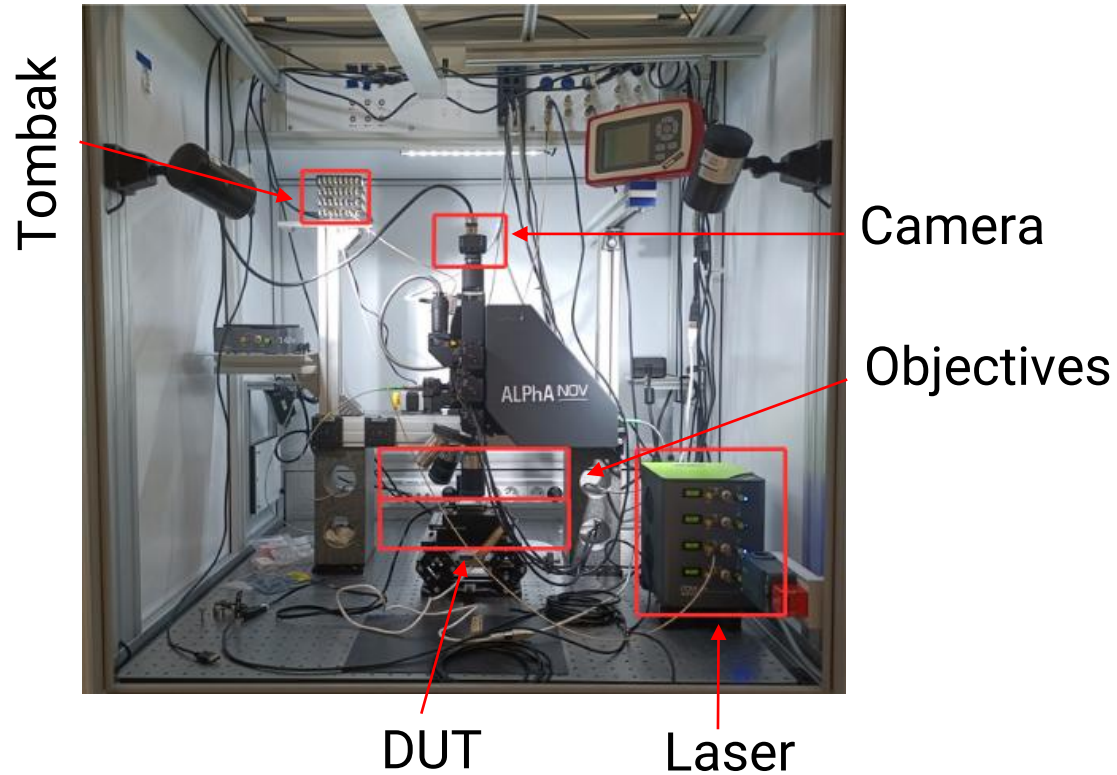
1byte (8 bits)

Second hypothesis: Bit line and \Bitline

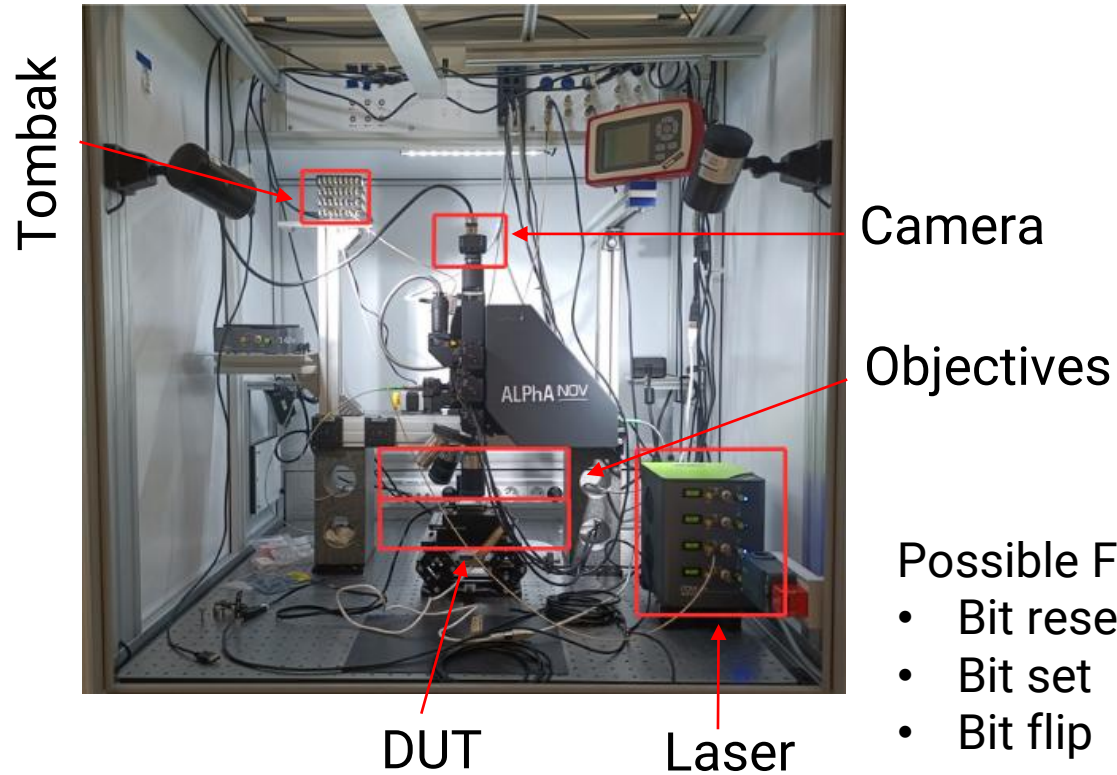
LFI — Characterization, FIRE, and Attack Implementation

- Step 3: LFI with $\times 20$ and $\times 50$ objectives
- Step 4: Implementation Attacks

LFI setup:



LFI setup:



Possible Fault model:

- Bit reset
- Bit set
- Bit flip

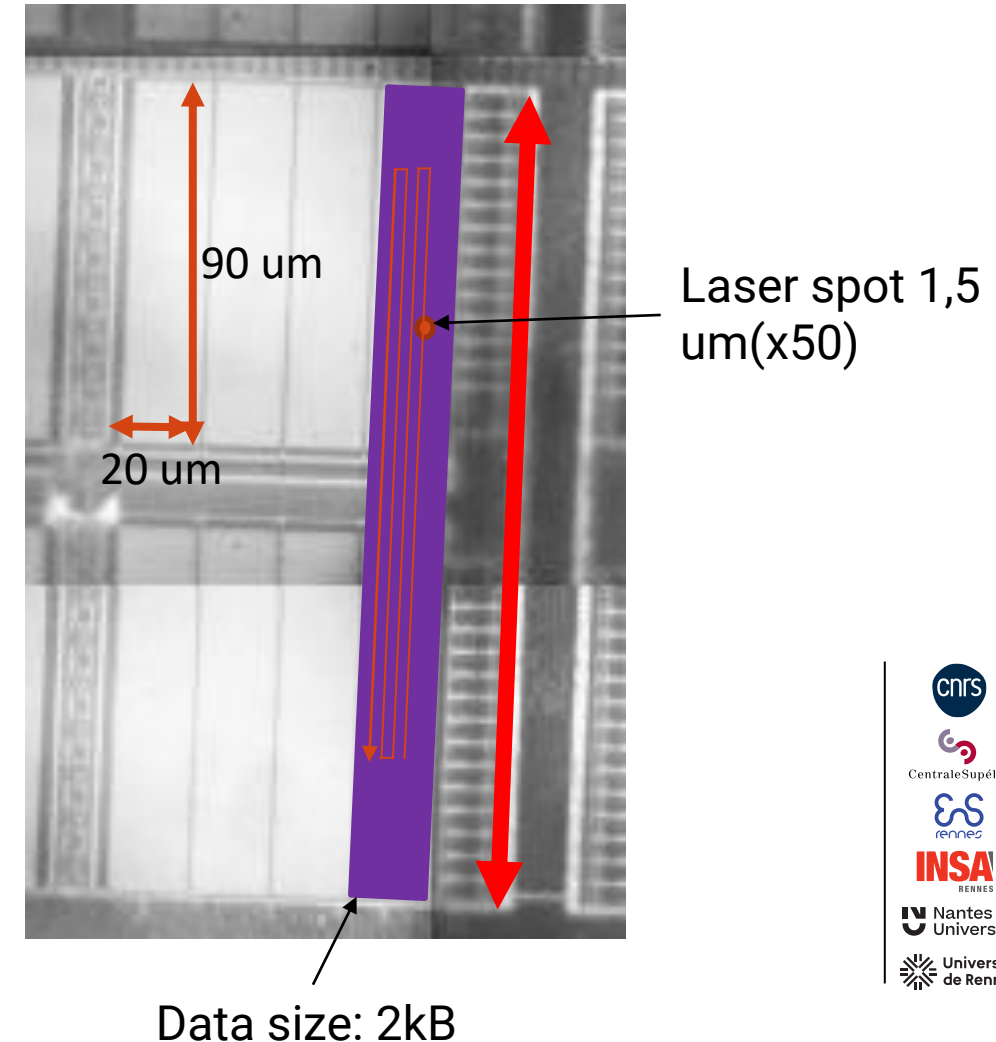
LFI PARAMETERS:

Objective: x20-x50

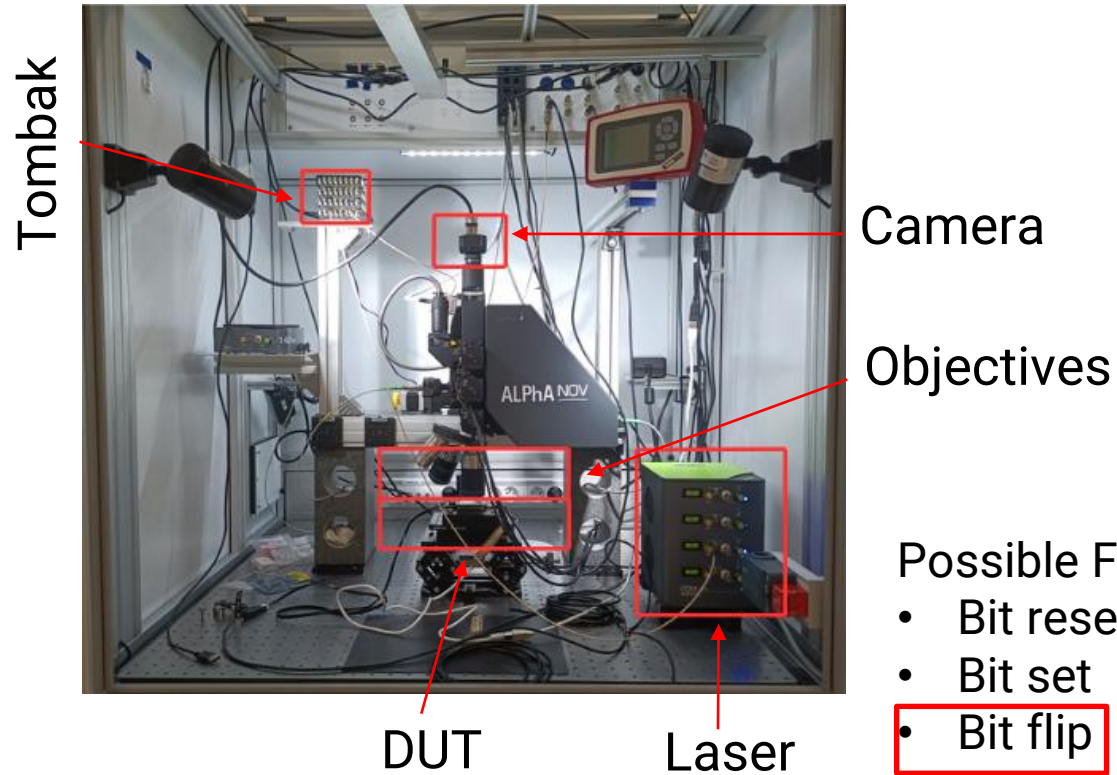
Power: 50-100 mW step: 50 mW

Pw: 50-500 ns step: 50 ns

Data : "0xAAAAAAAA"



LFI setup:



Possible Fault model:

- Bit reset
- Bit set
- Bit flip

Validation of hypothesis regarding the positions of word lines and bit lines

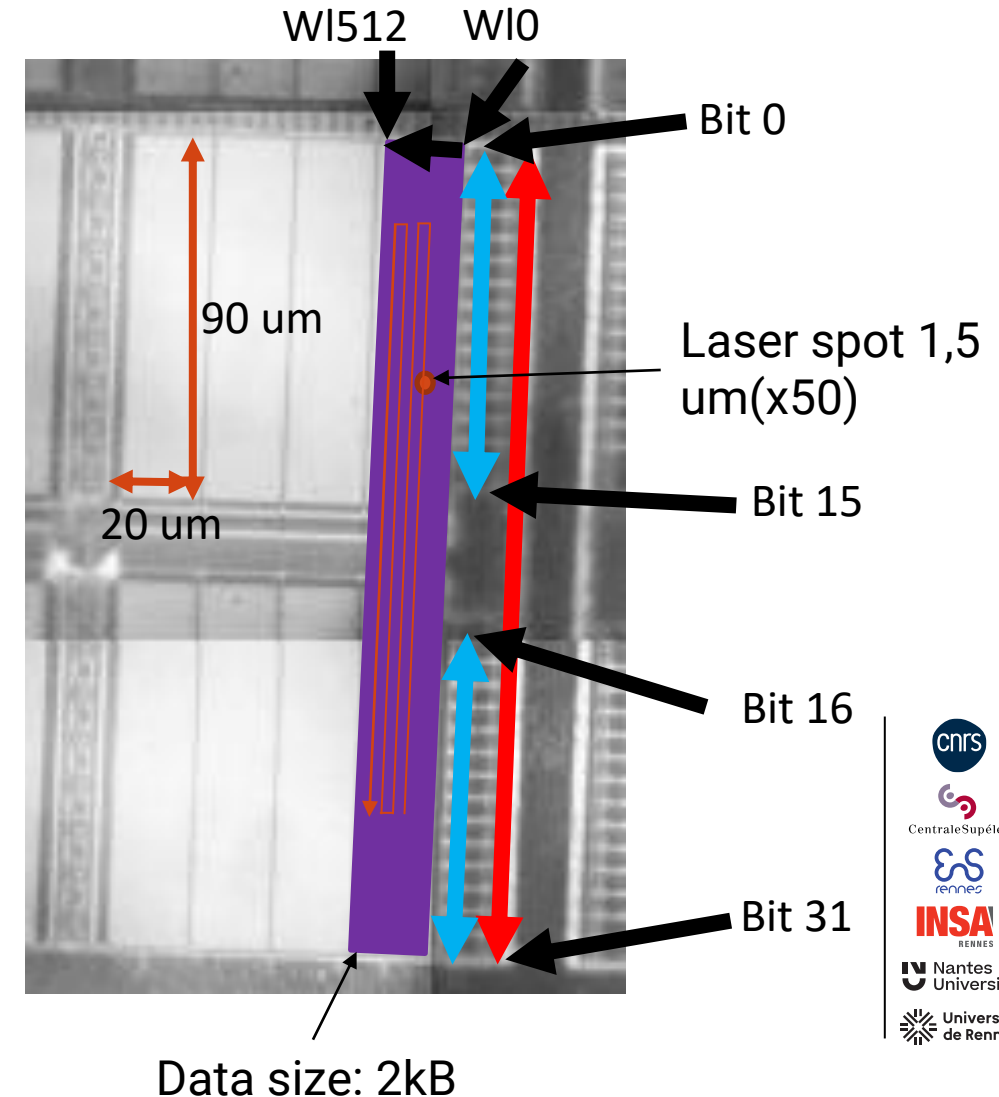
LFI PARAMETERS:

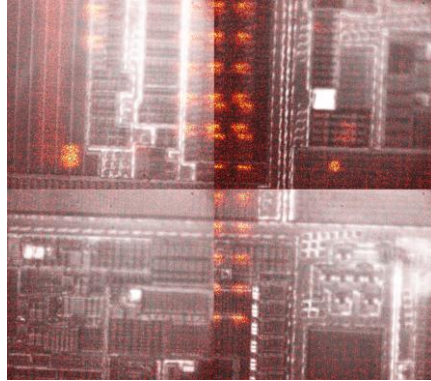
Objective: x20-x50

Power: 50-100 mW step: 50 mW

Pw: 50-500 ns step: 50 ns

Data : "0xAAAAAAAA"





Photon Emission Microscopy (PEM): 
Reverse engineering of SRAM

```
fibonacci(int):
    addi    sp,sp,-32
    sw      ra,28(sp)
    sw      s0,24(sp)
    sw      s1,20(sp)
    addi    sp,sp,32
    sw      a0,-20(a0)
    lw      a4,-20(a0)
    li      a5,1
    bgt     a5,a4,.L12
    lw      a5,-20(a0)
    .L12:
        lw      a5,-20(a0)
        addi    a5,a5,-1
        mv      a0,a5
        call    fibonacci(int)
        mv      s1,a0
        lw      a5,-20(a0)
        addi    a5,a5,-2
        mv      a0,a5
        call    fibonacci(int)
        mv      a5,a0
        add     a5,s1,a5
    .L13:
        mv      a0,a5
        lw      ra,28(sp)
        lw      s0,24(sp)
        lw      s1,20(sp)
        addi    sp,sp,32
        jr      ra
```

Implementation Attacks: 
Definition of attack scenarios

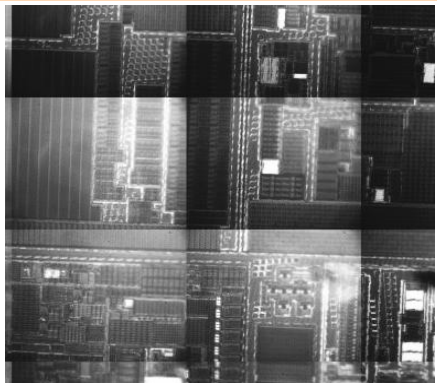
1

2

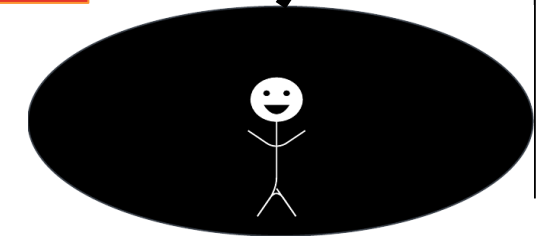
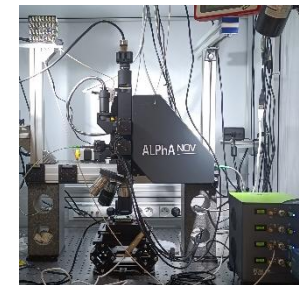
3

4

Visual Analysis: 
Structure Identification



LFI with ×20 and ×50 objectives: 
Fault Injection Reverse Engineering (FIRE) and classification of faults based on possible fault models .





Preliminary concept: VerifyPin FISSC

FISSC project: *Fault Injection and Simulation Secure Collection*[Dur16]

Version	security
V0	
V1	hardened bools
V2	hardened bools and fixed-time loop
V3	hardened bools, fixed-time loop and inlined function
V4	hardened bools, and fixed-time loop, inlined function, PTC decremented first, PTC backup and loop counter
V5	hardened bools, fixed-time loop, PTC decremented first and double call
V6	hardened bools, fixed-time loop, inlined call, PTC decremented first and double test
V7	hardened bools, fixed-time loop, inlined call, PTC decremented first, double test, step counter

VerifyPinV0:

```

BOOL byteArrayCompare(UBYTE* a1, UBYTE* a2, UBYTE size)
{
    int i;
    for(i = 0; i < size; i++) {
        if(a1[i] != a2[i]) {
            return 0;
        }
    }
    return 1;
}

BOOL verifyPIN() {
    g_authenticated = 0;

    if(g_ptc > 0) {
        if(byteArrayCompare(g_cardPin, g_userPin, PIN_SIZE) == 1) {
            g_ptc = 3;
            g_authenticated = 1; // Authentication();
            return 1;
        } else {
            g_ptc--;
            return 0;
        }
    }

    return 0;
}

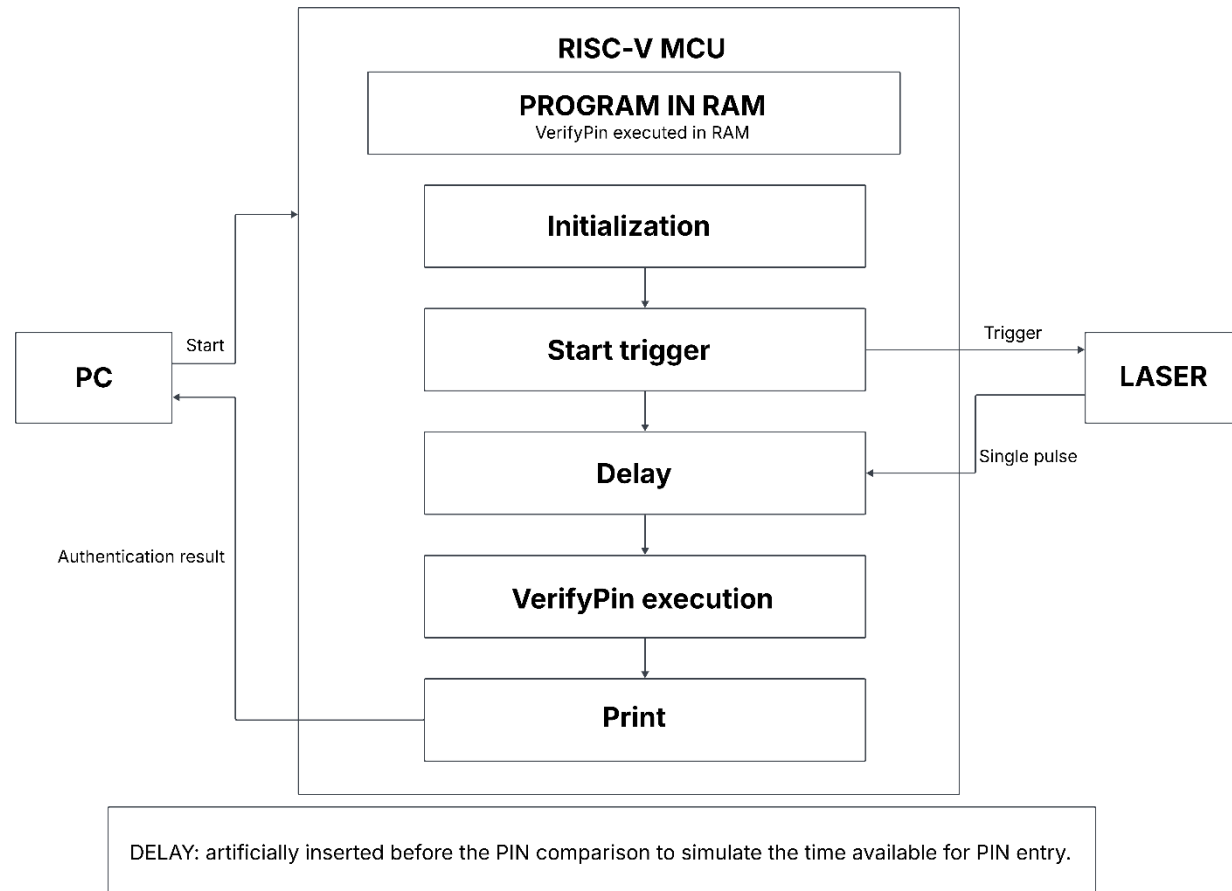
```



Experimental setup:

Hypothesis: The attacker is familiar with the program and has physical access to the target, but does not know the secret PIN. The attacker selects a candidate PIN and has three attempts.

- **Code executed in RAM**
- **Compiler: riscv32-unknown-elf-gcc + 13.2.0 + target riscv with GCC -O0 flag**
- **2 attacks**





Attack 1

byteArrayCompare in C:

```
BOOL byteArrayCompare(UBYTE* a1, UBYTE* a2, UBYTE size)
{
    int i;
    for(i = 0; i < size; i++) {
        if(a1[i] != a2[i]) {
            return 0;
        }
    }
    return 1;
}
```



Attack 1

byteArrayCompare in C:

```

BOOL byteArrayCompare(UBYTE* a1, UBYTE* a2, UBYTE size)
{
    int i;
    for(i = 0; i < size; i++) {
        if(a1[i] != a2[i]) {
            return 0;
        }
    }
    return 1;
}
    
```

byteArrayCompare in RISC V:

```

byteArrayCompare(unsigned char*, unsigned char*, unsigned char):
    addi    sp,sp,-48
    sw      ra,44(sp)
    sw      s0,40(sp)
    addi    s0,sp,48
    sw      a0,-36(s0)
    sw      a1,-40(s0)
    mv      a5,a2
    sb      a5,-41(s0)
    sw      zero,-20(s0)
    j       .L2

.L5:
    lw      a5,-20(s0)
    lw      a4,-36(s0)
    add     a5,a4,a5
    lbu     a4,0(a5)
    lw      a5,-20(s0)
    lw      a3,-40(s0)
    add     a5,a3,a5
    lbu     a5,0(a5)
    beq     a4,a5,.L3
    li      a5,0
    j       .L4

.L3:
    lw      a5,-20(s0)
    addi    a5,a5,1
    sw      a5,-20(s0)

.L2:
    lbu     a5,-41(s0)
    lw      a4,-20(s0)
    blt     a4,a5,.L5
    li      a5,1

.L4:
    mv      a0,a5
    lw      ra,44(sp)
    lw      s0,40(sp)
    addi    sp,sp,48
    jr      ra
    
```

Objective:
Jump to .L3
if $a1[i] \neq a2[i]$



Attack 1

byteArrayCompare in C:

```

BOOL byteArrayCompare(UBYTE* a1, UBYTE* a2, UBYTE size)
{
    int i;
    for(i = 0; i < size; i++) {
        if(a1[i] != a2[i]) {
            return 0;
        }
    }
    return 1;
}
    
```

beq a4, a5, 8 →

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
0	000000	01111	01110	000	0100	0	1100011	

byteArrayCompare in RISC V:

```

byteArrayCompare(unsigned char*, unsigned char*, unsigned char):
    addi    sp,sp,-48
    sw      ra,44(sp)
    sw      s0,40(sp)
    addi    s0,sp,48
    sw      a0,-36(s0)
    sw      a1,-40(s0)
    mv      a5,a2
    sb      a5,-41(s0)
    sw      zero,-20(s0)
    j        .L2

.L5:
    lw      a5,-20(s0)
    lw      a4,-36(s0)
    add     a5,a4,a5
    lbu     a4,0(a5)
    lw      a5,-20(s0)
    lw      a3,-40(s0)
    add     a5,a3,a5
    lbu     a5,0(a5)
    beq     a4,a5,.L3
    li      a5,0
    j        .L4

.L3:
    lw      a5,-20(s0)
    addi    a5,a5,1
    sw      a5,-20(s0)

.L2:
    lbu     a5,-41(s0)
    lw      a4,-20(s0)
    blt     a4,a5,.L5
    li      a5,1

.L4:
    mv      a0,a5
    lw      ra,44(sp)
    lw      s0,40(sp)
    addi    sp,sp,48
    jr      ra
    
```

Objective:
Jump to .L3
if a1[i] != a2[i]



Attack 1

byteArrayCompare in C:

```

BOOL byteArrayCompare(UBYTE* a1, UBYTE* a2, UBYTE size)
{
    int i;
    for(i = 0; i < size; i++) {
        if(a1[i] != a2[i]) {
            return 0;
        }
    }
    return 1;
}
    
```

byteArrayCompare in RISC V:

```

byteArrayCompare(unsigned char*, unsigned char*, unsigned char):
    addi    sp,sp,-48
    sw      ra,44(sp)
    sw      s0,40(sp)
    addi    s0,sp,48
    sw      a0,-36(s0)
    sw      a1,-40(s0)
    mv      a5,a2
    sb      a5,-41(s0)
    sw      zero,-20(s0)
    j       .L2

.L5:
    lw      a5,-20(s0)
    lw      a4,-36(s0)
    add     a5,a4,a5
    lbu     a4,0(a5)
    lw      a5,-20(s0)
    lw      a3,-40(s0)
    add     a5,a3,a5
    lbu     a5,0(a5)
    beq     a4,a5,.L3
    li      a5,0
    j       .L4

.L3:
    lw      a5,-20(s0)
    addi    a5,a5,1
    sw      a5,-20(s0)

.L2:
    lbu     a5,-41(s0)
    lw      a4,-20(s0)
    blt     a4,a5,.L5
    li      a5,1

.L4:
    mv      a0,a5
    lw      ra,44(sp)
    lw      s0,40(sp)
    addi    sp,sp,48
    jr      ra
    
```

Objective:
Jump to .L3
if a1[i]!=a2[i]

beq a4, a5, 8 →

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
0	000000	01111	01110	000	0100	0	1100011	

Fault models: one bitflip

blt a4, a5, 8

bne a4, a5, 8

Imm[12]	Imm[10:5]	rs2	rs1	funct3	Imm[4:1]	Imm[11]	opcode
0	000000	01111	01110	101	0100	0	1100011

Choice -> bne



Attack 2:

```
BOOL verifyPIN() {  
    g_authenticated = 0;  
  
    if(g_ptc > 0) {  
        if(byteArrayCompare(g_cardPin, g_userPin, PIN_SIZE) == 1) {  
            g_ptc = 3;  
            g_authenticated = 1; // Authentication();  
            return 1;  
        } else {  
            g_ptc--;  
            return 0;  
        }  
    }  
  
    return 0;  
}
```



Attack 2:

```
BOOL verifyPIN() {  
    g_authenticated = 0;  
  
    if(g_ptc > 0) {  
        if(byteArrayCompare(g_cardPin, g_userPin, PIN_SIZE) == 1) {  
            g_ptc = 3;  
            g_authenticated = 1; // Authentication();  
            return 1;  
        } else {  
            g_ptc--; → SKIP  
            return 0;  
        }  
    }  
  
    return 0;  
}
```



Attack 2:

```

BOOL verifyPIN() {
    g_authenticated = 0;

    if(g_ptc > 0) {
        if(byteArrayCompare(g_cardPin, g_userPin, PIN_SIZE) == 1) {
            g_ptc = 3;
            g_authenticated = 1; // Authentication();
            return 1;
        } else {
            g_ptc--; → SKIP
            return 0;
        }
    }

    return 0;
}
    
```

```

    su    a4,0(a5)
    li    a5,1
    j     .L10
.L9:
    lui   a5,0x20001
    lb    a5,853(a5)
    zext.b a5,a5
    c.addi a5,a5,-1
    zext.b a5,a5
    sext.b a4,a5
    lui   a5,0x20001
    sb    a4,853(a5) → SKIP
    li    a5,0
    j     .L10
.L8:
    li    a5,0
.L10:
    mv    a0,a5
    lw    ra,12(<n)
    
```



Attack 2:

```

BOOL verifyPIN() {
    g_authenticated = 0;

    if(g_ptc > 0) {
        if(byteArrayCompare(g_cardPin, g_userPin, PIN_SIZE) == 1) {
            g_ptc = 3;
            g_authenticated = 1; // Authentication();
            return 1;
        } else {
            g_ptc--; → SKIP
            return 0;
        }
    }

    return 0;
}
    
```

```

    su    a4,0(a5)
    li    a5,1
    j     .L10

.L9:
    lui   a5,0x20001
    lb    a5,853(a5)
    zext.b a5,a5
    c.addi a5,a5,-1
    zext.b a5,a5
    sext.b a4,a5
    lui   a5,0x20001
    sb    a4,853(a5) → SKIP

    li    a5,0
    j     .L10

.L8:
    li    a5,0

.L10:
    mv    a0,a5
    lw    ra,12(<a0)
    
```

sb a4, 853(a5) →

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
-----------	-----	-----	--------	----------	--------	--------

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
0000000	01111	01110	000	01001	0100011

beq a4, a5, 2900 →

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
---------	-----------	-----	-----	--------	----------	---------	--------	--------

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
0	011010	01110	01111	000	1010	1	1100011

Fault models: one bitflip



Version	security	Attack 1	Attack 2
V0		✓	✓
V1	hardened bools	✓	✓
V2	hardened bools and fixed-time loop	✓	✓
V3	hardened bools, fixed-time loop and inlined function	✓	✓
V4	hardened bools, and fixed-time loop, inlined function, PTC decremented first, PTC backup and loop counter	✓	✗
V5	hardened bools, fixed-time loop, PTC decremented first and double call	✓	✓
V6	hardened bools, fixed-time loop, inlined call, PTC decremented first and double test	✓	✓
V7	hardened bools, fixed-time loop, inlined call, PTC decremented first, double test, step counter	✓	✓

Conclusion:

- ✓ **PEM:** Identification of active zones and **reverse engineering** of the **SRAM**
- ✓ **LFI:** Characterization of laser-induced faults, identification of associated **fault models**, and correlation between the **physical location** of the injection and the **logical location** (FIRE)
- ✓ Identification of **attack scenarios**

Main difficulty:

- **20x - Side Effects** of the Generation
- **50x - internal and external vibration and thermal expansion** of materials, causing **positional drift**

Prospects:

- Develop attacks exploiting persistent faults in security-critical applications

En going work:

- The embedded code can be fully loaded into the cache; with the **validation** of **asynchronous attacks** via RAM code corruption, the same **attack process** can be **applied to the cache**
 - ✓ Reverse engineering the Set Positions and Bit Positions
 - ✓ Asynchronous fault injection
- Identifying ways in sets for mapping and executing attacks (en going work)

- [Lim24] R. S. Lima and al, “Reverse-Engineering and Data Extraction from SRAM using Photon Emission Analysis”, DOI: 10.1109/PAINE62042.2024.10792702, 2024.
- [Per24] H. Perrin and al, “Betrayed by Light: How Photon Emission Microscopy Empowers Register Bit-Level Laser Attacks on Microcontrollers”, DOI: 10.1109/HOST64725.2025.11050055, 2024.
- [Che18] S. Chef and al, “Descrambling of Embedded SRAM Using a Laser Probe”, DOI: 10.1109/IPFA.2018.8452604, 2018.
- [Col19] B. Colombier and al, “Laser-induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-bit Microcontroller”, DOI: 10.1109/HST.2019.8741030, 2019.
- [Zha20] F. Zhang and al, “Persistent Fault Attack in Practice”, DOI: <https://doi.org/10.13154/tches.v2020.i2.172-195>, 2020.
- [Dur16] L. Dureuil and al, “*FISSC: a Fault Injection and Simulation Secure Collection*”, DOI: 10.1007/978-3-319-45477-1_1, 2016.
- [Cas26] S. Casavecchia and al, “Exploring Laser Fault Injection in System-on-Chips: A New Approach for CPU and Cache Attacks”, DOI: 10.1109/HOST68814.2026.11604707, 2026.
- [Cul24] A. Cullen, “Hacking the RP2350”, URL: <https://www.youtube.com/watch?v=hbFRnPNQnoY>, 2024.

Acknowledgements

This study is supported/partially funded by the ANR within the framework of the PIA EUR CyberSchool project (ANR-18-EURE-0004)

This study is supported/partly funded by the ANR as part of the Cyberskills4all project (ANR-23-CMAS-0026).

My thesis directors and supervisors