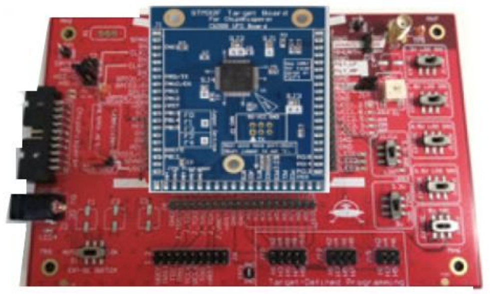
A BIT LEVEL APPROACH TO SIDE CHANNEL BASED DISASSEMBLING

Valence Cristiani, Maxime Leconte, Thomas Hiscock | 11/12/2019

GOAL



Target board

Side channel measurments

Oscilloscope

Computer

⋮

Xor r2, r3, r2
Mov r0, r1
Sub r1, r2, r3

⋮

Instruction flow

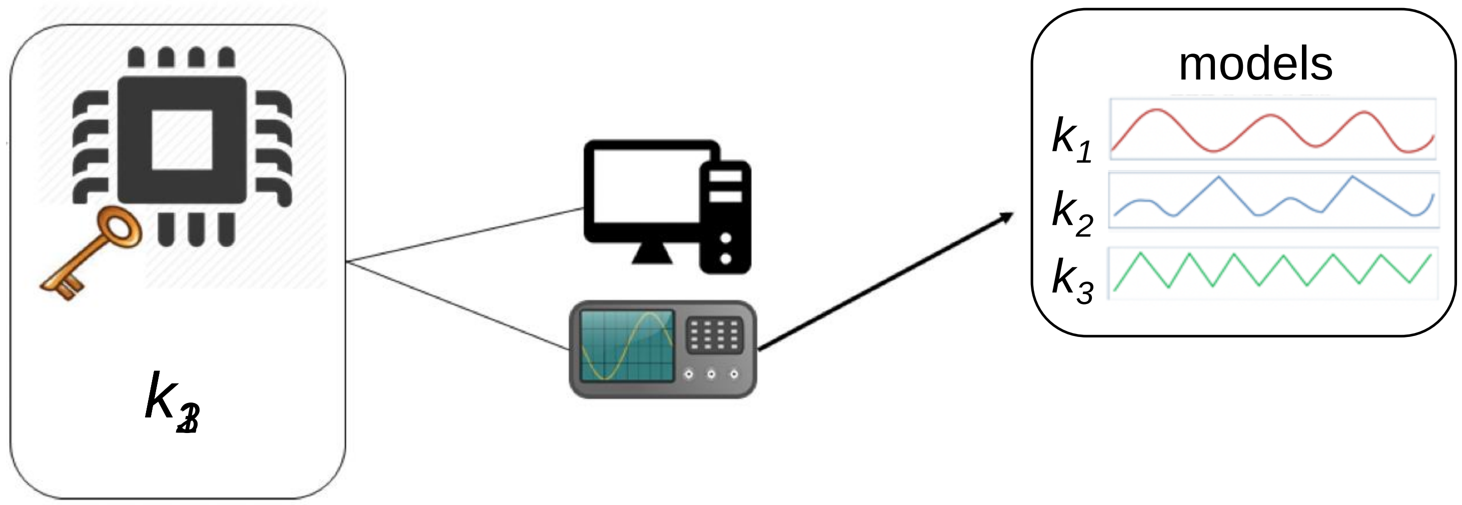
WHY IS IT INTERESTING ?

- ➢ Leakage characterization of common processor architecture
- ➢ Detect interesting zones of the code
  - AES
  - Function entry/exit point
  - Combine with fault injection attack
- ➢ Detect malwares
- ➢ Reverse proprietary source code

SUMMARY

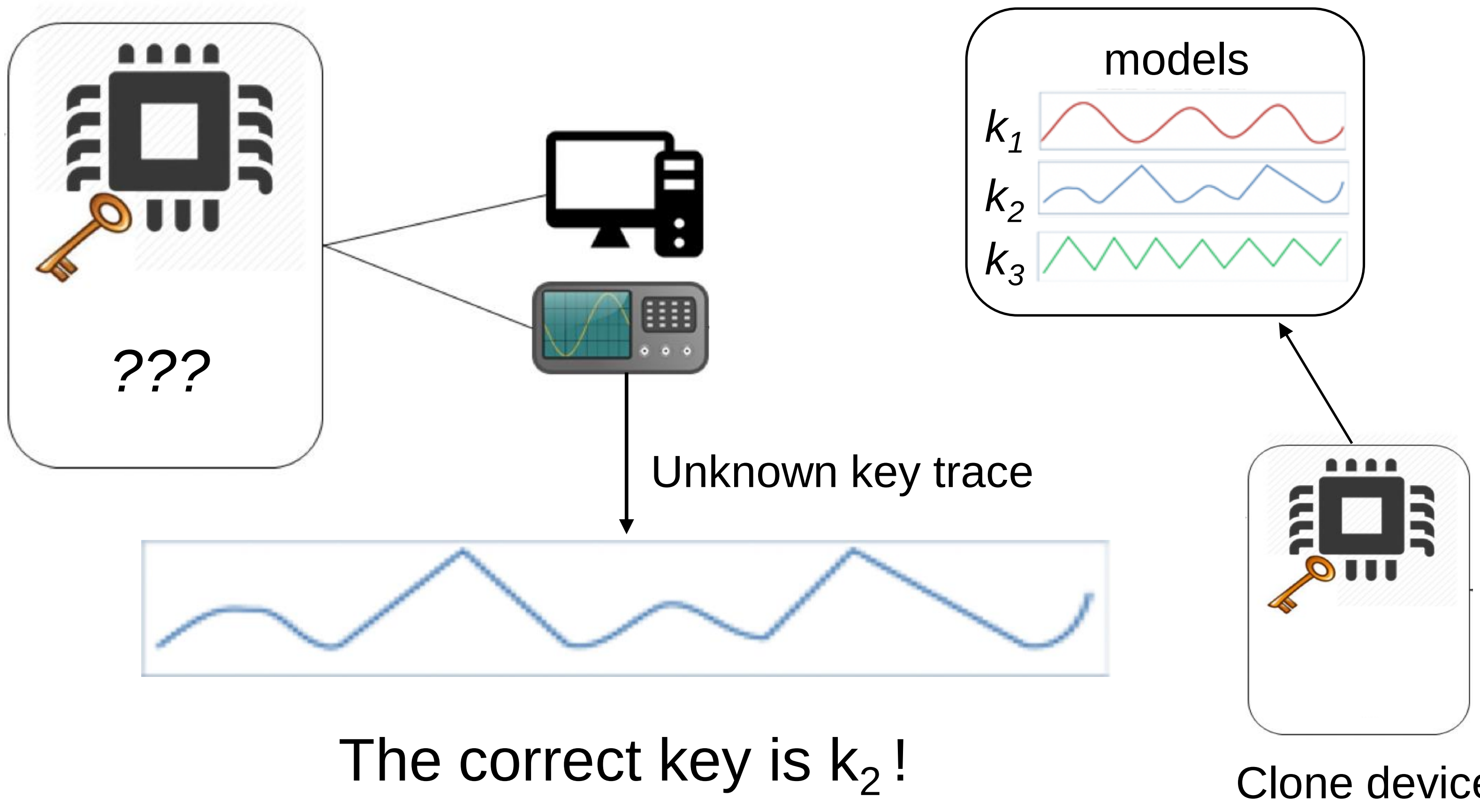
TEMPLATE ATTACK

- 3 possible secret keys : k_1, k_2 and k_3



TEMPLATE ATTACK

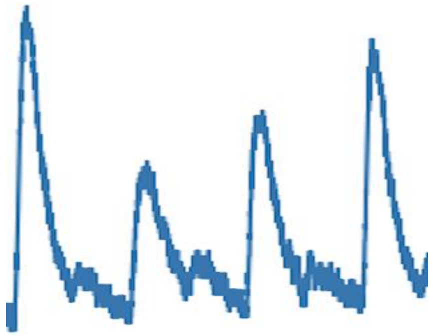
- 3 possible secret keys : k_1, k_2 and k_3



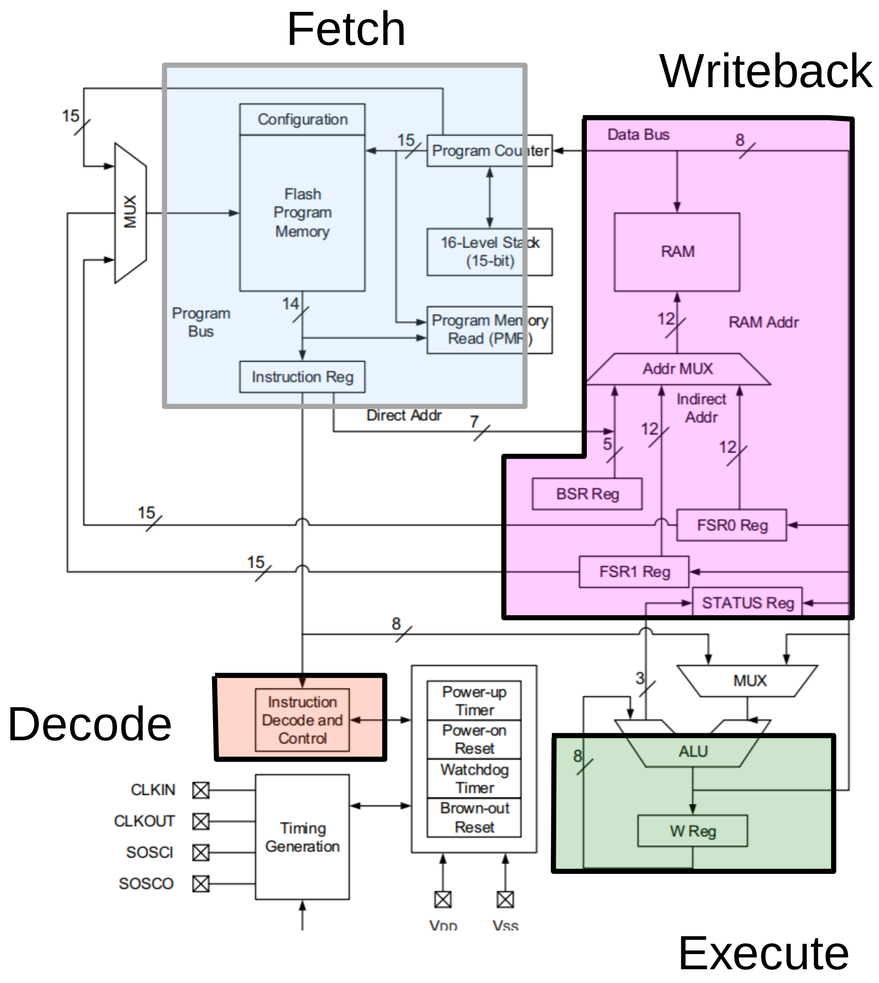
The correct key is k_2 !

PIC16F ARCHITECTURE

- Simple 8 bits microcontroller
- 14 bits instructions
- Why PIC ?
  - Most widely used target in SotA
  - Very simple
- 4 clock cycles per instruction
- 2 stages pipeline

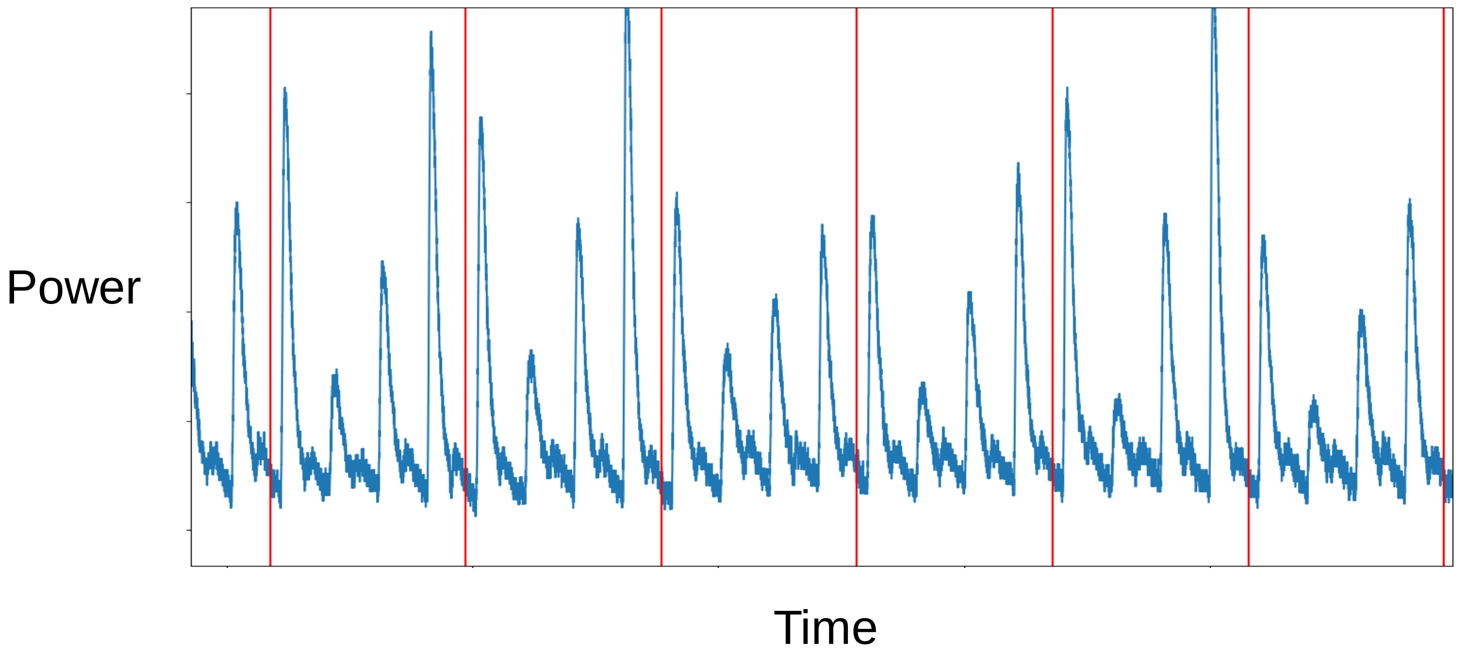


Typical instruction trace (Power)



INSTRUCTIONS EXTRACTION

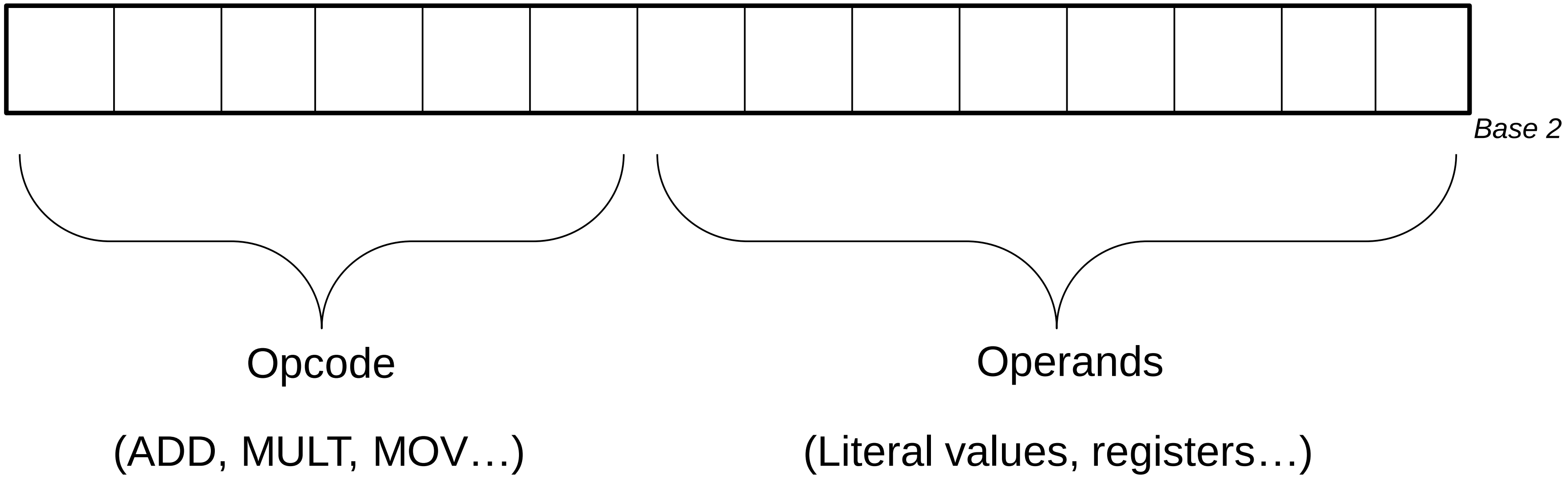
A very long trace…



- ➢ Instruction cutting and alignment ✓

ASSEMBLY INSTRUCTIONS

- Binary word (opcode+operands) stored in the instruction register

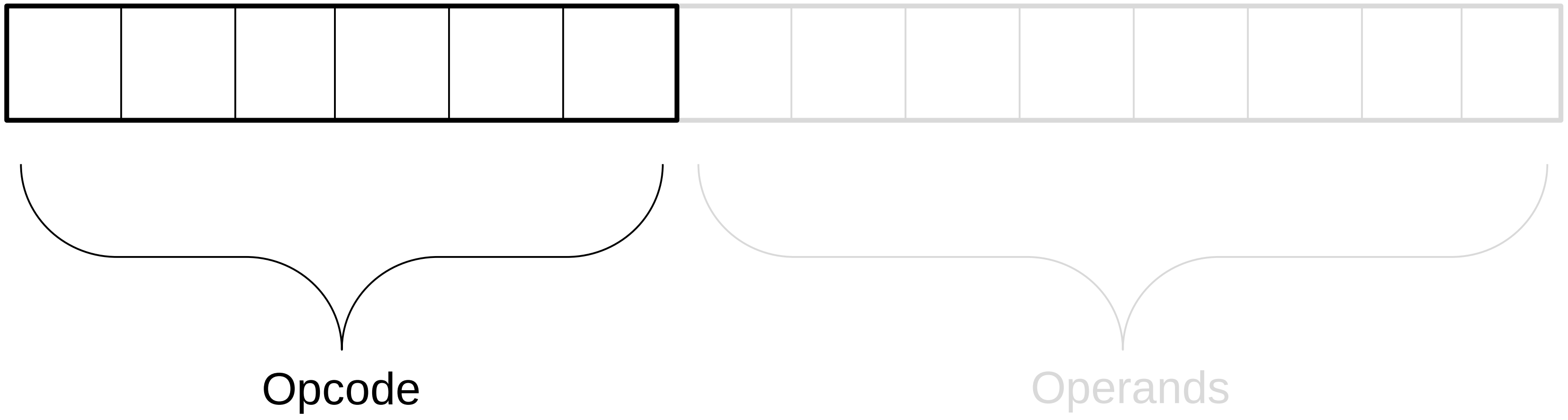


- Naïve solution: model each possible combination (opcode, operands) as a class $\longrightarrow$ Too many classes !

DIFFERENCES WITH KEY RECOVERING TEMPLATE

- Divide and conquer is not efficient. Where to divide ?
  - Number of operands is not fixed
  - Size of opcode and operands are not fixed

State of the art only focuses the opcodes !



STATE OF THE ART

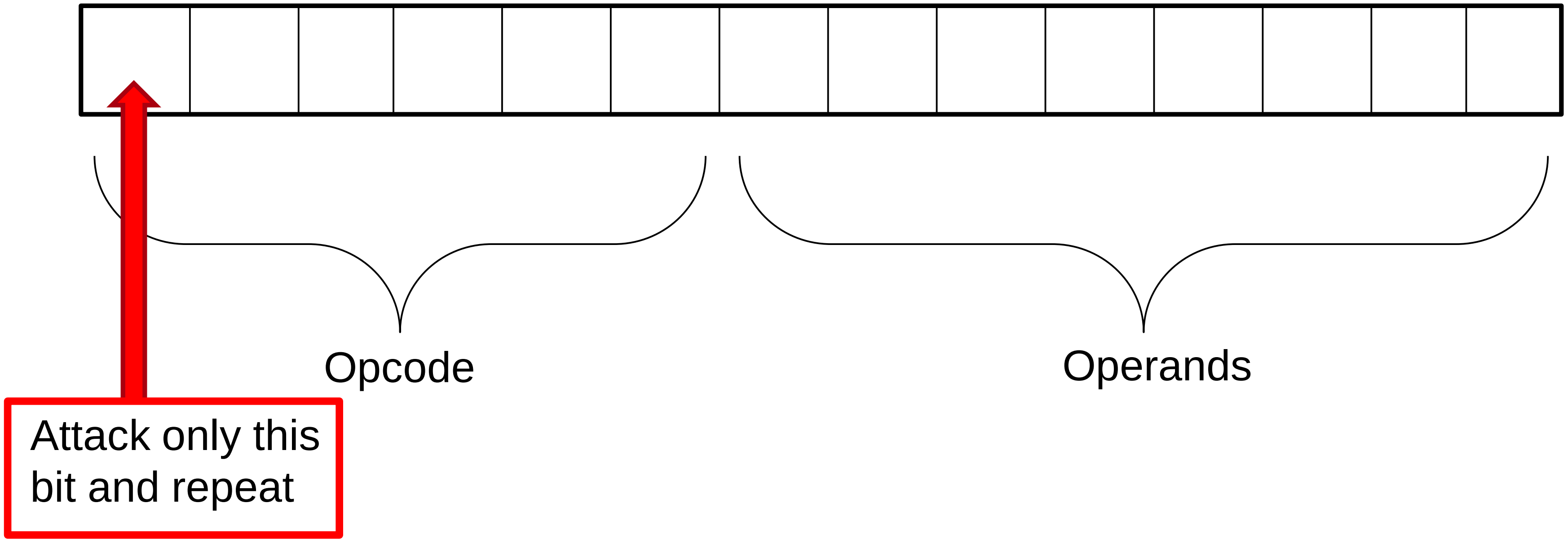
- Reach a good success rate (90 - 95%) on the PIC (Eisenbarth et al. 2010, Strobel et al. 2015)

- Usually do not recover the operands

- Require a long profiling phase with a lot of data

- Are not scalable to more complex processors with
    - More instructions (encoded on 32 bits)
    - Deeper pipeline

BIT ORIENTED TEMPLATE ATTACK ?



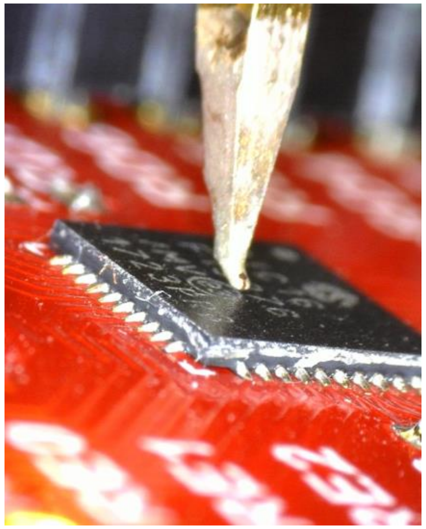
- Only 2 classes by template (0 and 1)
- Profiling can be done on random instructions (correctly labelled)
- We would get the operands as a bonus !

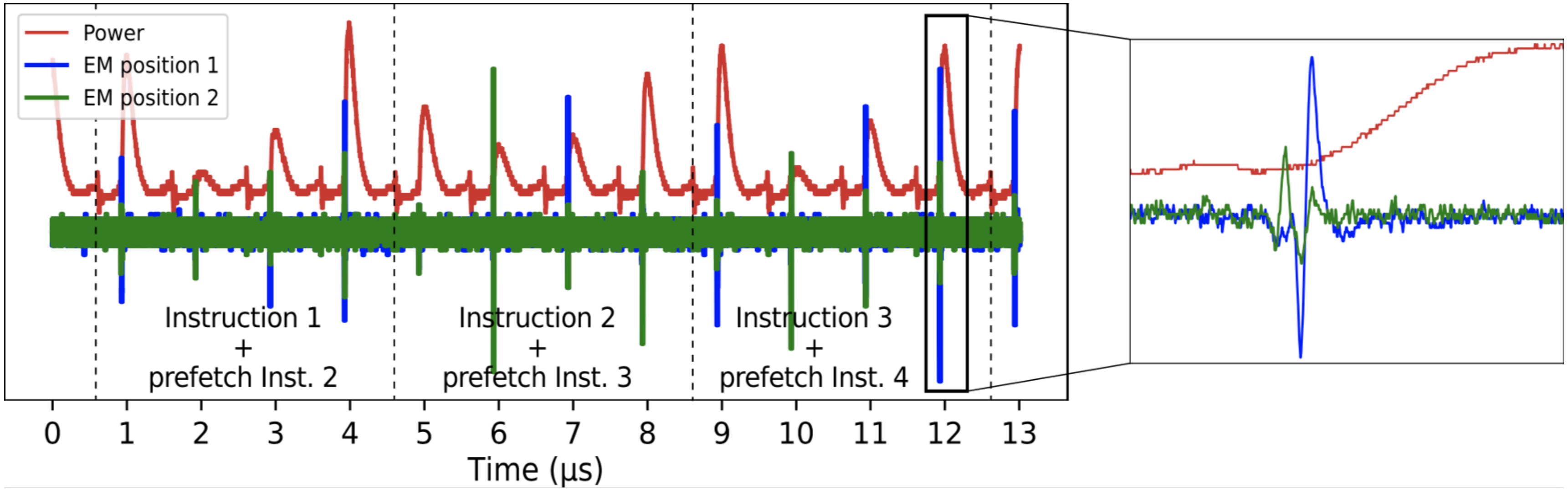
VIABILITY QUESTIONS…

1. Distinguish bit level variation (good enough SNR) ?
2. Does each bit have its own leakage ?
3. Does each bit leak independently ?
4. What is the leakage model ?

EM VS POWER

- ➢ EM + micro-probe → exploit local leakage.
- ➢ Leakage vary with probe position → cartography

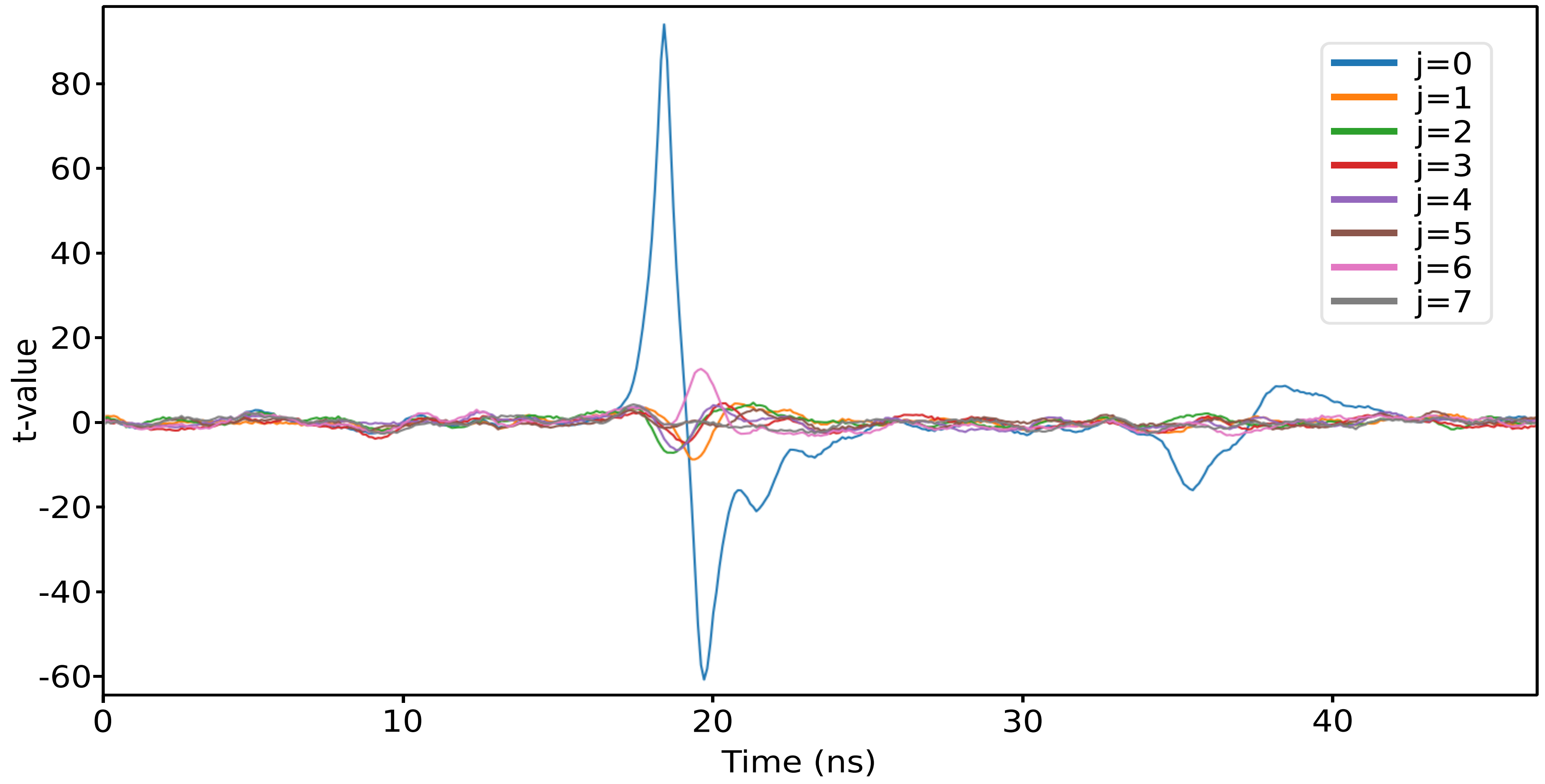




Power
EM position 1
EM position 2
Instruction 1 + prefetch Inst. 2
Instruction 2 + prefetch Inst. 3
Instruction 3 + prefetch Inst. 4
0
1
2
3
4
5
6
7
8
9
10
11
12
13
Time (µs)

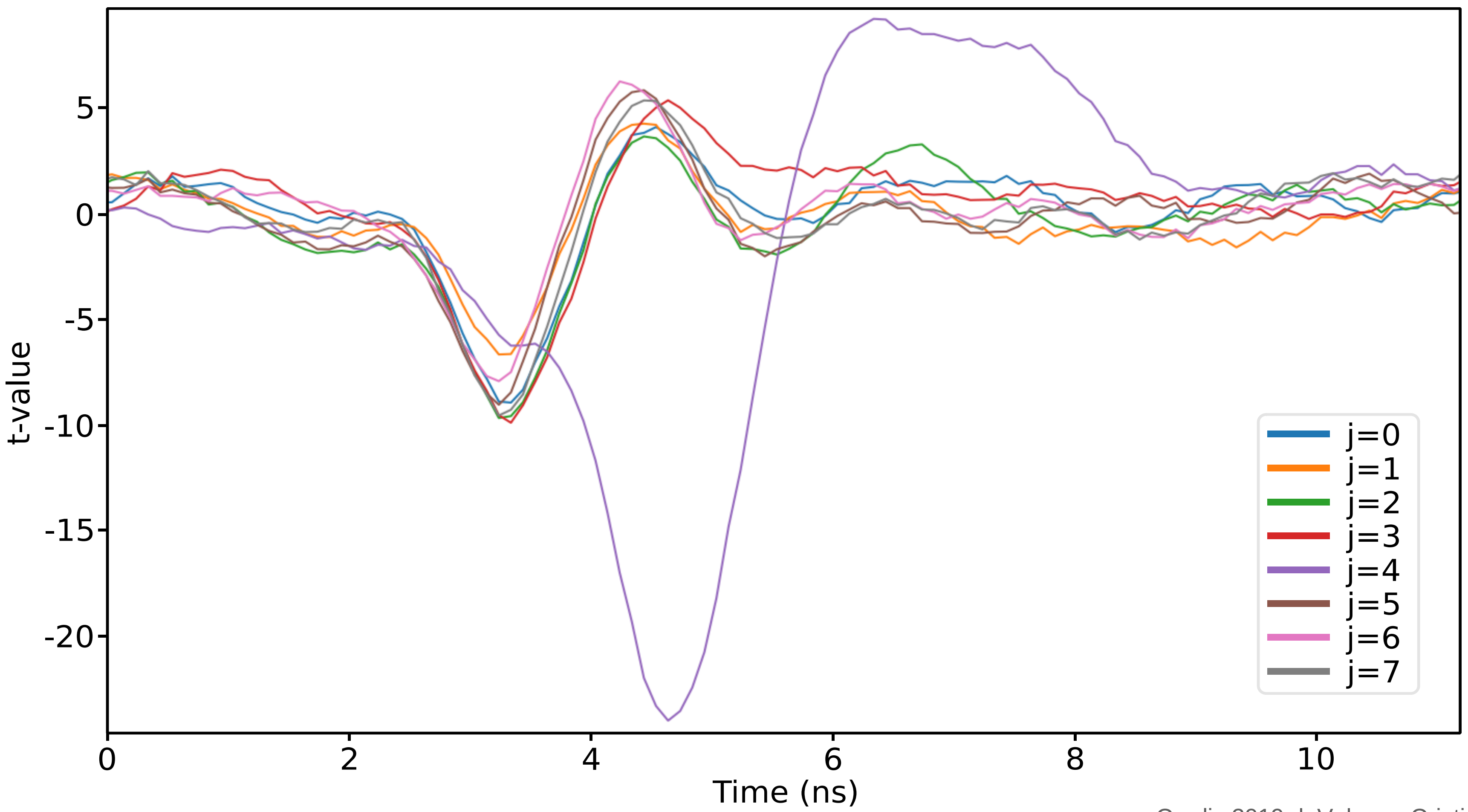
BEST PROBE POSITION FOR BIT 0

T-test between Mov 0 and Mov 2^j $(00\ldots1\ldots00_2)$

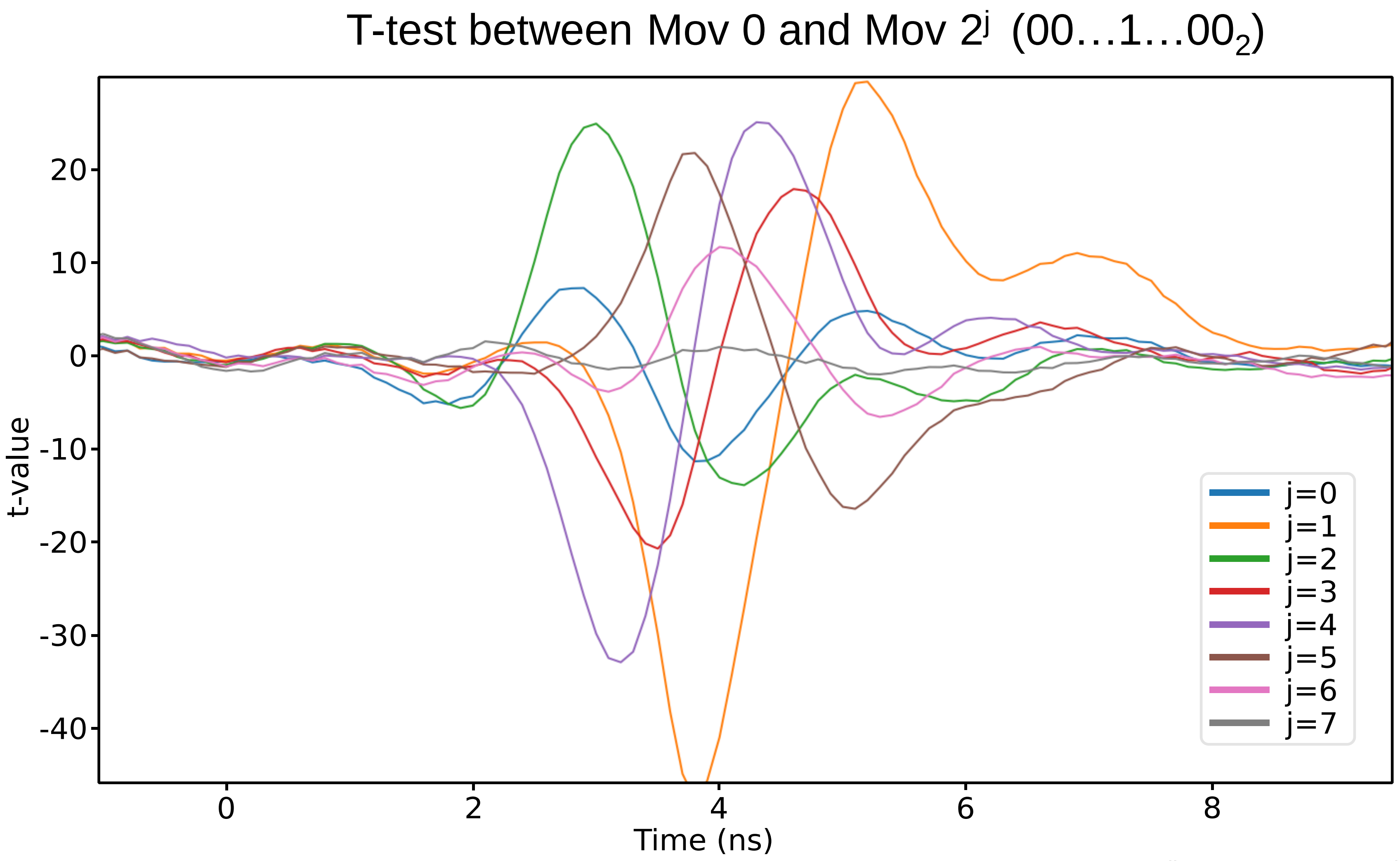


BEST PROBE POSITION FOR BIT 4

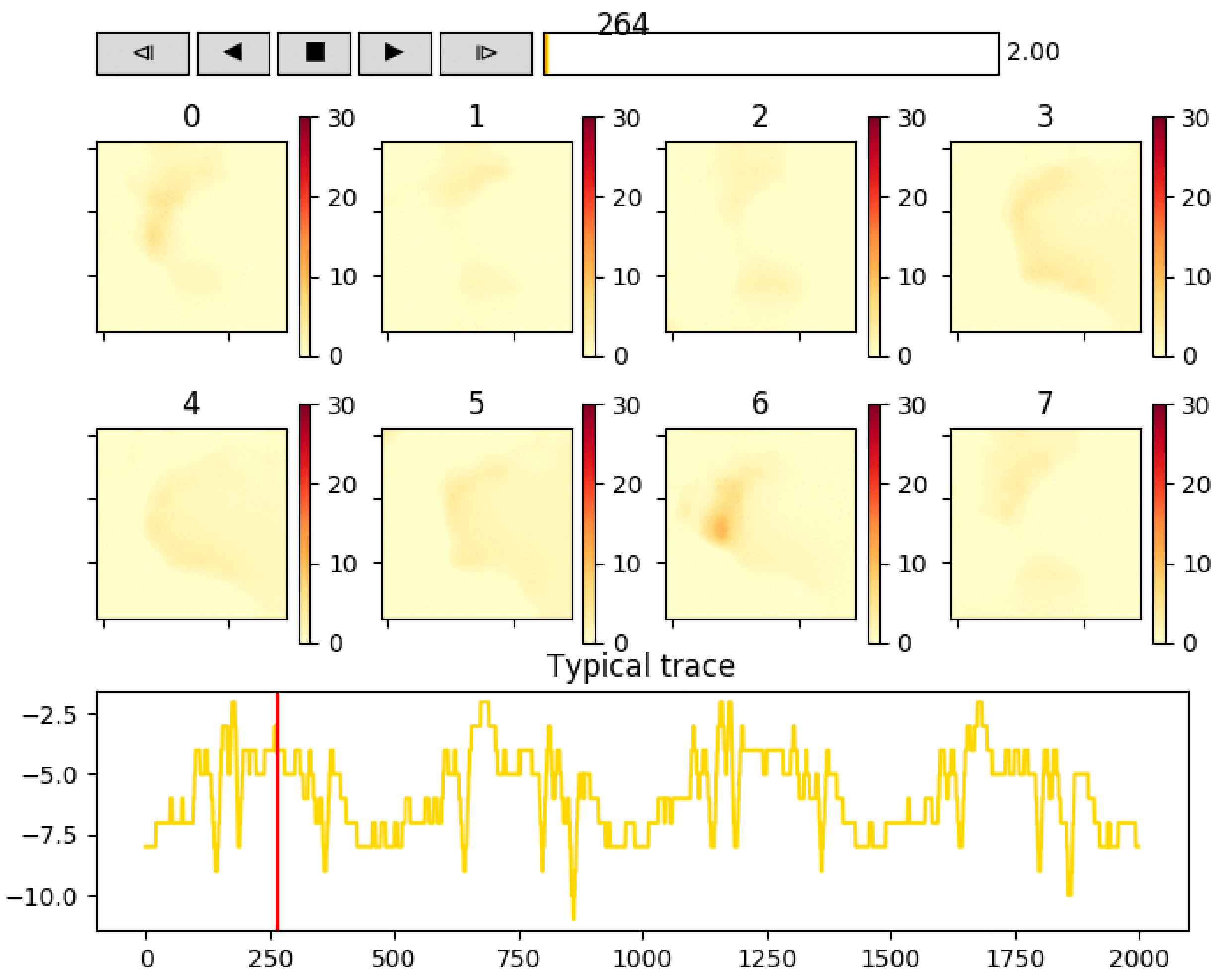
T-test between Mov 0 and Mov 2^j ($00\ldots1\ldots00_2$)



… FOR ALL THE BITS

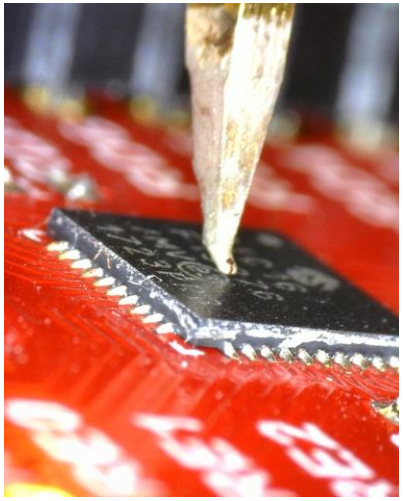


SPATIAL AND TEMPORAL LEAKAGE



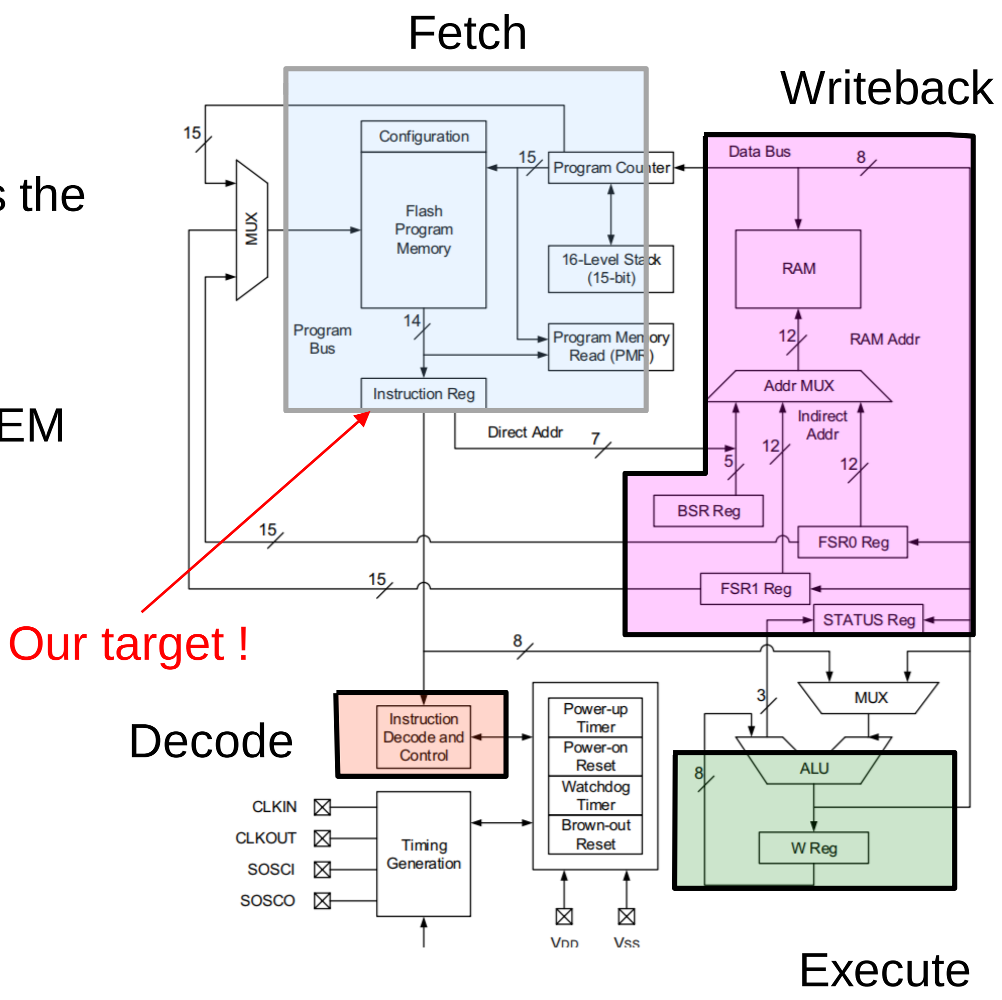
VIABILITY QUESTIONS…

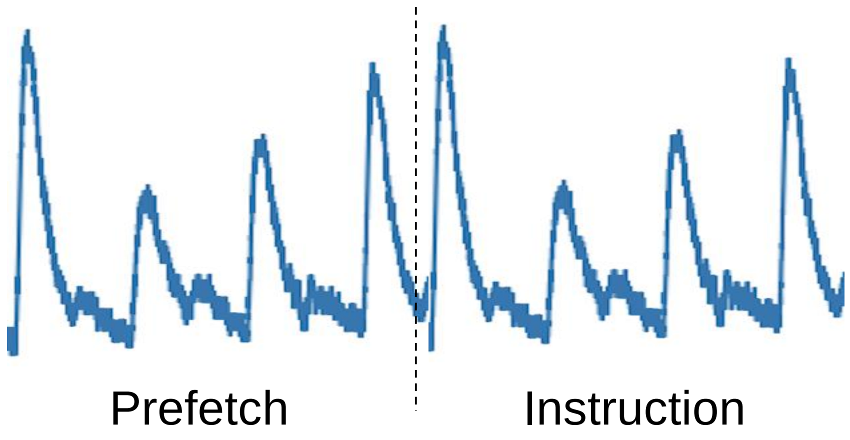
1. Distinguish bit level variation (good enough SNR) ? ✓
2. Does each bit have its own leakage ? ✓
3. Does each bit leak independently ?
4. What is the leakage model ?



PIC16F ARCHITECTURE

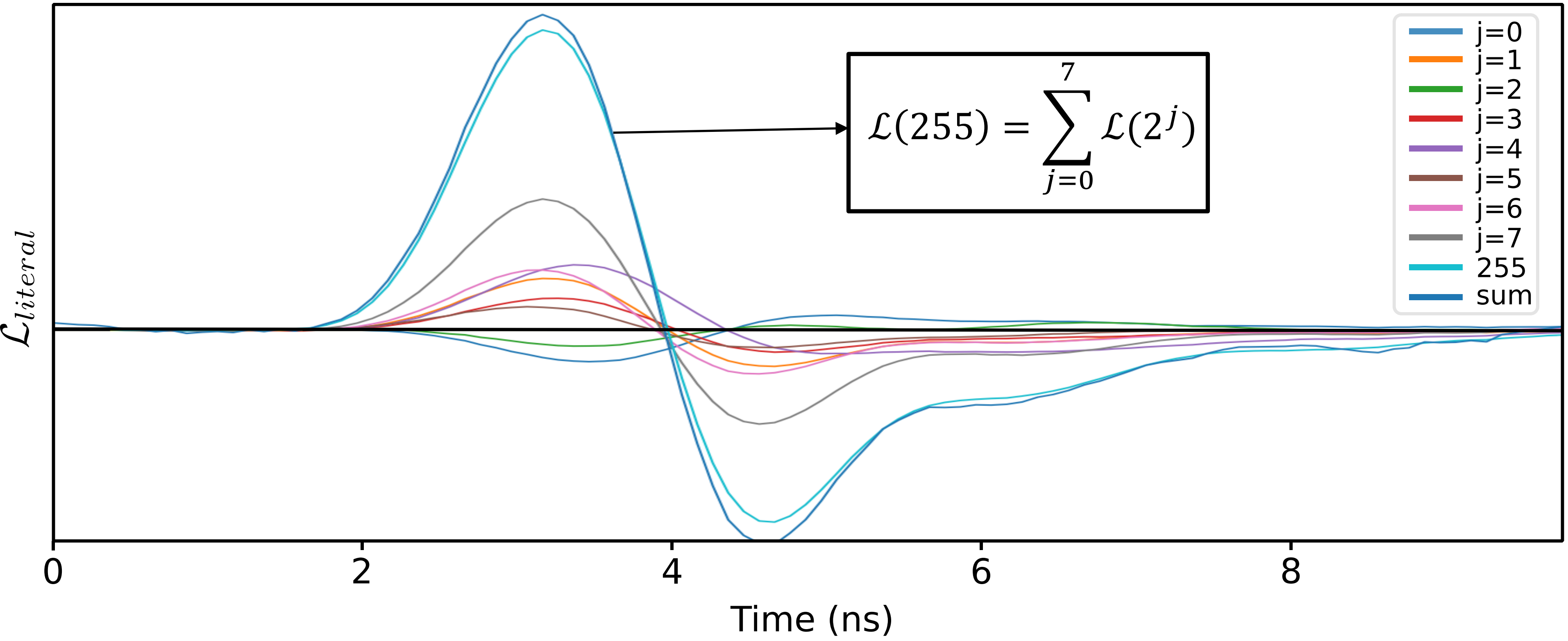
- ➢ Attack only the prefecth to focus the updtate of the intruction register
- ➢ Each bit may have its own independent contribution to the EM





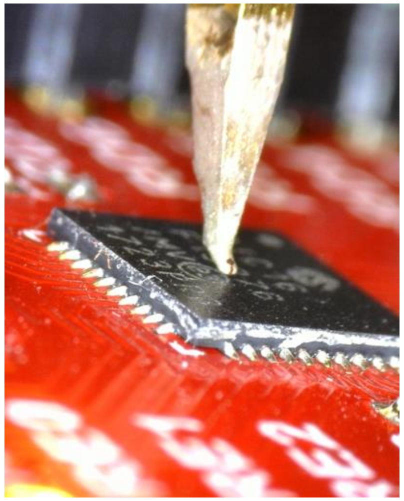
BIT INDEPENDENCE

$$\mathcal{L}(2^j) = Leakage(Mov\ 2^j) - Leakage\ (Mov\ 0)$$



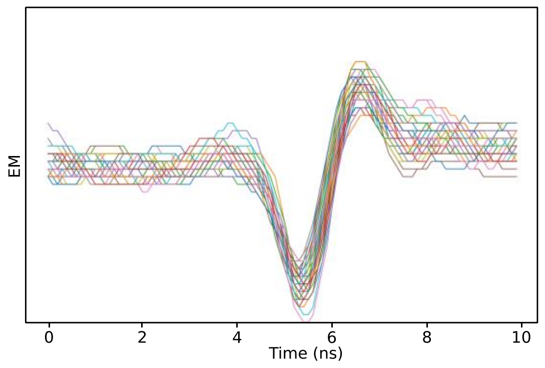
VIABILITY QUESTIONS…

1. Distinguish bit level variation (good enough SNR) ? ✓
2. Does each bit have its own leakage ? ✓
3. Does each bit leak independently ? ✓
4. What is the leakage model ?

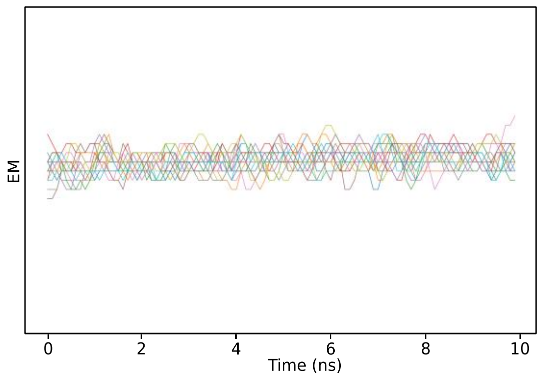


LEAKAGE MODEL

- Leakage depends on the previous state of the bit. Power consumption is caused by a bit flip.
- EM allows to get the direction of the transition: 0→1 or 1→0

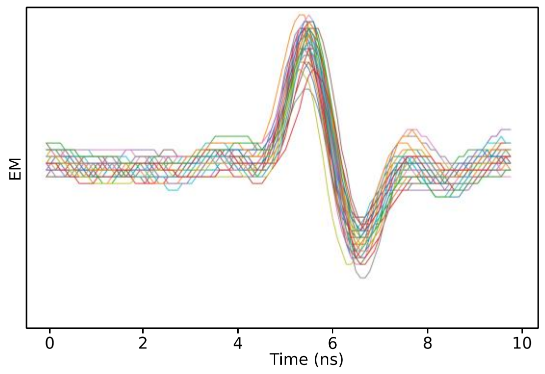


$0 \longrightarrow 1$



$0 \longrightarrow 0$

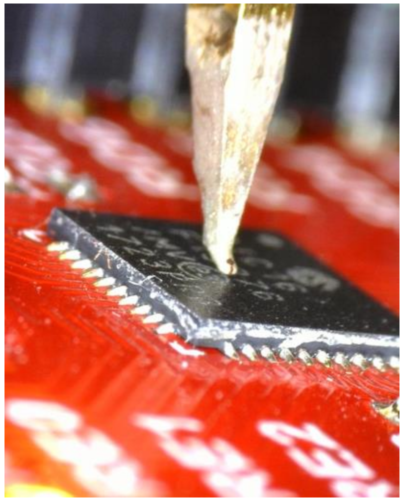
$1 \longrightarrow 1$



$1 \longrightarrow 0$

VIABILITY QUESTIONS…

1. Distinguish bit level variation (good enough SNR) ? ✓
2. Does each bit have its own leakage ? ✓
3. Does each bit leak independently ? ✓
4. What is the leakage model ? ✓



OUR METHODOLOGY

1. Build a 3 classes template ($0 \rightarrow 1$, $1 \rightarrow 0$, constant)

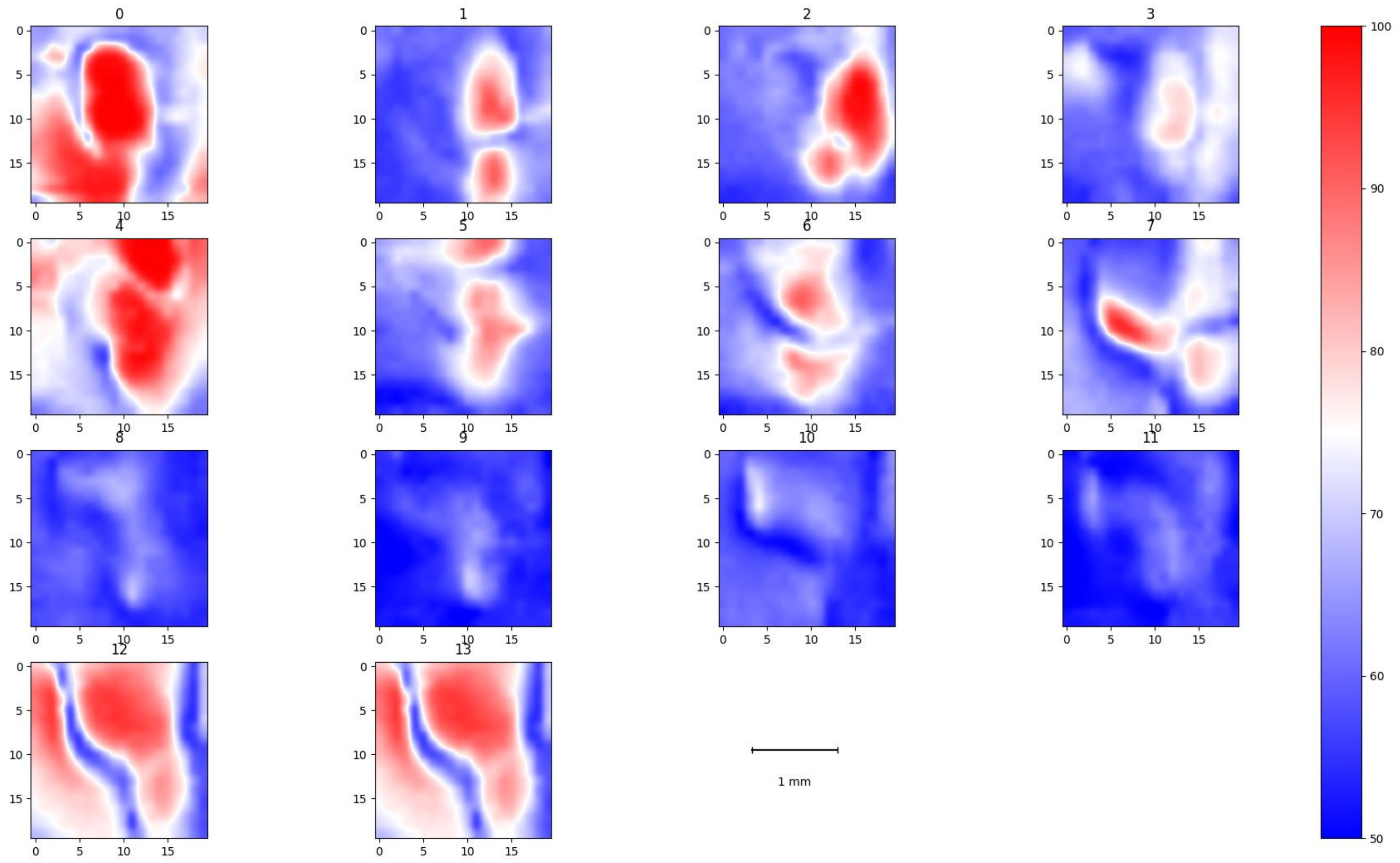
2. Apply it to get a sequence of transitions on your attack data

3. Convert it to a sequence of bits

4. Measure your success rate

5. Repeat for all bit at every probe position on a grid

CARTOGRAPHY FOR ALL BITS



Success rate

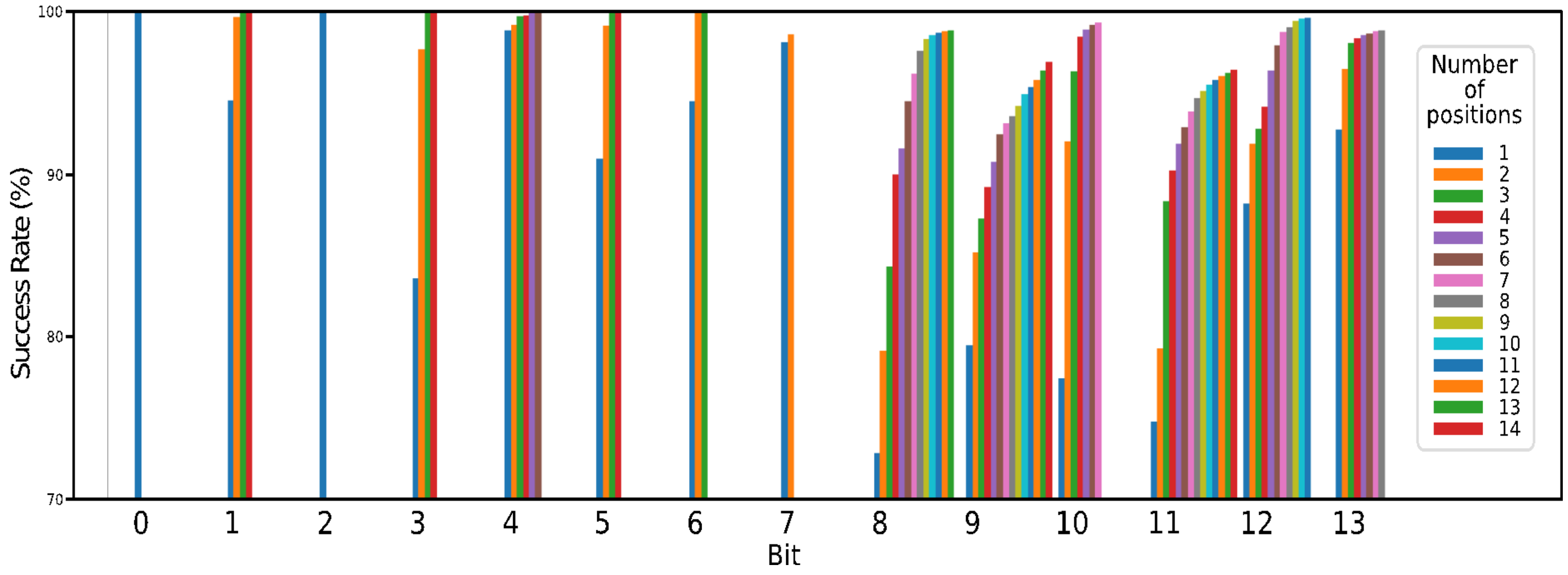
USE MULTIPLE POSITIONS ?

How to increase the success rate of the attack ?

- Combine the information from multiple probe positions !

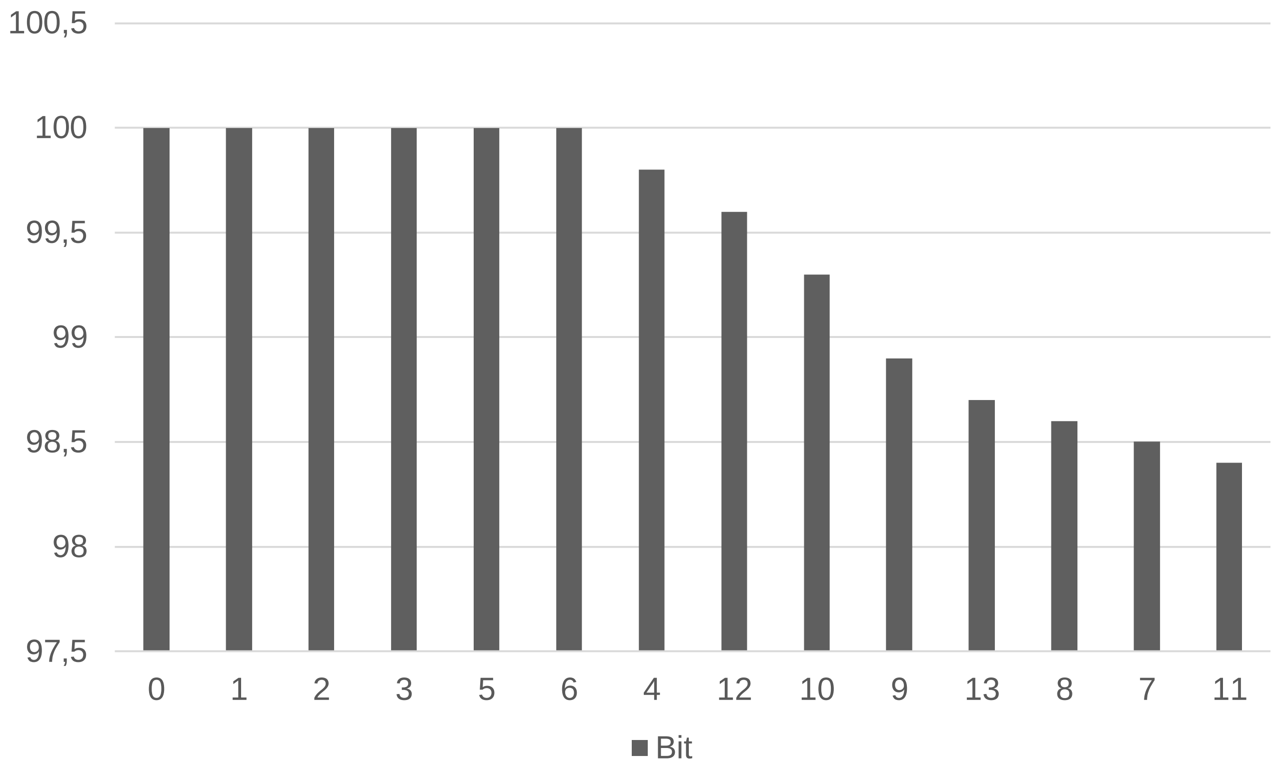
Concatenate the traces and apply the template attack as if it was a single trace.

USE MULTIPLE POSITIONS ?



- ➢ We selected a subset of up to 14 positions per bit
- ➢ Success rate converges to 100%

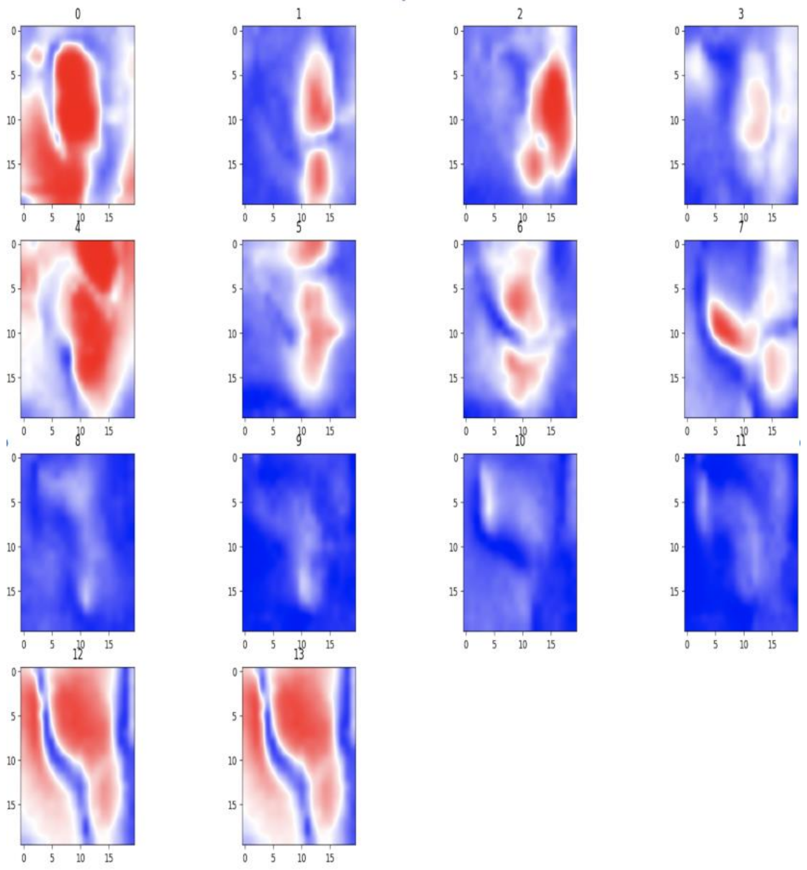
RESULT ON A RANDOM PROGRAM



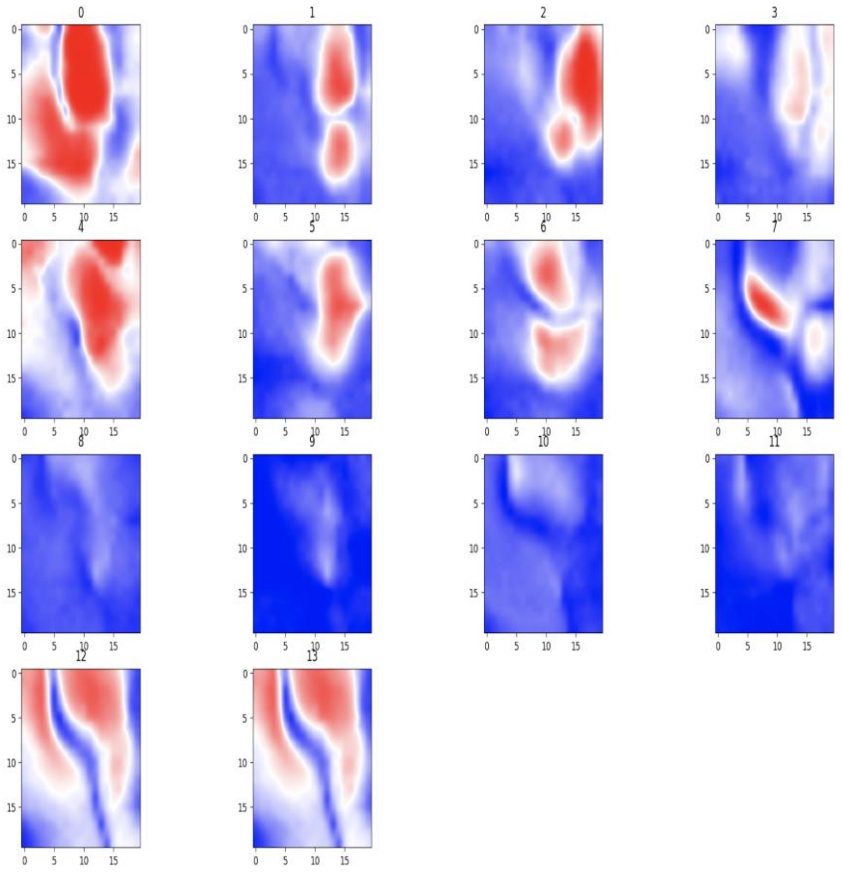
Average: 99.4%

➢ 95% of the instruction were recovered without any fault on the 14 bits !

TEMPLATE PORTABILITY ?



Target 1



Target 2

CONCLUSION

- ➢ Monobit approach
  - Easier to train
  - Potentially scalable
  - Gives usefull information even in case of error

- ➢ Exploit local leakage
  - Different leakage between the bits
  - Find the best probe positions for each bit
  - Combine information from multiple positions

- ➢ Our attack is portable between 2 targets

PERSPECTIVES

- Improve efficency using post traitement analysis
  - Find the closest real instruction
  - Use prior knowledge on instructions sequence probability

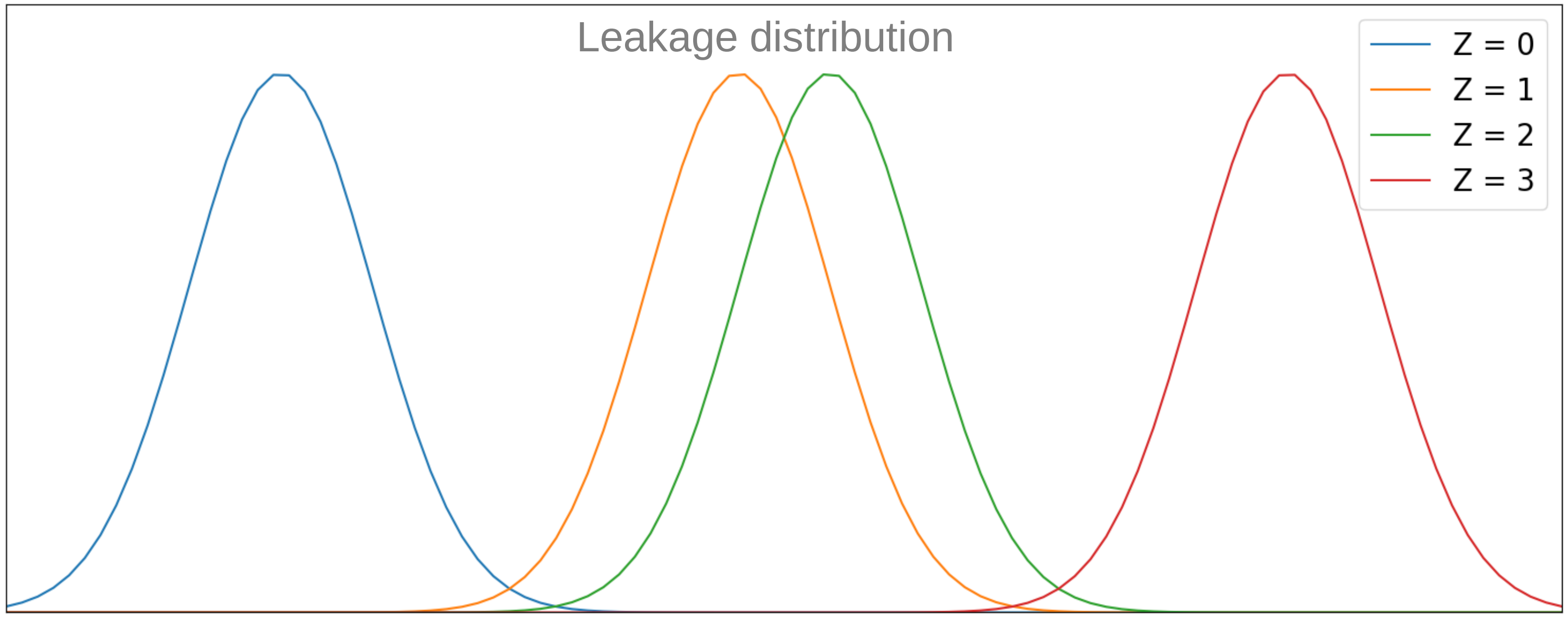
- Build a disassembler on more complex processors
  - 32 bits encoding
  - Deeper pipeline

Any questions ?

DIFFERENCES WITH KEY RECOVERING TEMPLATE

- Can not use the plaintext to add variability to the attacked variable

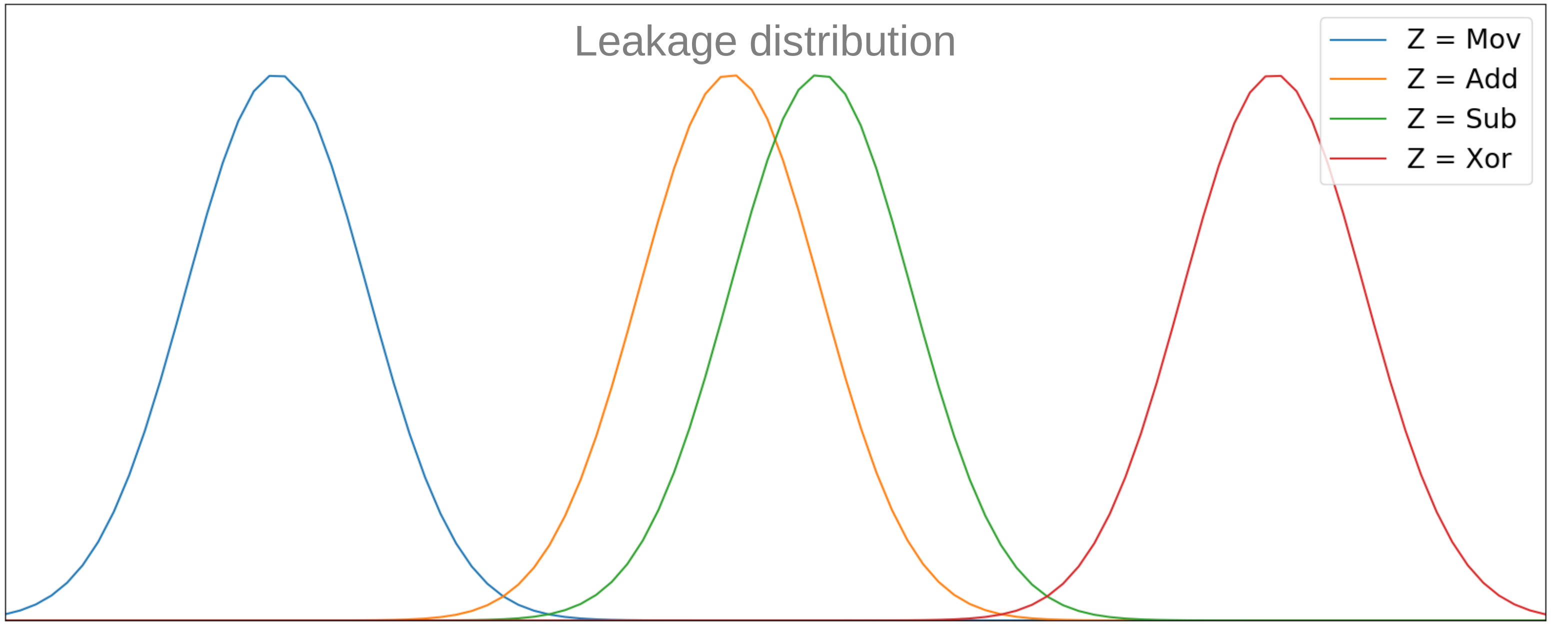
$$Z = Sbox(K \oplus P)$$



DIFFERENCES WITH KEY RECOVERING TEMPLATE

- With instructions template we are stuck in one gaussian (one class)

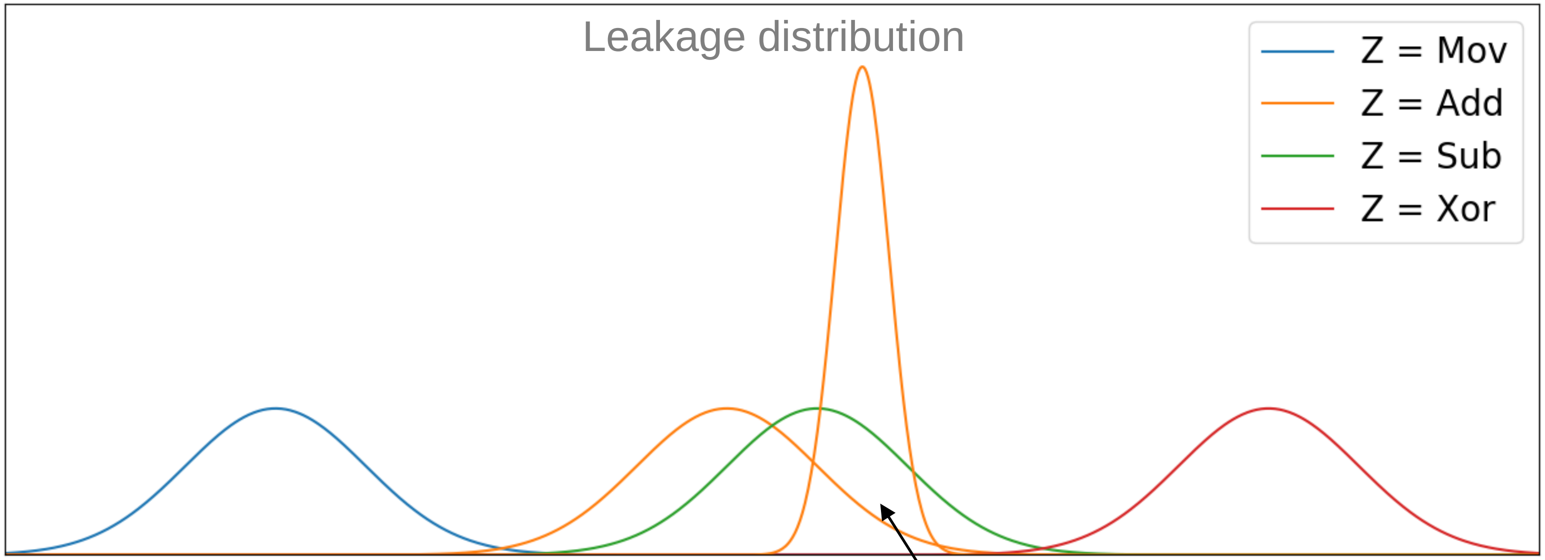
$$Z = instruction$$



DIFFERENCES WITH KEY RECOVERING TEMPLATE

- During the attack phase, operand are fixed …

$$Z = instruction$$



Attack phase distribution: Z = Add r0, r1, r2