

A practical approach to side channel attacks and fault injection

E. RICHARD - M. CIZDZIEL - C. CARDIN - A. BOURY-LE-STRAT

ENSIBS

Context - Objectives

Hardware attacks have become a critical concern as they bypass traditional software protections by directly exploiting the physical properties of embedded systems.

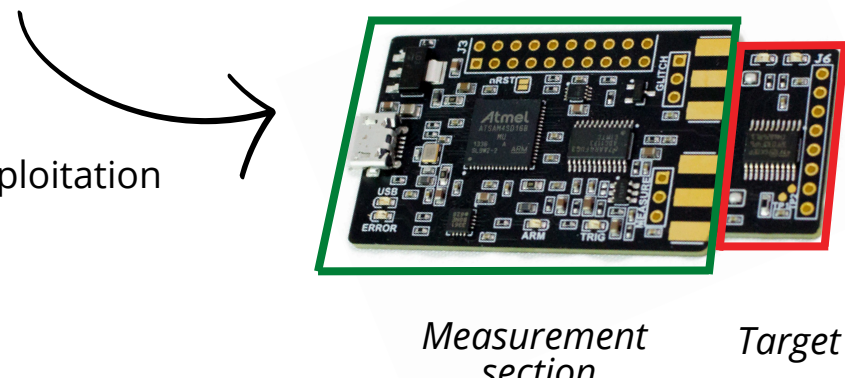
Across 10 hardware challenges 🛡️, CSAW Embedded Security Challenge provide an opportunity to discover and exploit two of these hardware attacks :

- Fault Injections (FI)
- Side Channel Attack (SCA)

To solve them, we a **ChipWhisperer Nano** with a STM32F0 target was the hardware imposed.

Main objectives are :

- **Learn** about FI and SCA
- **Apply practical** methods of hardware exploitation
- **Solve** as many challenges as possible



2 Fault Injection

Clock Glitching

Principle

Briefly disrupt the clock line to cause an execution anomaly :

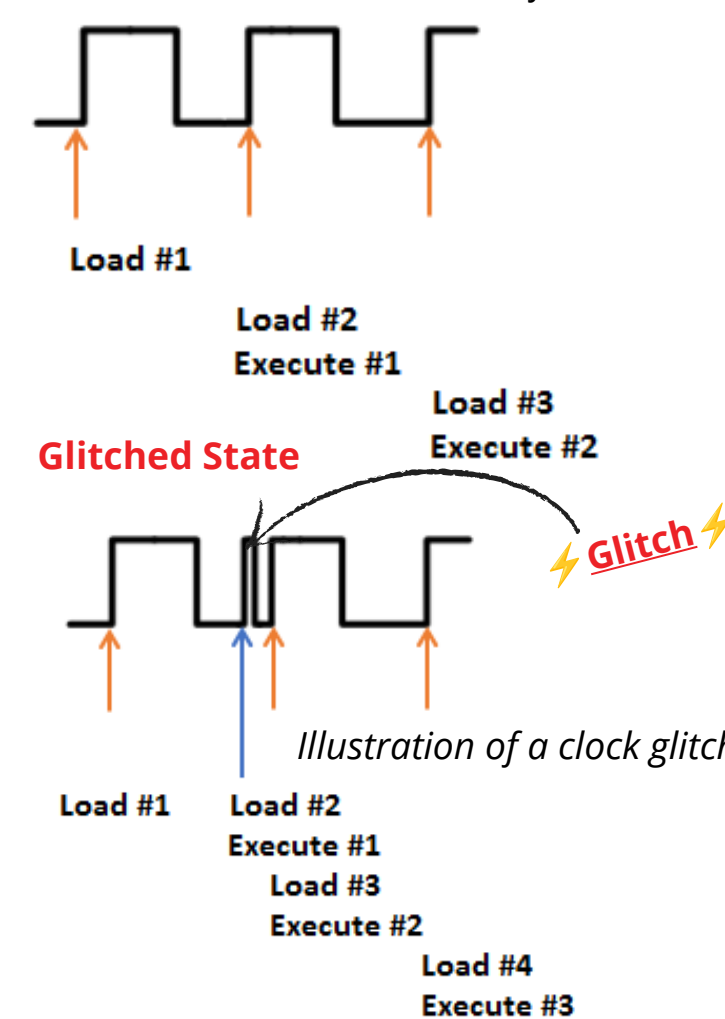
- ⚡ instruction skip
- ⚡ partial execution
- ⚡ desynchronisation

and create an unexpected internal state.

Countermeasures :

- Redundancy of operations
- Implementation of a fault counter
- Jitter injection

Normal State Illustration of a clock



Path of execution

1. Glitch the clock on an "cnt+=2" instruction
2. This instruction is not executed
3. The sum is invalid
4. The flag is printed

Critical Calculation

```
cnt = 0;
for(i = 0; i < 100; i++){
    for(j = 0; j < 40; j++){
        cnt += 2;
    }
}
```

Code from the Critical Calculation Challenge

```
cc1{C0RRUPT3D_C4LCUL4T10N}
```

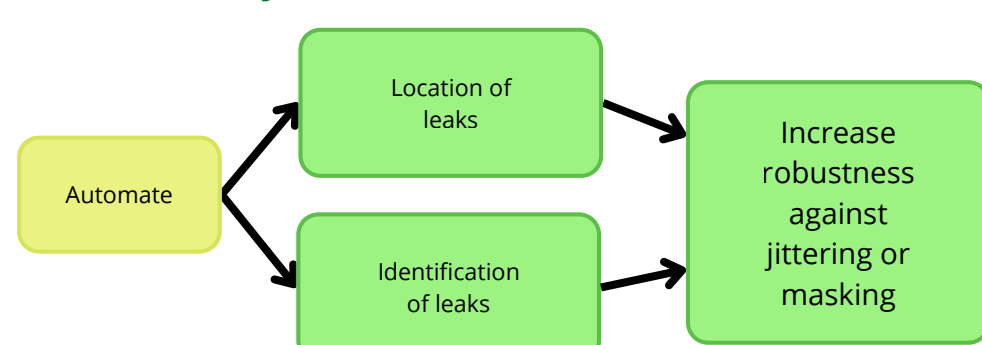
4 Usage of Artificial Intelligence

Principle :

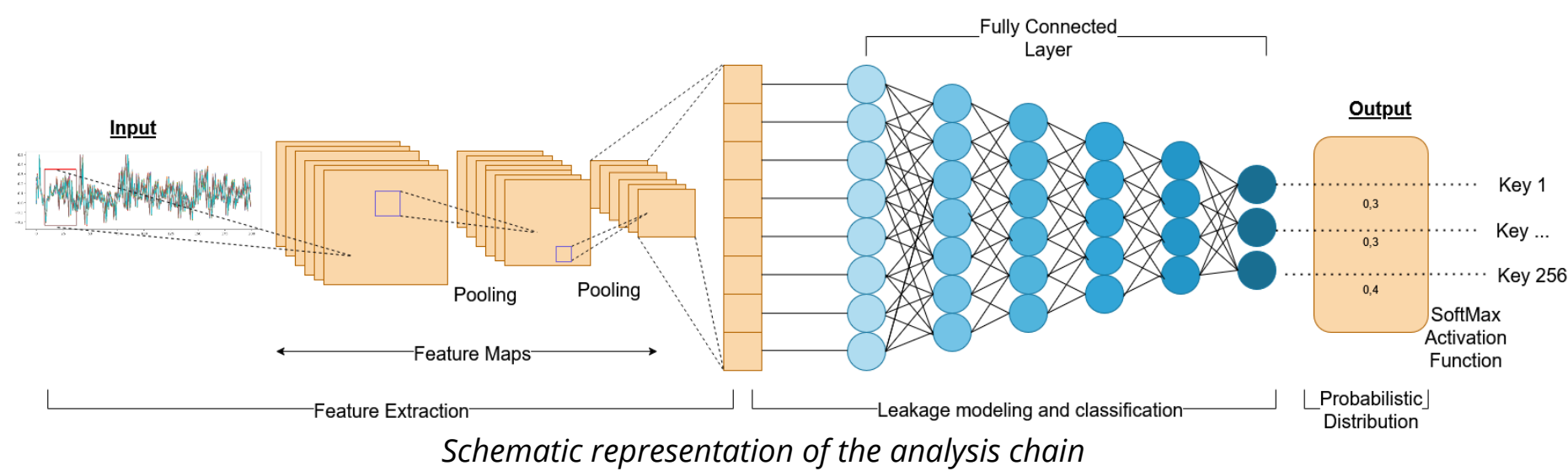
Usage of AI algorithm to automate power consumption traces analysis :

- Isolation Forest
- CNN
- Multiple Output Classification

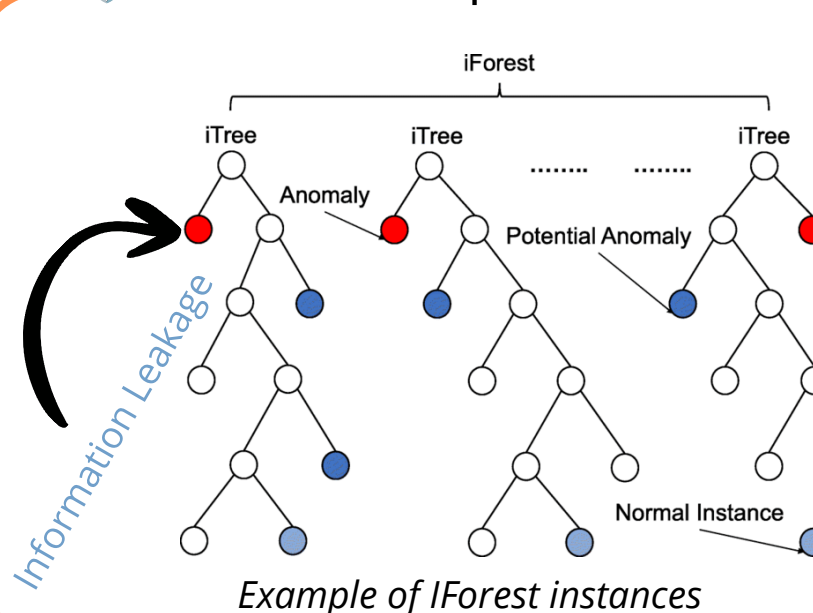
Objective:



Example of a complete analysis chain



DarkGateKeeper



IsolationForest:

An unsupervised anomaly detection algorithm that identifies atypical consumption or execution patterns within a large set of measurements that deviate from normal system behaviour:

- A trace detected as abnormal is considered a likely candidate for containing the key byte.
- Information leakage related to cryptographic operations often causes a subtle but measurable disturbance compared to ordinary traces.

1 Simple Power/Timing Analysis

Simple power analysis and timing analysis are both side channel attacks that take advantages of physical leakage of the DUT (Device Under Test).

Simple power analysis

Principle:

Directly observe the electricity consumption curve of a device while an algorithm is running to deduce the secret key bits.

Countermeasures :

- Avoid branches to secret elements
- Constant time algorithms

Link with Timing Analysis

- Timing Analysis exploits variations in execution time to obtain similar information.
- In both cases, the observable behaviour reveals a dependency between operations and secret data.

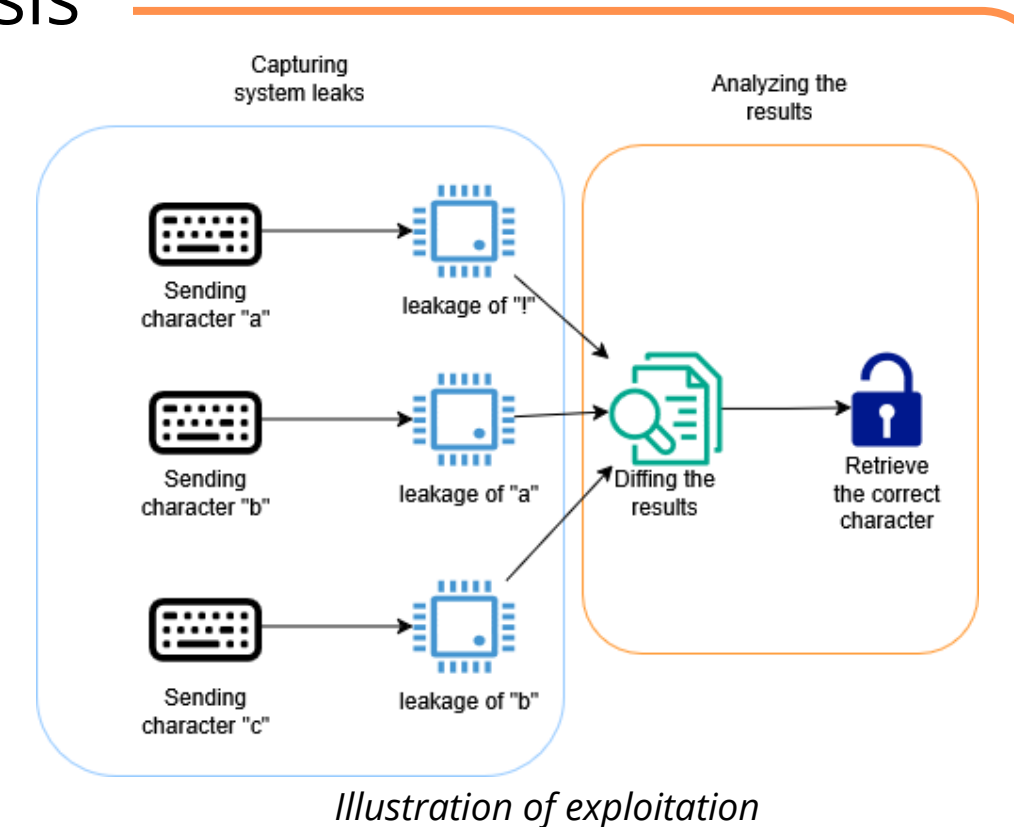


Illustration of exploitation

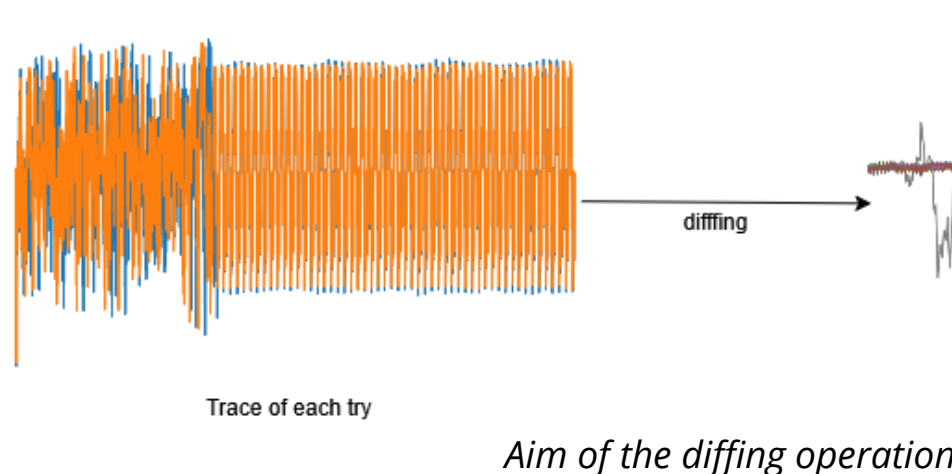
Diffing

We subtract the test trace from a reference trace (invalid character).

Goal: highlight significant variations.

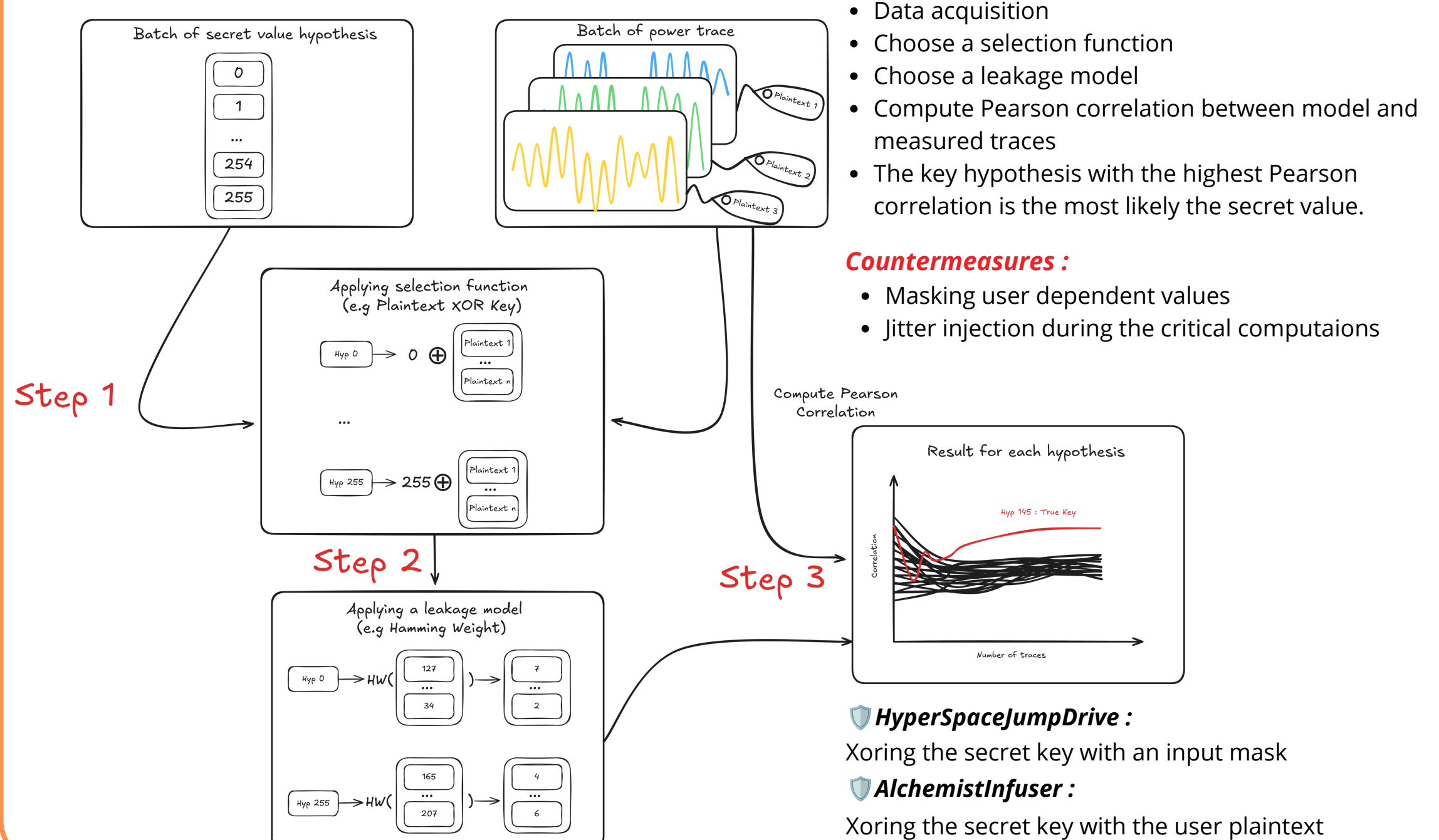
Gatekeeper

If the character matches, an intensive loop is launched → peak consumption. This energy difference allows us to identify matches.



3 Correlation Power Analysis

CPA workflow



5 Unintended solution

In both of the *sorterSong* challenges the *rand* function was used to generate private arrays; the goal of these challenges was to exploit a non-constant time implementation of sorting algorithm to retrieve these arrays.

At the begining of the source code, the *srand* function—who seed the RNG behind the *rand* function—is called with the argument : *time(NULL)*; the current time is the seed.

Principle:

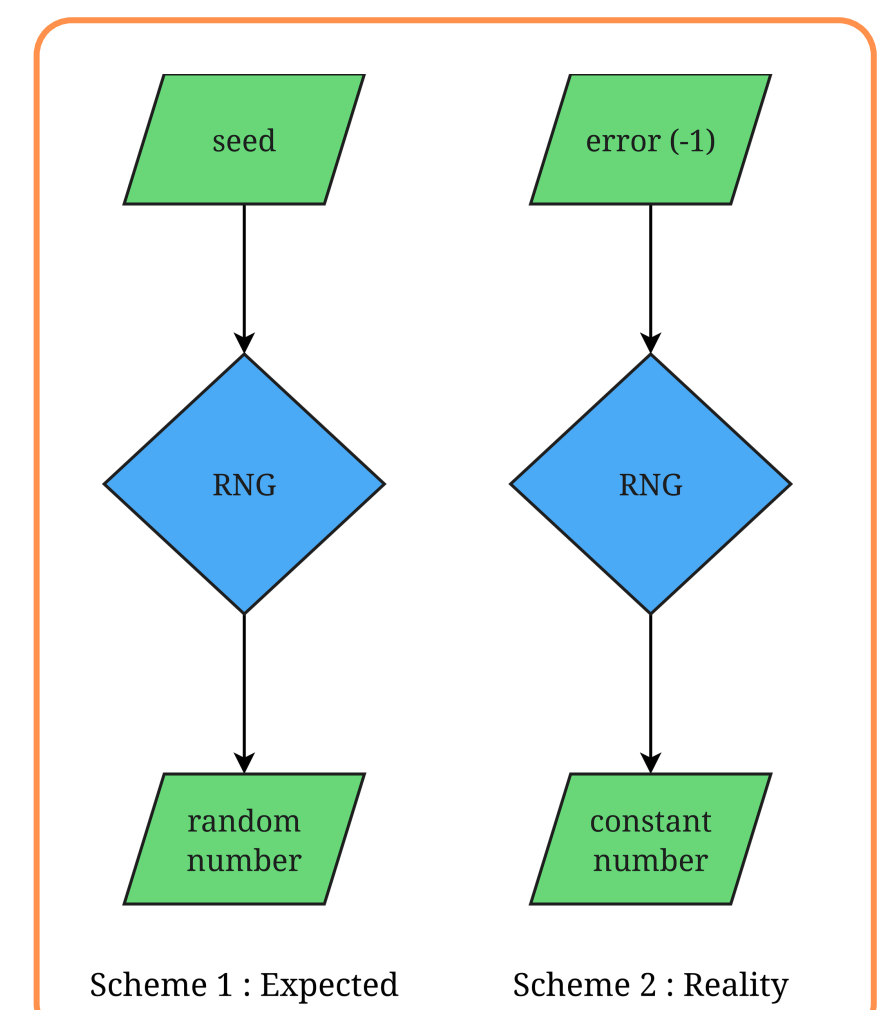
All of these usages—sorted by ascending severity—represent vulnerabilities:

- modulo operator to generate number in a range (introduce bias),
- the *rand* C function implement an *Linear Congruential Generator*, which is not cryptographically secure,
- the time is a poor source of entropy and should not be used as a seed (e.g. approx time within 1h reduce entropy to 11 bits),
- absence of real-clock in the ChipWhisperer; entropy is reduced to 0 bit because the time function fails (time function always return -1).

We exploit the last vulnerability by generating the private arrays on our side and sent them to the Chipwhisper. It works because the generation process was fully deterministic.

Countermeasures :

In this context (i.e. embeded device without any source of entropy) a non-volatile memory should be use in the RNG (e.g. as recommended by ANSSI in *ANSSI-PG-083*).

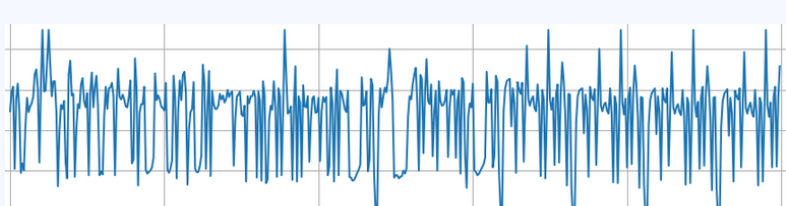


Our experience

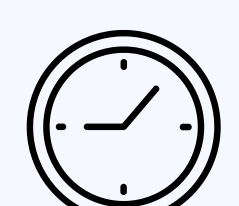
Two conclusion, one opportunity

1

It confirms the feasibility of extracting sensitive information and modifying the execution flow from a program with a low cost device, as a Nano ChipWhisperer and simple technics, for example :



Simple Power Analysis



CPU Clock glitch

Low cost-embedded devices are highly vulnerable

2

The limited capabilities of the ChipWhisperer-Nano's Analog-to-Digital Converter (ADC) constrains :

- Capturing long traces (ex : *GateKeeper 2*)
- Precision of fault injection based-attacks (ex: *DarkGateKeeper*)

Hardware capabilities is crucial in side-channel research

3

This work opens the way for future investigations into the use of machine learning techniques to improve :

- Correlation between traces and secret keys
- Detection of hardware vulnerabilities in code

This will enhance the efficiency and accuracy of next-generation attacks, and therefore strengthening defense strategies.