Fault Analysis Assisted by Simulation

Kais Chibani, Adrien Facon, Sylvain Guilley, Damien Marion, Yves Mathieu, Laurent Sauvage, Youssef Souissi, Sofiane Takarabt

Fault Analysis Assisted by Simulation

Kais Chibani[1], Adrien Facon[1,2,3], Sylvain Guilley[1,2,3], Damien Marion[1], Yves Mathieu[2], Laurent Sauvage[1,2], Youssef Souissi[1], and Sofiane Takarabt[1]

[1]Secure-IC S.A.S., 15 Rue Claude Chappe, Bât. B – ZAC des Champs Blancs, 35 510 Cesson-Sévigné, France
[2]LTCI, CNRS, Télécom ParisTech, Université Paris-Saclay, 75 014 Paris, France
[3]École Normale Supérieure, département d'informatique, CNRS, PSL Research University, 75 005 Paris, France

Abstract

Side-channel and fault injection attacks are renown techniques to extract keys from cryptographic devices. Fortunately, leakage reduction and fault detection countermeasures exist and can be implemented right in the source-code. However, source-code level countermeasures might be altered during the compilation process. Design simulation is an effective means to detect such harmful simplifications. This is a well-known methodology to analyze regressions in terms of side-channel leakage.

In this chapter, we explain that protections against fault injection attacks are no exception. First of all, we show that vulnerabilities to those attacks can be easily detected by simulation methods. Second, we highlight that simulation techniques are also highly efficient in detecting logic simplifications which destroy (fully or partly) the countermeasures. Thus, the simulation-based methodology we present in this chapter shows that it is possible to decide quickly which compilation options are safe and which ones are detrimental to the security.

1 Introduction

Embedded systems are based on hardware integrated circuits. Basically, any hardware design has its own conception life-cycle that starts with the algorithm and architecture specification. In fact, the designer starts by describing sequentially the functional part of his design based on High Description Language (HDL). Then, we distinguish three abstraction levels in the design life cycle as follows:

- **Register Transfer Level** (RTL). It consists in specifying the logical operations and data-flow between registers. This level involves an explicit clock to synchronize the data-flow (clock/event accurate).
- **Post synthesis level** or Gate level. It describes the timing properties of logical operations. In other words, it takes into consideration the delay propagation of signals within the circuit gates. Hence, this netlist level is technology dependent. Such netlist is generated by synthesis tools.

- **Place Route level** or layout level. It comes with placed and routed cells; and more timing information. In fact it takes into account the delay propagation into circuit's routing.

In the context of embedded security, the designer can conduct a security analysis hand-in-hand with the functional and timing verifications. This is very useful as the designer will be able to think about the security testing at an early stage. He will not wait until the tapeout of a testing chip to start a security evaluation of his implementation. Moreover, he will be able to conduct such evaluation by himself without the need of high skills in physical security analysis. In fact, recently, the world of Electronic Design Automation (EDA) arsenal of tools has came with a new tool, called VIRTUALYZR® [24, 22], that allows for such security analysis with seamless integration within the design life cycle as shown in fig. 1. In the sequel, all the presented results are obtained with such Pre-Silicon Assisted Verification Tool, that we denote by its acronym PAVT.

1.1 Security Verification Assisted by Simulation: PAVT Workflow

PAVT deals with both Side-Channel Attack (SCA) and Fault Injection Attack (FIA). The general workflow consists in two phases as illustrated in fig. 2.

1. **Simulation Phase.** During this phase the tool interacts with a HDL simulator to generate the so-called *virtual* activity of the design. The obtained virtual activity is an ideal image of the behaviour of the cryptographic design during its execution. Usually, several queries are needed to conduct SCA or FIA analyses. PAVT manages the automatic configuration of input vectors (*i.e., input parameters*) based on the testbench of the design. This testbench needs not be written specifically for the purpose of using PAVT. It can either be the unitary testbench used for functional verification, or a testbench generated automatically by EDA frameworks (such as Cadence Specman Elite). This phase allows for the generation of the fingerprint of the design activity. The fingerprint is just a dataset managed and organized by the database of the tool. At this point, the PAVT comes with two approaches to deal with obtained dataset. A first approach is called *real approach*. It consists in building one leakage trace from the overall activity of the design signals given one fixed input (e.g., one fixed key, one variable message for an AES implementation). This way, the tool generates a number of traces equal to the number of variable input messages. This approach is based on a high-level modeling of the electrical activity of the design. According to the literature [14], the basic model is composed of a dynamic factor reflecting the transistors activity and a static factor reflecting the activity of the circuit while at rest. Digitally, the dynamic factor can be computed through a toggle count and the static factor is just the actual binary value of the signal at each simulated instant. In order to generate one trace, the sum is performed over all signals (so-called Hamming weight leakage model). Now, a second approach called *ideal approach* can be envisioned and is very fruitful in terms of security analysis in practice. It consists in dealing with the raw state of simulation without the need of any power consumption model. In fact, this wire-level approach allows detecting any security anomalies regarding each signal in the design. This approach is more complete and requires managing matrices-based traces. Besides, for both approaches, the analysis is always conducted in the best conditions as the designer here does not care about the impact of real factors like the measurement noise and configuration of real equipment like oscilloscopes and pulses generators that require more skills and processing. Regarding fault(s) injection, erroneous values are forced from the simulation tool. The resulting traces are also termed "virtual".

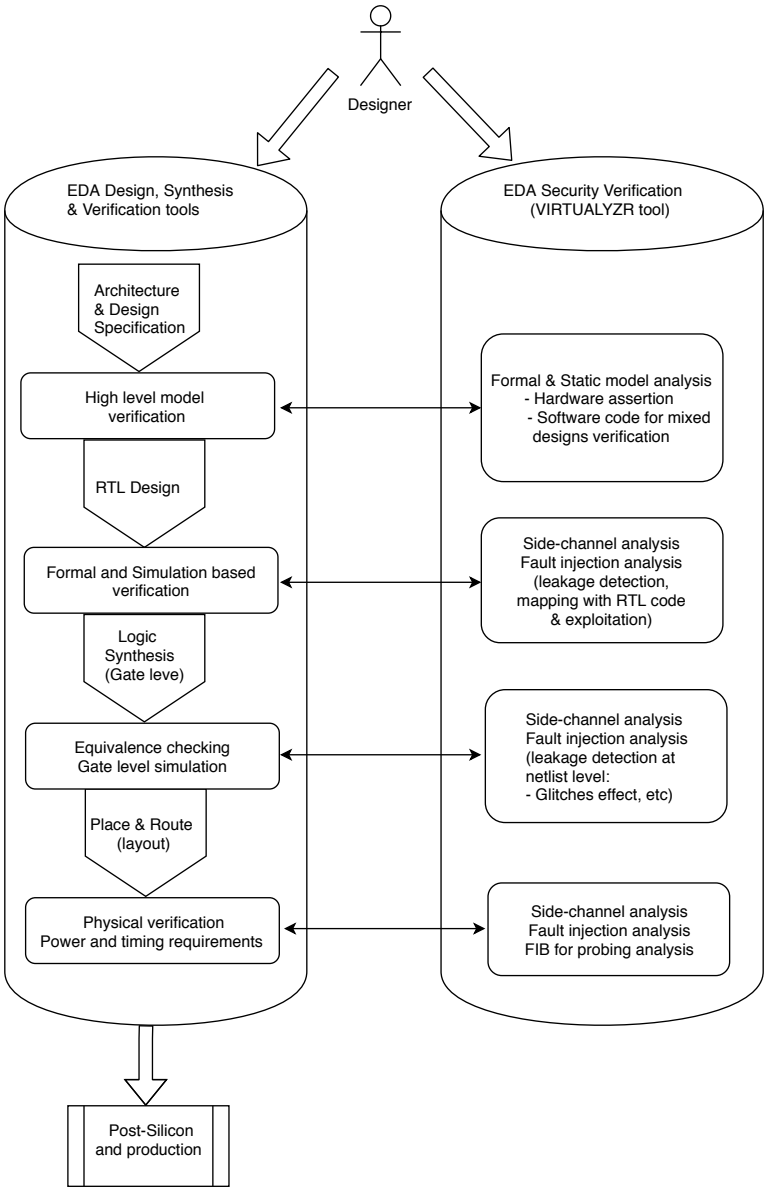

Figure 1: Seamless security verification integration in the design life-cycle workflow.

2. **Analysis Phase.** This phase consists in analyzing the obtained virtual trace for SCA or faulty log trace for FIA. It is noteworthy that the PAVT is not a tool designed for attackers, but a tool to the attention of designers. The tool allows a security analysis checkpoint at all the design levels. The main goal is to look after any potential leakage and then to map it with the leaking signals and therefore the HDL code itself. The tool allows going further by

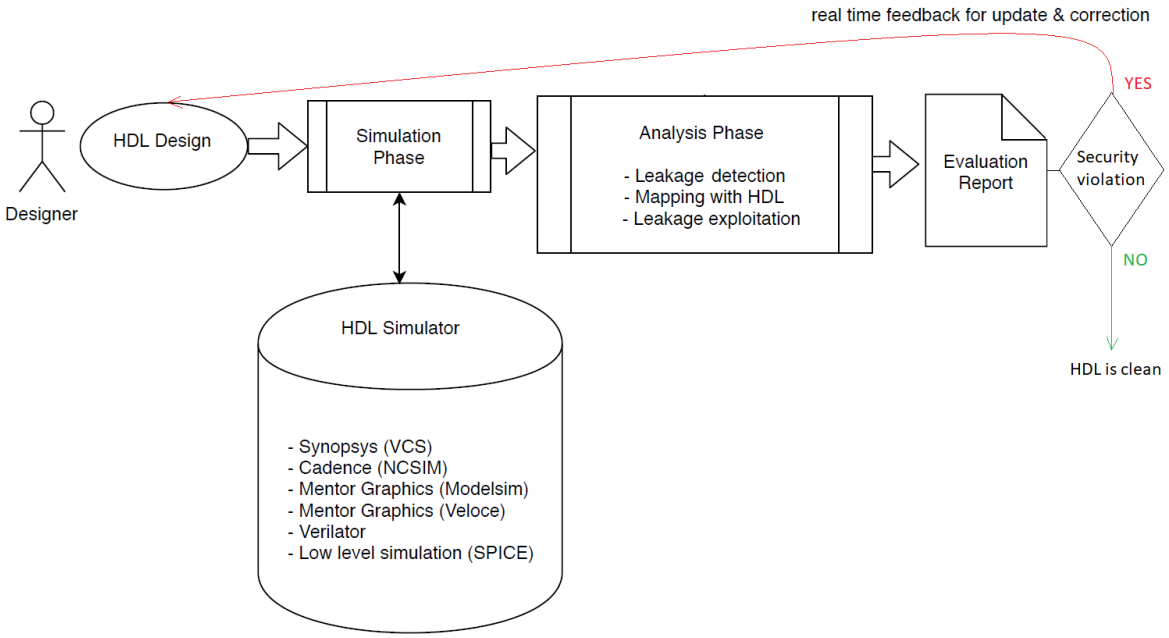

Figure 2: PAVT: security analysis assisted by simulation general workflow (see algorithm 1).

exploiting the found leakage and assess whether such leakage can lead to the recovery of some sensitive variables (*e.g., secret key*). The tool comes with ready-to-use use-cases to deal with the most commonly used and recent analysis such as Linear Regression Analysis (LRA) and Machine-Learning (ML) based analyses.

Both phases are processed in an iterative manner until the HDL design is clean with no security violations (see algorithm 1 and fig. 2). The fig. 3 shows deeper details about the internal workflow of the PAVT. First, it extracts input design structure. Then, after having properly configured the user project, the tool runs a couple of simulations, which will produce a database of raw results. the PAVT workflow distinguishes raw results generation and dataset generation as two separate operations. First, simulation results are computed and stored on a hard disk. Once simulation results are available, it is possible to use them to generate datasets with different properties. A dataset is associated with a given consumption model (for real approach) and a set of probed signals used for trace generation. To reflect this distinction, the tool uses the twin concepts of *target* and *probe*:

- A project has only one *target*. It is the set of signals for which we store simulation results.
- A project can handle multiple *probes*. A probe is useful to study the activity of a sub-group of signals. Probing is useful to focus on sub-modules, to spot leaking signals, to simulate a cartography, etc.

The same notions naturally apply to fault injection simulation campaigns. The user interaction with tool is limited to the following actions in order to achieve a complete end-to-end security

Algorithm 1 Security verification and refinement of the HDL design by PAVT (see fig. 2)

$HDL \leftarrow initial\ HDL\ design$
while HDL is not clean **do**
  Simulation phase
  Analysis phase
  Evaluation report generation
  if Security violation **then**
    $HDL \leftarrow fix\ the\ HDL\ design$
  else
    return HDL
  end if
end while

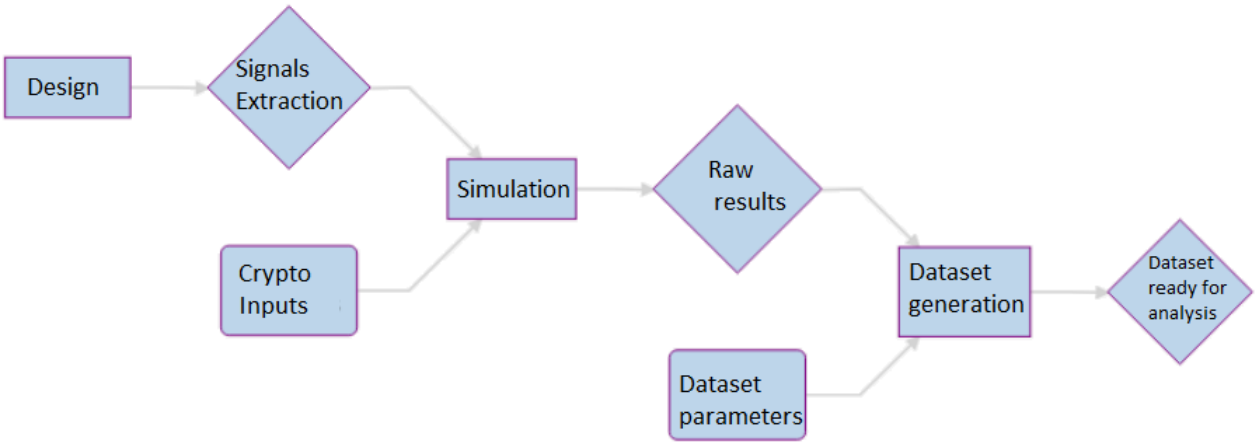

Figure 3: PAVT internal workflow.

verification:

1. Create and manage a project by providing the design testbench, the RTL design (for RTL analysis), the netlist with its Standard Delay Format (SDF) file. The user is required to provide a compilation script (usually in TCL language) to automate the interaction with his Intellectual Property (IP) or System-on-Chip (SoC);
2. Define the simulator environment: simulator model, execution mode;
3. Define the input design to evaluate: hardware IP, software on SoC;
4. Extract design information;
5. Define simulation parameters, depending on the evaluation mode;
6. Instrument hardware simulators;
7. Retrieve, process, and store simulation results;
8. Configure and generate virtual consumption traces for a batch of simulations;
9. Execute external analysis modules on the datasets;
10. Ask for security analysis report generation.

The PAVT allows a full scripting of those actions in order to make an easy interaction with functional verification flow. A snapshot example is given in the fig. 4.

2 Side-Channel Analysis Study Case

2.1 Canright Architecture Overview

Canright proposed in 2005 [6] a very compact implementation of the AES Substitution Box (S-Box). The proposed implementation is based on the usage of sub-field (also called "tower field") for the S-Box computation. He showed that his version is about 20% smaller than the initial version of the state of the art [20]. The Canright computation is described as follows:

Definition 1 *S-Box computation for an input byte a, we compute two steps:*

1. *Inverse, let* $c = \begin{cases} a^{-1}, & \text{if } a \neq 0 \\ 0, & \text{otherwise} \end{cases}$ *the multiplicative inverse in* $GF(2^8)$.
2. *Affine Transformation, the output* $s = Mc \oplus b$, *with the constant bit matrix* M *and the byte* b *as follows:*

$$\begin{bmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{1}$$

```
## creation
delete project vtz-test-1
new project sca as vtz-test-1

## simulator
set simulator local modelsim

## input design
set workspace ip VERILOG_AES_unprotected COMPILE/User_compile_RTL.tcl TESTBENCH/SIC_Trusted_aes_c128_tb.v
extract

## simulation parameters

# target
target add SIC_Trusted_aes_c128_tb

# parameters
new parameter busy_tb as busy on risingedge
new parameter clk_tb as clock
new parameter data_in_tb as input
new parameter key_tb as input
new parameter data_out_tb as output

# batches
set parameter data_in_tb as list list10000 for batch SCA
set parameter key_tb as fixed 5102BE4D39C0C82925A356AF50CF71D1 for batch SCA
set simulation count 1000 for batch SCA

# options
enable results database
enable auto delete

## dataset generation parameters
new probe top on SIC_Trusted_aes_c128_tb
set dataset sampling 1ns

## simulate
simulate

## analyze
set consumption model static
generate dataset as static
notify dataset static to AES_CPA_ACC as sca-stat.tra
exec python AES_CPA_ACC/analyze.py sca-stat.tra 100 5 0 0
```

Figure 4: PAVT automation script snapshot (VIRTUALYZR® language).

Table 1: Number of signals of the implementation: RTL and PS.

	RTL	PS
Number of wire	45888	79936
Studied number of time samples (AES last round)	7	2000
Initial number of time samples	28	9868

In a second paper [7] on the topic of AES S-Box, Canright proposed a protected version as a countermeasure against SCAs. The countermeasure is a so-called first-order masking [4].

2.2 Side-Channel Leakage Detection

Masking is considered to be the most effective countermeasure against SCA as it aims at hiding the sensitive data when manipulated by some cryptographic algorithm during its execution. Canright showed in his paper [7] how to compute the non-linear part of the S-Box (*i.e.*, the Galois inverse) with a perfect masking [4]. Theoretically, such countermeasure should be robust, at least assuming the gates evaluate in the adequate order, *i.e.*, only once all inputs have arrived. However, in practice, when masking is implemented and mapped to technology, there might be some information leak due to *glitches* phenomena. Glitches can be observed through the switches of a combinational signal when a new event occurs (input changes). In fact, when the signal bounces, the sensitive data become visible and therefore vulnerable to basic SCA such as first-order SCA (e.g., Correlation Power Analysis, or CPA). For more details about glitches phenomena, we refer the reader to [17]. In the literature, Canright compact architecture [16] and glitches [18] have been analyzed separately from the SCA viewpoint. Of course, failure of Canright design to resist first-order attacks points out a first-order leakage. To our best knowledge, the precise glitch impact on Canright masked AES version has not been studied yet.

In the following we propose to study such effect based on a pre-silicon analysis and putting forward the tools to conduct such analysis. Pre-silicon analysis, or security assisted by simulation, has several advantages as the evaluation is carried out in the best conditions. Actually, the analysis is fully white-box as we have a full access to all the signals making up the design. The actual leakage contribution of each signal is studied. Therefore, any glitch effect shall be easily observable. The tool that we have used for this purpose is the PAVT [21, 8, 24]. It allows detecting leakages at all levels of the design flow, namely the RTL, Post Synthesis (PS) and layout levels respectively. In the sequel, we put forward the methodology and the results we obtained on Canright masked AES-128. (We note that our AES was clocked at 20 MHz.) We denote by $X^{D,R}$ a trace provided by the PAVT, with D the number time values and R the number of wires. We choose a binary representation: $\forall d < D,\ \forall r < R,\ X_{d,r} \in \{0,1\}$, we have $X_{d,r}$ be the value of the wire r at the time sample d. By extension, we denote $X^{D,R,Q}$ of set of Q traces. One example of the $X^{D,R}$ trace matrix is shown in fig. 5. One challenging task to consider is the size of analyzed data. As a matter of fact, the table 1 shows the number of signals to study at both RTL and Post Synthesis (PS) design levels. The technology considered for PS netlist level is a Xilinx Field Programmable Gates Array (FPGA).

Generally speaking, the refinement flow goes hand-in-hand with optimizations made by the synthesizer, such as simplifications, redundancy optimizations (logic factoring), mutualization, but also ressources duplication (under performance stress), etc. However, in the present example (refer

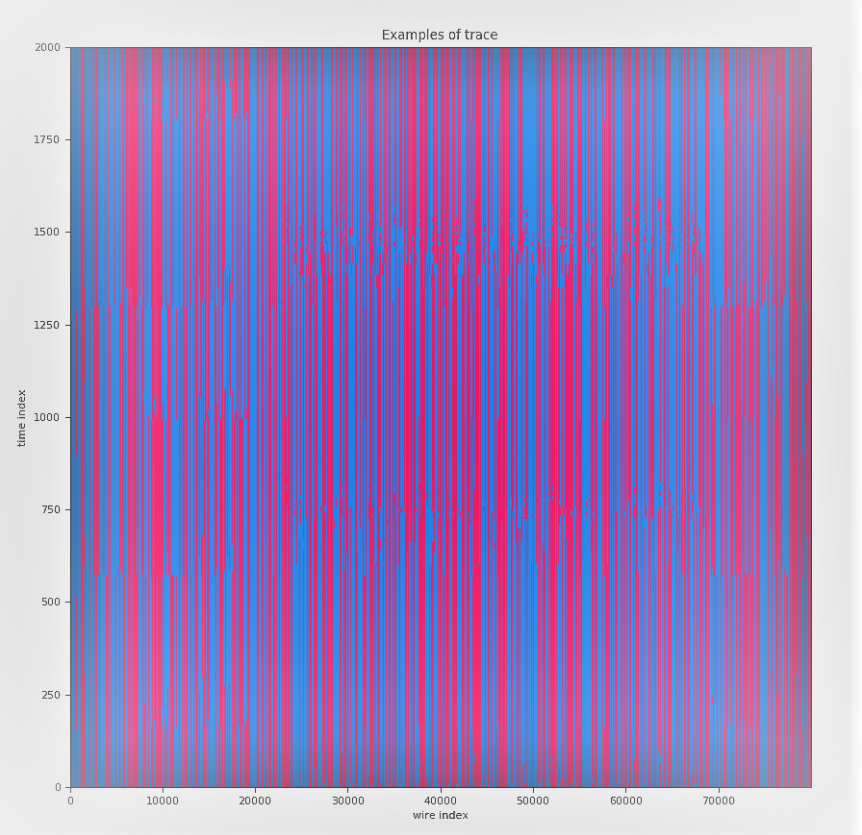

Figure 5: Virtual matricial trace as generated by the PAVT

to Tab. 1), it is noteworthy that the deeper the design level (RTL→PS), the more signals, and the bigger the dataset (trace). Therefore, the more complex the analysis. This pecularity can be explained by the fact that all the combinatorial functions are described in RTL as tables (hence only signals are the input and the output of the tables), which are synthesized (from RTL to PS) in complex combinational logic, which leads to a significant increase of the numbers of signals. In order to manage the complexity of the analysis at PS level (owing to the greater number of signals), we have performed some dimentionality reduction on the initial dataset, by removing all useless constant and redundant toggle activities [15]. We only analyze samples (we call a sample a pair composed of a time value and a register index) that change at least once over all the set of trace, that we call an event. In other words, we reduced the dataset size without losing any useful information for leakage detection. In term of analysis, we chose the Correlation Power Analysis (CPA) as basic approach [5]. Notice that the CPA is the *optimal distinguisher* when the affine link between leakage and model is unknown [11]. For the sake of convenience, we studied the most commonly used selection function for AES analysis, that is the input bit values of the S-Box at the last round. We define $Y(k)^{B,Q}$ the matrix of size $B \times Q$ (with $B = 8$, the number of bits in a byte) as:

$$Y(k)_{b,q} = \left(\left(\text{S-Box}^{-1}\left(k \oplus \text{cipher}\right) \gg b\right) \&\texttt{0x1}\right)$$

where $b < 8$ and $q < Q$. We study each sample (composed of a time and a wire index) independently, by computing all the following correlations:

$$\left\{ \frac{\text{cov}(X^Q_{d,r}, Y(k)^Q_b)}{\sigma(X^Q_{d,r})\sigma(Y(k)^Q_b)} \right\}_{\substack{r<R \\ d<D \\ k<256 \\ b<8}} .$$

2.2.1 RTL Level Analysis

In fig. 6 we display for each byte of key the $S = 16$ differential traces computed on 1K traces. With same notations, a differential trace can be expressed as:

$$\text{diff\_Traces}^{S,D} = \max_R(\mathcal{D}(X^{D,R,Q}, Y^{S,Q})) \ , \tag{2}$$

where $\mathcal{D}$ is the distinguisher (for instance the Pearson coefficient as for CPA).

In fig. 6, the red curves are relative to the good key hypothesis correlation. Clearly this key cannot be distinguished from the rest of key hypotheses (*i.e.*, the bad ones) even when the number of traces is highly increased. Logically, we conclude that the first order analysis is not successful: The RTL level is safe and Canright masked AES is not leaking at this stage as expected. How about Post Synthesis level and glitches effect? This is what we are going to investigate in the next subsection.

2.2.2 PS Level Analysis

We have generated a post synthesis netlist based on Xilinx FPGA technology. The goal was to see the real effect of glitches on our target implementation (Canright masked AES). Now, the PAVT takes as input the netlist and the SDF file that describes the delay propagation within the circuit gates. In fig. 7 we can see the analyzed clock periods and the time events with their probability of occurrence. An example of trace is displayed in fig. 8.

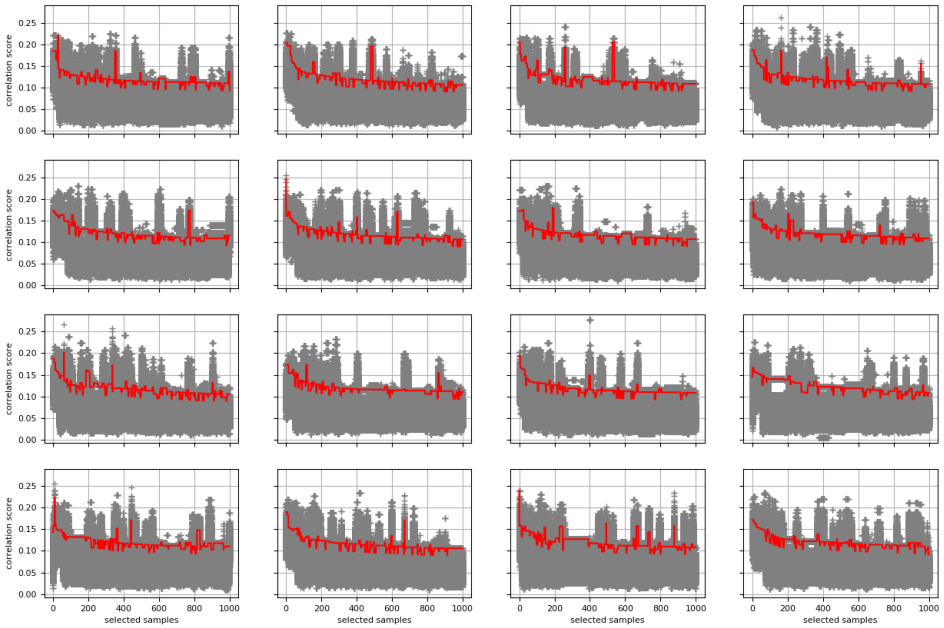

Figure 6: Result of the CPA on RTL level: not successful.

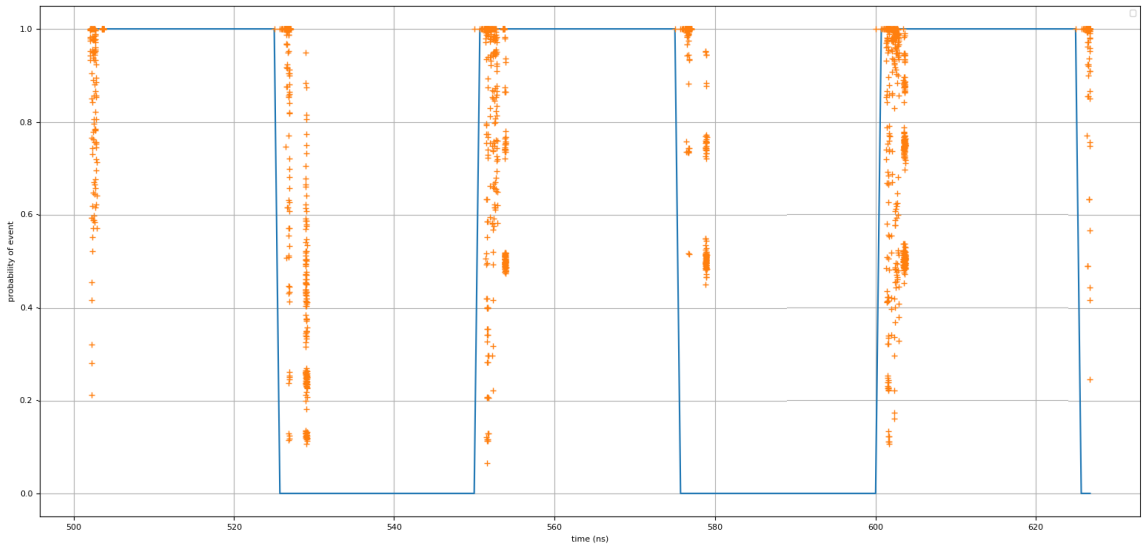

Figure 7: All events probability occurrences with the clock signal in blue.

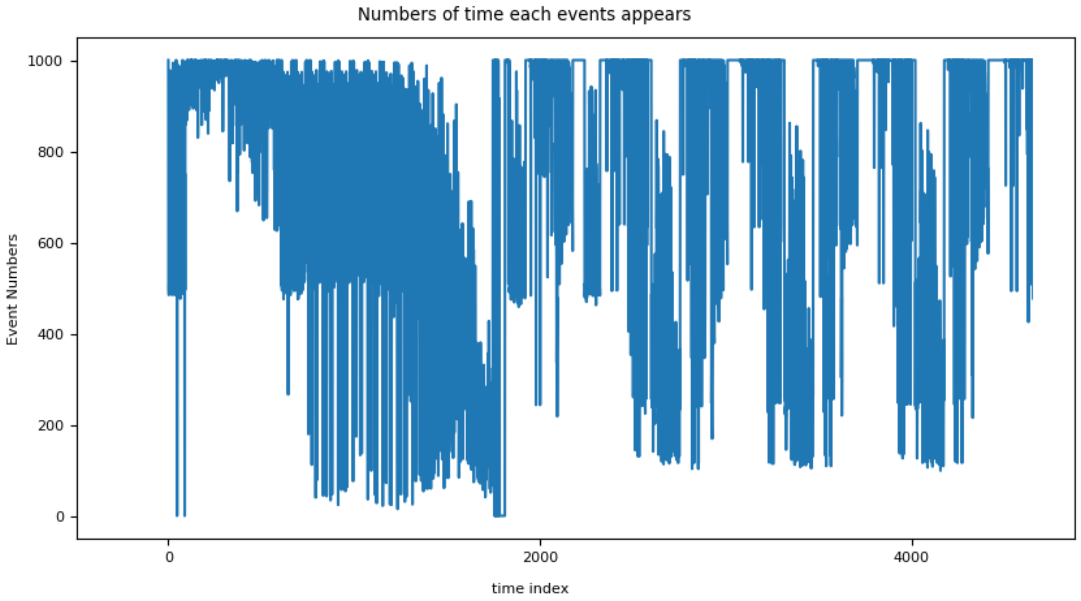

Figure 8: Virtual trace at PS netlist level illustration

Obviously, those switching events have a significant probability of occurrence and this requires more investigation in term of security analysis. For this purpose, we run a CPA with the same tool, the PAVT, and realized that the secret key is easily recovered (the success rate is 100%) with about 250 traces. We remind that the CPA is performed at the bit-level, meaning that each bit involved in the target node (input of the last round of the AES) is processed and analyzed separately. The purpose of this exhaustive methodology is to investigate the contribution of each bit and therefore the effect of the glitches on each signal, so as to be able to pinpoint to the designer the root cause of the leakage problem.

We note also that at the PS level, the time line could be different between two executions (fig. 8), since combinational delays are not synchronized from trace to trace. We denote by $\mathcal{D}_q$ the values of the time events of the trace q. This way, according to our notations, we have $\mathcal{D}_Q = \bigcup_{q=1}^{Q} \mathcal{D}_q$. In fig. 10, we focus only on the good key hypothesis and display its differential CPA traces for all the key bytes (16 bytes) and for all the bits. With only 250 traces, it is clear that a significant leakage shows up (in terms of correlation amplitude value) at the PS netlist level. In fact, the higher the value of correlation is, the higher the leakage is. We were able to identify a set of 7 significantly leaking key bytes, namely those at position $\{0, 2, 5, 6, 7, 11, 14\}$. This automatically exhaustive analysis is performed with the PAVT. The tool provides us with the leakage instants and the set of leaking wires in the design. This way, we obtained a direct mapping between the identified side-channel leakage and the leaking signals and therefore the HDL line of code (instance). In table 2, we provide some numerical information about the number of leaking samples and wires respectively.

In the fig. 10 we display in red the samples that leak the most ($Correlation > 0.6$). The leaking

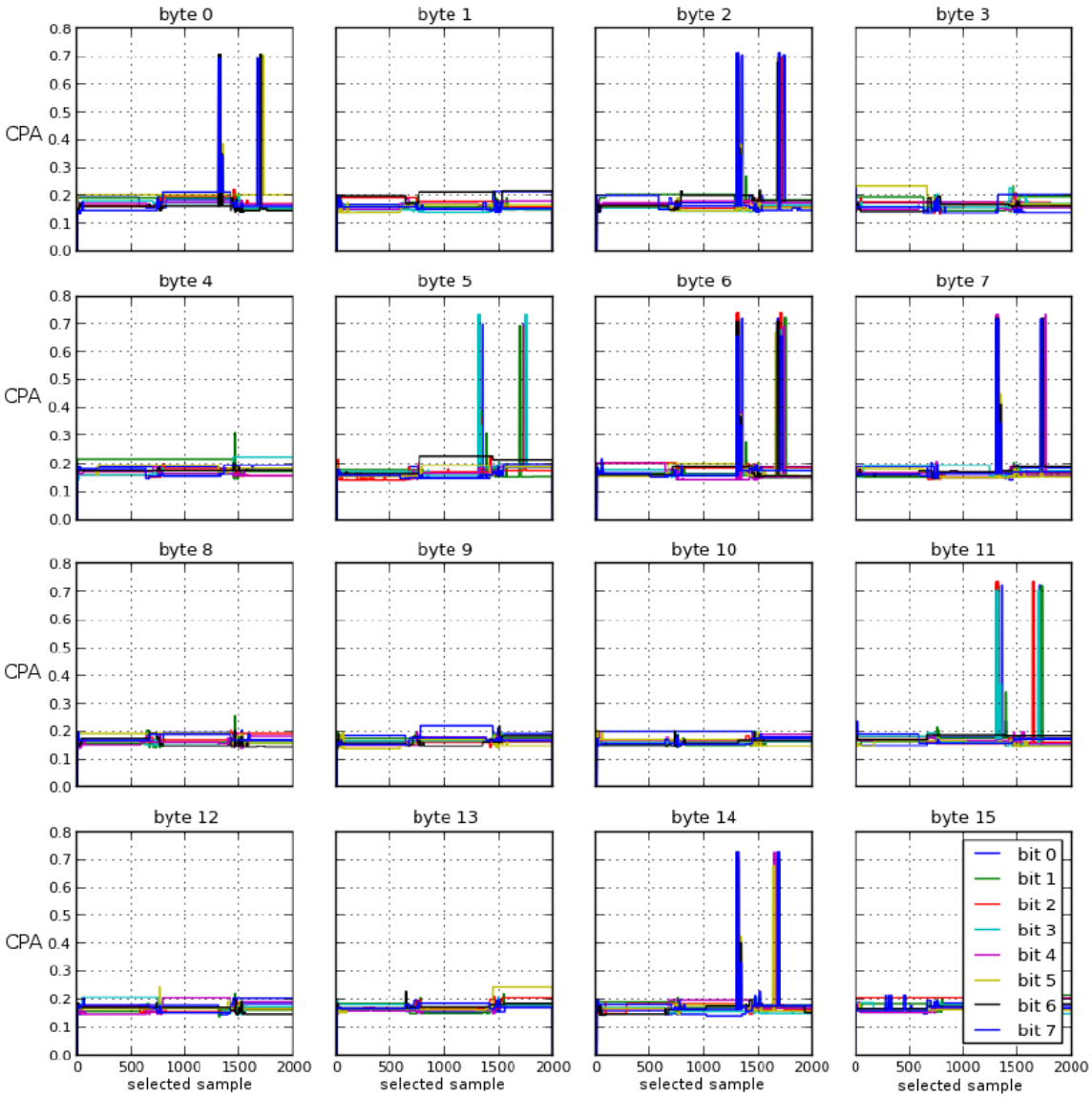

Figure 9: Good key differential traces at PS level with bit-level precision.

Table 2: Statistics on leaking wires, as generated by PAVT.

key byte	number of leaking samples	number of leaking wires
0	68	76
2	74	171
5	62	95
6	106	171
7	56	76
11	74	95
14	52	76

events are glitches with the probabilities noted in the y-axis.

Definition 2 (glitch) *If a wire switches at least twice before stabilizing during a clock period, this event is termed a* glitch*.*

3 Fault Analysis Assisted by Simulation

Fault attacks are active attacks, which need an adversary to induce errors into the target device, using some tampering means. This tampering can be accomplished in several ways, as extensively discussed in literature [12]. In general, tampering means or fault injection techniques are classified in two broad categories, i.e., *global* and *local*. Global fault injections [10] are, in general, low-cost techniques which create disturbances on global parameters like voltage, clock, temperature, etc. The resultant faults are more or less random in nature and the adversary might need several injections, to find required faults. On the other hand, local techniques (e.g., clock glitch, optical/electromagnetic injections, body bias injection [1]) are more precise in terms of fault location and model. However, this precision comes at the expense of costly and bespoke[1] equipments. The kind of injected fault can be defined as fault model. The fault model has two important parameters, namely *location* and *impact*. Location means the spatial and temporal location of fault injection during the execution of target algorithm. Depending on the type and precision of the technique, location can be at the level of bit, variable or random. Coming to the impact of fault, it is the effect on the target data. Commonly known fault injection impacts on target data can cause stuck-at, bit-flip, random-byte, or uniformly distributed random value.

3.1 Simulation-based Fault Injection Evaluation

With the PAVT tool, it has become possible to model the effects of several fault injection attacks, even those that require advanced technical skills and equipment as stated above. It is also possible to accurately analyze the intrinsic robustness of a digital circuit against such attacks early in the design flow. Such approach becomes very useful knowing that fault injection analysis are very hard (in practice) to mount on real targets. In the following, we detail two different use-cases related to local fault injection attacks on a hardware implementation of an unprotected AES 128 bit, which needs 10 clock cycles to perform an encryption.

[1] In Common Criteria parliance.

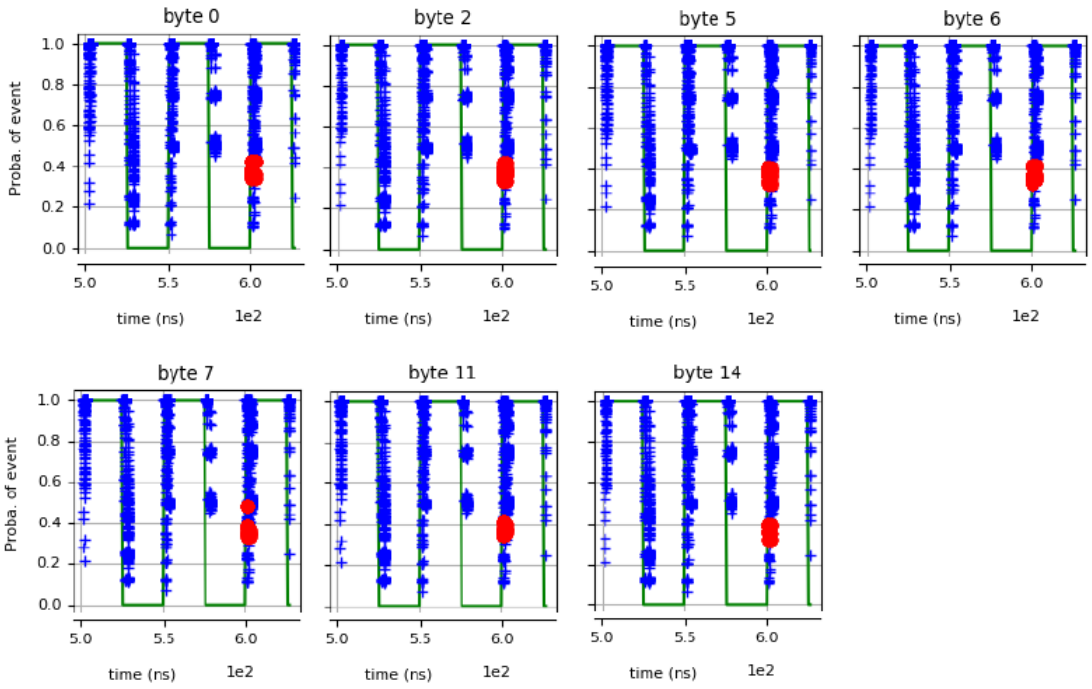

Figure 10: Glitched events probability (blue); Highly leaking glitches (red).

3.1.1 Clock-Glitch Injection

The principle of the Clock Glitch injection consists in precisely modifying the period of one or more clock cycles of the target design during AES execution. When the modified clock period is much shorter than what is expected in the normal clock, it shall create setup violation faults [23]. These faults can be exploited to retrieve the secret key. Since the modification of the clock frequency at RTL level is meaningless, we can perform clock glitch injections only with gate-level descriptions (i.e., post-synthesis level or place and route level). To this end, we synthesized the AES core to the gate level using ASIC 65 nm CMOS technology for 1.2 V supply voltage. After that, we configure the PAVT to perform clock glitch on a specific cycle during gate-level simulations to take into account the circuit delays (e.g., Standard Delay Format file). The configuration consists in defining some parameters needed to set the stimuli for simulations and the clock glitch parameters, in particular, the cycle target and the glitch duration. In our case, the main configuration was as follows:

- Target cycle: last round of the AES execution;
- Glitches duration: from 4 ns to 7 ns with steps of 100 ps;
- Number of simulations: 310.

Fig. 11 shows a cartographic view of the effects of clock glitches in terms of erroneous bits observed in the final output or otherwise the ciphertext. Based on such information, the evaluator can easily identify the maximal glitch duration that would lead to a final output error for a given cycle.

Simulation results can be used to apply a set of Differential Fault Analyses (DFA) which exploit differences between correct and faulty outputs to recover the key. The PAVT offers a set of DFA metrics which allow to analyze fault injection results. One example is the AES-128 DFA NUEVA (Non-Uniform Error Value Analysis) metric [13] which measures the uniformity of error values injected before the last SubBytes operation in order to find the key. Another example is the AES-128 DFA using Giraud metric [9]: This fault analysis requires single-bit faults at the input of the last SubBytes operation. As shown in Fig. 12, the PAVT is able to recover the entire key with only 126 simulations using DFA of Giraud. A few more simulations are required to perform the full analysis with the NUEVA technique.

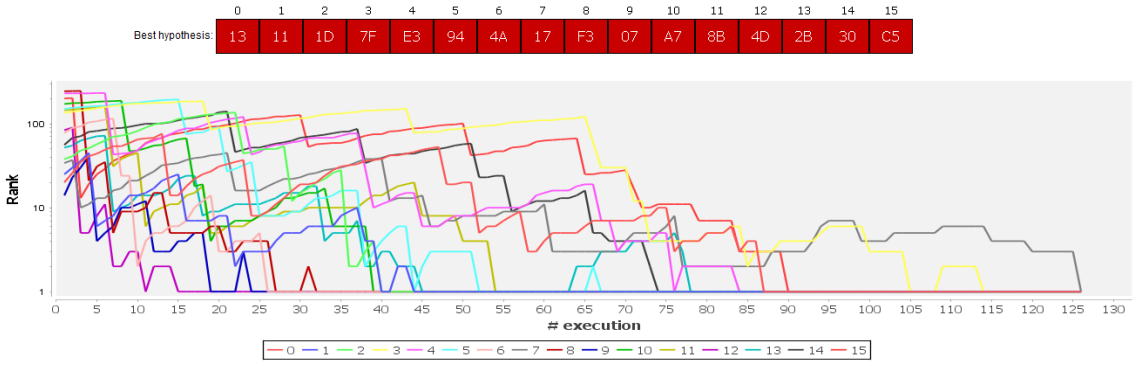

Figure 12: Analysis of clock-glitch injection results using DFA AES-128 Giraud metric.

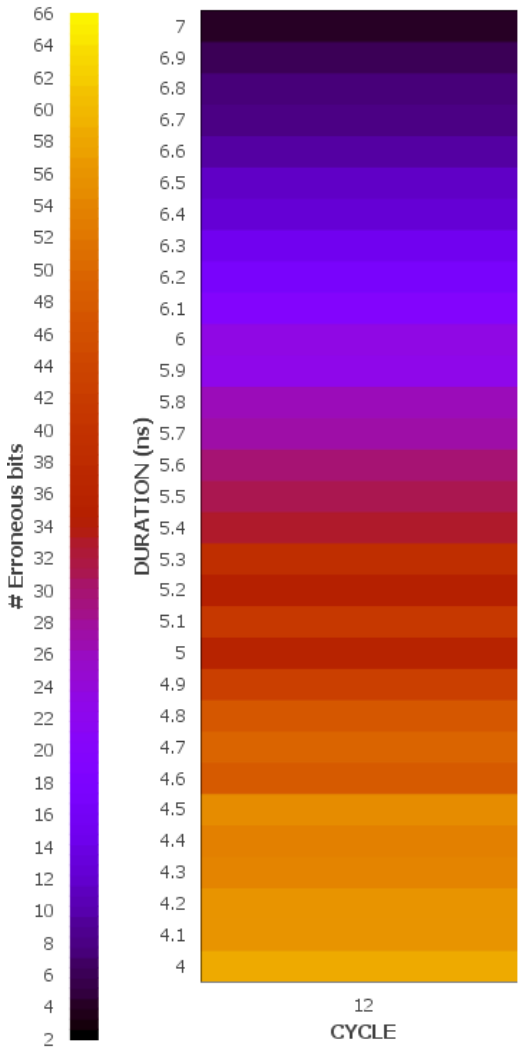

Figure 11: Erroneous bits according to the glitch duration.

3.1.2 Laser Injection

Laser fault injection falls into optical fault injection methods which consist in exposing the device to an intense light for a brief period of time. The injection can be performed either through the front-side or the backside of the target chip. Laser attacks can be used to inject faults characterized by high locality and timing accuracy. In the PAVT, the laser injections can be modeled not only at gate-level but also at functional level (i.e., Register Transfer Level) by configuring parameters such as the fault type (e.g., permanent/transient), the fault model (e.g., bit-flip, bit-set, bit-reset,

stuck-at-0/1), the fault location (e.g., wires, registers) and the fault time. When the time event occurs and the fault injection conditions are met, it becomes the fault time, and the fault model is injected into the fault location during simulation.

For this use-case, we have performed our analysis at RTL level with the following configuration: Fault time (last round cycle of the AES execution), Fault location (inputs of the SubBytes module), Fault model (bit-flip model), Number of simulations (100). The DFA results we obtained show that all key bytes are broken with only 10 simulations. Fig. 13 illustrates the results of the analyses completed using a DFA metrics already presented in the previous section. We can see that all the key bytes are broken at the end of a few simulations, in this case 10 with the DFA based on Giraud metric.

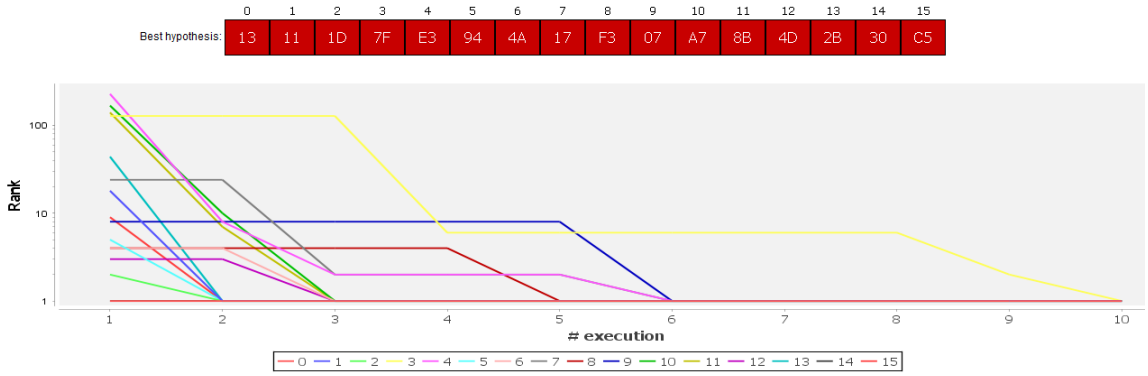

Figure 13: Analysis of laser injection results using DFA AES-128 Giraud metric.

3.2 Case Study: Netlist Level Leakage Fault Detection

As shown in the previous section 3.1, the attacks based on malicious injection of faults can degrade seriously the security of a cryptosystem. Faults injected into the cryptographic modules during the encryption (or decryption) operation will very likely result in a number of errors in the encrypted/decrypted data. Such faults must be detected before their spread to avoid the transmission and use of incorrect data. Fault detection techniques represent therefore a possible countermeasure against fault injection attacks and a desirable property for preventing malicious attacks, aimed at extracting sensitive information from the device, like the secret key.

For the AES block cipher, two main approaches have been proposed for achieving fault detection. The first one is based on temporal or spatial redundancy; in temporal redundancy, the same hardware is used to repeat the same process twice using the same input data. This technique uses minimum hardware overhead. Yet, it entails time overhead. In spatial redundancy, two copies of the hardware are used concurrently to perform the same computation on the same data. After each computation, the results are compared and every difference is reported as a fault. The advantage of this technique is that it can detect all kinds of faults. However, it requires an important hardware overhead. The second approach is concurrent error detection using Error Detecting Codes

(EDCs). It employs circuit-level coding techniques, e.g, parity schemes, modular redundancy, etc., to produce and verify results after each computation.

From a security point of view, designers have to verify the effectiveness of a given implemented countermeasure and be sure that it prevents against fault analysis. Remark that all countermeasures detect faults only to some extent (*e.g.*, up to a certain order, that is to say, up to a certain bit-wise multiplicity). For this purpose, we present our results based on the countermeasure presented by Bertoni et al. [2] which targets the datapath of the AES encryption module. This countermeasure uses a 4×4 parity matrix. Each bit corresponds to the byte state, and at each round the matrix is predicted and then can be compared with the computed one from the state. This countermeasure can detect all single errors and perhaps all odd errors and furthermore actually locate them. The hardware overhead is less than many other countermeasures (e.g., [3]) where a computation redundancy is required (2 times overhead). We designed an AES-128 encryption module implementing this countermeasure for the datapath. The control unit is also protected by computing the parity of the rounds counter. Then, we perform several simulation-based fault injection campaigns at the Register Transfer Level (RTL) in order to evaluate the fault coverage of the proposed parity-based EDC scheme. One hundred thousand injections are performed using plaintexts and keys selected randomly. The fault model is a single bit-flip at the last round of the encryption operation. The obtained results show that the detection rate is equal to 100 percent as shown in [2]. Then, we launch the logic synthesis on a Virtex-V Xilinx FPGA as technology target in order to perform the same fault injection campaigns but at Post-synthesis level (PS) (i.e., the post-map netlist is used during simulations). As expected, the detection rate is equal to 100 percent.

Thereafter, we re-synthesize the same RTL code but with different logic synthesis options to optimize the logic and to improve timing and design performances. As a matter of fact, the Xilinx Synthesis Technology (XST) synthesis tool allows designers to configure several options and properties that are taken into account during the synthesis process. These options target possible optimizations for area, speed or power consumption.

Switch Name	Property Name	
-ol	Placer Effort Level	High
-xe	Placer Extra Effort	None
-t	Starting Placer Cost Table (1-100)	1
-logic_opt	Combinatorial Logic Optimization	✖
-register_duplication	Register Duplication	Off
-global_opt	Global Optimization	Power
-equivalent_register_removal	Equivalent Register Removal	✖
-x	Ignore User Timing Constraints	☐
-ntd	Timing Mode	Performance Evaluation
-u	Trim Unconnected Signals	✖
-ignore_keep_hierarchy	Allow Logic Optimization Across Hierarchy	☐
-cm	Optimization Strategy (Cover Mode)	Area
-detail	Generate Detailed MAP Report	☐
-ir	Use RLOC Constraints	Yes
-pr	Pack I/O Registers/Latches into IOBs	Off
-c	Maximum Compression	☐
-lc	LUT Combining	Auto
-bp	Map Slice Logic into Unused Block RAMs	☐
-power	Power Reduction	☒
-activityfile	Power Activity File	
-mt	Enable Multi-Threading	Off
	Other Map Command Line Options	

Figure 14: Extract from the XST synthesis options for Xilinx FPGAs.

Fig. 14 is an extract from the Xilinx synthesis settings dialog box. In our case, we activate some options to optimize the design such as the $-logic\_opt$ option which optimizes timing-critical connections through restructuring and resynthesis, followed by incremental placement and incremental timing analysis. Previous injection campaigns are performed based on the obtained netlist. However, results are not the same because the detection rate decreases from 100 percent to 18.75 percent. More precisely, only faults injected on the AES control unit are detected. All faults into the datapath are no longer detected due the synthesis tool optimizations, as shown in Fig. 15.

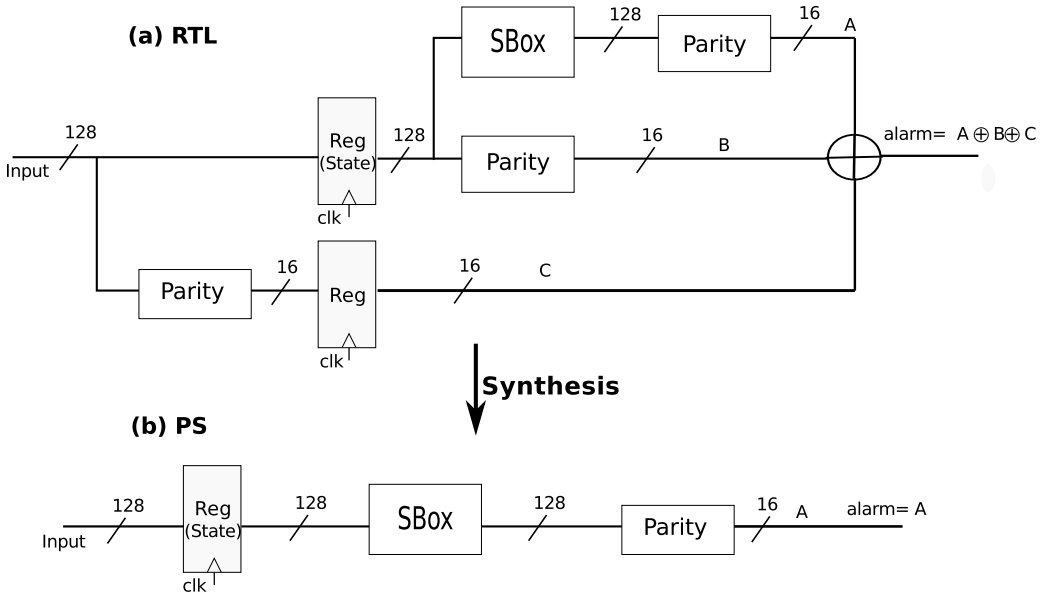

Figure 15: Total simplification of fault detection logic upon synthesis.

The countermeasure logic on the datapath was completely removed after the logical synthesis to optimize the design for area by reducing the total amount of logic used for design implementation. With obviously less gates, an equivalent functionality is obtained, albeit with a lesser security. Indeed, the S-Box is left unprotected, simply because the synthesizer has been smart enough to eliminate some combinational schemes considered to be equivalent. Functionally speaking, there is no alteration. However, from a security standpoint, the complete SubBytes transformation is left unprotected.

For the optimization prevention of signal B in Fig. 15, we use the `DONT_TOUCH` attribute. This attribute prevents optimizations where signals are either optimized or absorbed into logic blocks. It instructs the synthesis tool to keep the signal it was placed on, and that signal is placed in the netlist. Logic synthesis and fault injections are remade with the same options used during the previous experimentation. Results indicate that the detection rate increases from 18.75 percent to 56.43 percent. Indeed, the synthesis tool has simplified partially the fault detection logic as shown in Fig. 16 by eliminating the combinational block producing C signal. Consequently, only faults injected on the state register are detected, which opens a large door for successful fault injection attacks within the combinational logic.

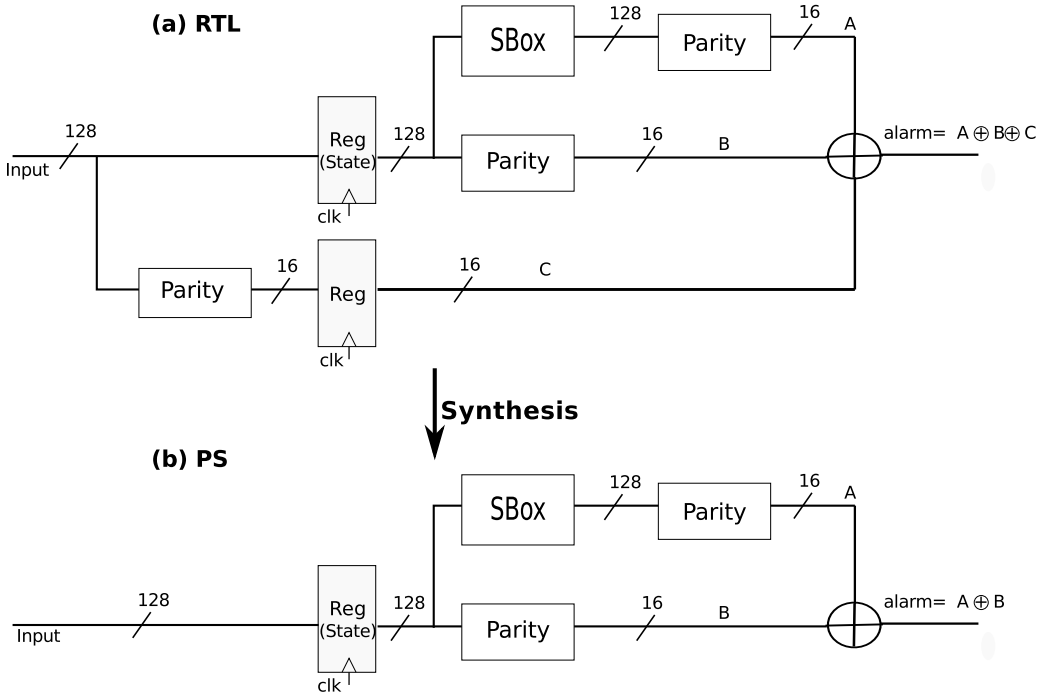

Figure 16: Partial simplification of fault detection logic upon synthesis.

Table 3: Fault detection rate for RTL and post-synthesis levels.

Level	RTL	PS (default options)	PS -logic_opt = true -xor_collapsing = true	PS -logic_opt = true DONT_TOUCH attribute
Detection rate	100%	100%	18.75%	56.43%

Tab. 3 summarizes the fault detection rate according to the analyzed level. From this, we conclude that the protection can be removed altogether during logical synthesis, thereby causing a security regression. This kind of mis-integration may happen in real case, where designers do not check the security evolution of their design at each stage of synthesis. Therefore, robustness of hardware cryptographic modules against fault injection attacks should be evaluated at each abstraction level in the design conception flow.

Another reason for designers attention to be deflected from security is the requirements for testability. Clearly, in Fig. 16(a), the alarm signal is not testable. Indeed, it is consistently equal to '0'. Therefore, in a view to achieve DFT (Design For Test) requirements, some test logic to address independently the registers driving signals A, B & C, shall be added. But in the meantime, the designer might shift its focus so conscientiously that he might forget about the need for setting

DONT_TOUCH attributes. Hence the need for the PAVT tool as an independent third-party verification tool.

4 Conclusion

SCA and FIA are serious threats to cryptographic algorithms [12]. Countermeasures have been developed against such attacks. Still, it is non-obvious how to implement such protections at source-code level. There are many options to configure the synthesis. Hence exploring their combinatorics is exponential. In practice, users select a few options. Some options can lead to total or partial simplification of the countermeasure. Using a simulation-based methodology, we manage to detect such alterations and we quantify the amount of degradation. In addition, we precisely pinpoint the residual leakage samples.

Acknowledgments

This work has benefited from a funding via TeamPlay (https://teamplay-h2020.eu/), a project from European Union's Horizon2020 research and innovation programme, under grant agreement N° 779882. Besides, the French FUI (AAP-22) program CSAFE+ also funded part of this work.

References

[1] Noemie Beringuier-Boher, Marc Lacruche, David El-Baze, Jean-Max Dutertre, Jean-Baptiste Rigaud, and Philippe Maurine. Body Biasing Injection Attacks in Practice. In Martin Palkovic, Giovanni Agosta, Alessandro Barenghi, Israel Koren, and Gerardo Pelosi, editors, *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems, CS2@HiPEAC, Prague, Czech Republic, January 20, 2016*, pages 49–54. ACM, 2016.

[2] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. *IEEE transactions on Computers*, 52(4):492–505, 2003.

[3] Guido Bertoni, Luca Breveglieri, Israel Koren, and Vincenzo Piuri. Fault detection in the advanced encryption standard. In *Proc. Conf. Massively Parallel Computing Systems (MPCS'02)*, pages 92–97, 2002.

[4] Johannes Blömer, Jorge Guajardo, and Volker Krummel. Provably Secure Masking of AES. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography*, volume 3357 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2004.

[5] Éric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[6] David Canright. A Very Compact S-Box for AES. In Rao and Sunar [19], pages 441–455.

[7] David Canright and Lejla Batina. A Very Compact "Perfectly Masked" S-Box for AES. In *ACNS*, volume 5037 of *Lecture Notes in Computer Science*, pages 446–459, 2008.

[8] Jean-Luc Danger, Sylvain Guilley, Philippe Nguyen, Robert Nguyen, and Youssef Souissi. Analyzing security breaches of countermeasures throughout the refinement process in hardware design flow. In David Atienza and Giorgio Di Natale, editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, pages 1129–1134. IEEE, 2017.

[9] Christophe Giraud. DFA on AES. In *Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers*, pages 27–41, 2004.

[10] Sylvain Guilley and Jean-Luc Danger. Global Faults on Cryptographic Circuits. Chapter 17 of [12].

[11] Annelie Heuser, Olivier Rioul, and Sylvain Guilley. Good Is Not Good Enough - Deriving Optimal Distinguishers from Communication Theory. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 55–74. Springer, 2014.

[12] Marc Joye and Michael Tunstall. *Fault Analysis in Cryptography*. Springer LNCS, March 2011. DOI: 10.1007/978-3-642-29656-7 ; ISBN 978-3-642-29655-0.

[13] Ronan Lashermes, Guillaume Reymond, Jean-Max Dutertre, Jacques Fournier, Bruno Robisson, and Assia Tria. A DFA on AES Based on the Entropy of Error Distributions. In IEEE, editor, *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 34–43, Sept 2012.

[14] Huiyun Li, A. Theodore Markettos, and Simon W. Moore. Security evaluation against electromagnetic analysis at design time. In Rao and Sunar [19], pages 280–292.

[15] Damien Marion, Adrien Facon, Sylvain Guilley, Thomas Perianin, and Matthieu Lec'hvien. Binary Data Analysis for Source Code Leakage Assessment. In *International Conference on Information Technology and Communication Security (SecITC), Budapest, Romania, November 8-9, 2018. Proceedings*, 2018.

[16] Marcel Medwed, François-Xavier Standaert, and Antoine Joux. Towards Super-Exponential Side-Channel Security with Efficient Leakage-Resilient PRFs. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 193–212, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[17] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security*, pages 529–545, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[18] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *Journal of Cryptology*, 24(2):292–321, Apr 2011.

[19] Josyula R. Rao and Berk Sunar, editors. *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*. Springer, 2005.

[20] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A Compact Rijndael Hardware Architecture with S-Box Optimization. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT 2001*, pages 239–254, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[21] Laurent Sauvage, Sofiane Takarabt, and Youssef Souissi. Secure silicon: Towards virtual prototyping. In *2017 International Symposium on Electromagnetic Compatibility - EMC EUROPE*, pages 1–5, 4-7 September 2017. DOI: 10.1109/EMCEurope.2017.8094744. Angers, France.

[22] Secure-IC S.A.S. VIRTUALYZR® tool. `http://www.secure-ic.com/solutions/virtualyzr/` (accessed on November 2nd, 2018).

[23] Nidhal Selmane, Sylvain Guilley, and Jean-Luc Danger. Setup Time Violation Attacks on AES. In *EDCC, The seventh European Dependable Computing Conference*, pages 91–96, Kaunas, Lithuania, May 7-9 2008. ISBN: 978-0-7695-3138-0, DOI: 10.1109/EDCC-7.2008.11.

[24] Sofiane Takarabt, Kais Chibani, Adrien Facon, Sylvain Guilley, Yves Mathieu, Laurent Sauvage, and Youssef Souissi. Pre-silicon embedded system evaluation as new EDA tool for security verification. In *3rd IEEE International Verification and Security Workshop, IVSW 2018, Costa Brava, Spain, July 2-4, 2018*, pages 74–79. IEEE, 2018.