A Low-Cost Methodology for EM Fault Emulation on FPGA

Paolo Maistri and Jiayun Po
Univ Grenoble Alpes, CNRS, Grenoble INP*, TIMA, 38000 Grenoble, France
paolo.maistri@univ-grenoble-alpes.fr

***Abstract*—In embedded systems, the presence of a security layer is now a well-established requirement. In order to guarantee the suitable level of performance and resistance against attacks, dedicated hardware implementations are often proposed to accelerate cryptographic computations in a controllable environment. On the other hand, these same implementations may be vulnerable to physical attacks, such as side channel analysis or fault injections. In this scenario, the designer must hence be able to assess the robustness of the implementation (and of the adopted countermeasures) as soon as possible in the design flow against several different threats. In this paper, we propose a methodology to characterize the robustness of a generic hardware design described at RTL against EM fault injections. Thanks to our framework, we are able to emulate the EM faults on FPGA platforms, without the need of expensive equipment or lengthy experimental campaigns. We present a tool supporting our methodology and the first validations tests done on several AES designs confirming the feasibility of the proposed approach.**

Keywords—EM fault injection, emulation, clock glitching, FPGA, RTL

I. Introduction

Many applications nowadays require confidentiality, integrity, and authentication, which can be achieved through the implementation of cryptographic primitives. It is hence quite common to find these implementations in many embedded systems, either as software or, if demanded by requirements in terms of performance or efficiency, as hardware designs.

Although cryptographic algorithms can be considered reasonably secure, it is however well known that careless implementations of the secure primitives can nonetheless reveal critical information. This kind of attacks directed to the actual implementation can be passive [1][2], where the behavior of the system is observed and analyzed, or active [3], where the regular functioning is altered in order to gather useful information about the data being processed.

The correct behavior of the device can be altered in several ways, with the usual goal of injecting some computational error that will be further analyzed and exploited in order to break the cryptosystem. Fault injection attacks can be performed by using a variety of techniques: voltage and clock glitches, laser or EM illumination, etc. All these techniques have different properties in terms of precision, required knowledge, and cost. Glitches, for instance, are quite simple and cheap to implement but act on global signals of the integrated circuit (e.g., the clock or the voltage supply), so they affect the design in its entirety: for this reason, they are mostly effective against software implementations (running on CPUs or microcontrollers), where the serial execution of the code limits the error propagation. On the other hand, their impact on hardware implementations is much less predictable or controllable: hence, secure hardware implementations are usually attacked through laser or, more recently, EM fault injections (EMFI), which exhibit good spatial locality and excellent temporal precision. In particular, laser fault injection (LFI) has excellent controllability even when attacking reasonably recent technology nodes [4]: on the other hand, the cost in terms of equipment, preparation, and related expertise is not affordable for everybody. EMFI attacks, while still providing good potential from the point of view of the attacker, are more accessible and are thus becoming an interesting attack technique in the recent literature.

For these reasons, designers must hence take into consideration such threats as early as possible, in order to assess the vulnerability of the architecture against physical attacks and propose solutions (i.e., countermeasures) at all levels in order to face these threats. This may constitute a serious challenge: protecting as early as possible means that solutions must be proposed at software (if it is the case) or Register Transfer Level (RTL), when there may be no actual silicon yet to perform the experiments.

In this work, we want to propose a methodology aimed at validating the architectural consequences of EM fault injections. Our framework does not aim at being an alternative attack technique nor a generic fault injection framework, but it allows emulating the EM fault on a configurable platform (an FPGA board) by mimicking the spatial and temporal characteristics of EM pulses on any digital designs, without any additional equipment other than a programmable board. Thanks to our approach, designers can therefore validate architectural solutions with respect to a fast, inexpensive, and accurate flow. We have validated the methodology on three different hardware implementations of the Advanced Encryption Standard, proving that exploitable faults, issued from literature models, can be easily injected.

This paper is organized as follows. In the next section, we resume the existing state of the art, which constitute also the motivation behind this work: we briefly describe the experimental platforms for EMFI, and the main abstractions that have been proposed to model the interaction between the EM pulse and the silicon. Section III describes our methodology more in detail, highlighting the different steps of our approach. First preliminary results are elaborated in Section IV, along with some perspectives on future research directions. Finally, Section V concludes the paper.

II. Motivation

Electromagnetic Fault Injection (EMFI) has been gaining in popularity in the recent years. EMFI does not require special preparation of the integrated circuit (i.e., no backside

*Institute of Engineering Univ. Grenoble Alpes

access for illumination, no lapping in order to expose the silicon layers) and the experimental setup is of lower complexity: a pulse generator, an injection probe, and a control machine are the main components of the platform (with the target as well). After a first period when experimental platforms were mostly built using commercial or home-made parts, a few complete platforms have recently made their appearance on the market [5]. Nonetheless, these setups still constitute an important investment [6]. Some studies have explored alternative solution to improve the interaction between the probe and the circuit, but are mostly oriented to passive analysis [7].

For these reasons, it is important to understand the mechanisms of error injections due to EMFI. Several studies have appeared in the latest years, aiming at modelling at different levels the effects in both hardware and software implementation of cryptographic algorithms. In [8], AES was attacked by EMFI both in software, running on an 8-bit microcontroller, and on a hardware iterative design implemented on FPGA. Although the fault model of the two campaigns was not directly comparable, the authors were nonetheless able to propose and validate some initial guess about the error injection mechanism: according to their conclusion, the most likely fault model was due to localized timing violations. The authors did not prove the assumption from a formal point of view, but their experimental validation of a timing countermeasure seemed to confirm their assumptions. This similarity was also confirmed in later works, such as [9].

Nonetheless, the understanding of the actual mechanisms occurring during EMFI is not as simple as that. Several other studies have shown that timing faults are only part of the global picture, as the EM pulse can disrupt the set/reset mechanism of flip-flops in a subtler way [10]. This mechanism has been later detailed as a double interaction with the power grid [11], as the EMFI creates two opposite pulses within the circuit. While the first lowers temporarily the supply voltage (thus creating timing violations), the second restores it and possibly interferes with the DFF sampling. This is confirmed by the fact that the experiments heavily depend not only on physical location of the injection, but on its exact timing as well [12].

Despite the complexity of the actual interaction between the EM pulse and the gates, it is generally accepted that EM faults can be approximated by timing faults in a first instance [13]. This assumption was thus adopted in [14], where the authors have shown the feasibility of using internal primitives of an FPGA to create clock glitch at a local level. However, their approach was limited to just evaluate the impact of the modification on resource usage and on performance, without any actual analysis of the faults that could be injected.

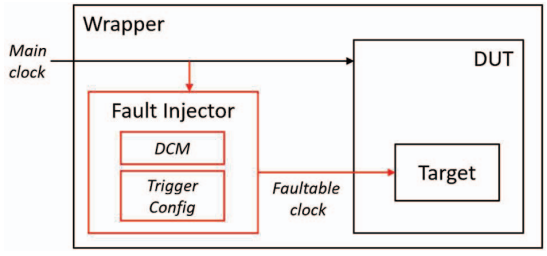

Fig. 1. Clock signals in the design, and how they are generated. Red color shows new or modified elements of the design.

Additionally, their methodology was applied manually on a specific design (namely, a lightweight ASCON architecture) and in a specific location.

In this paper, we want to show that an automated approach can be generalized on any design and that exploitable faults can be injected. We describe our methodology in the next section.

III. Methodology

A. Basic Principles

Our goal is to control timing violation faults in any part of the design. This fault model is quite specific to EM fault injection, as highlighted in the previous section: other fault attacks (or general reliability assessments) may be based on different fault models, such as bit-flip or bit set/reset. In order to accomplish this result, we instrument the design in order to create a clock glitch only in the specific part that has been defined as the target of the injection: as a consequence, we can emulate the sampling faults that would occur in the gates when targeted by EM injections. This approach first requires a clock glitch to be generated internally in the design and routed to the target. The second constraint is time-related: the glitch must occur only when needed, and nominal conditions (i.e., the regular clock) should be in general restored immediately after the glitch. The triggering condition can be set by the designer, but it is usually based on a custom delay with respect to a specific event (e.g., the start of the encryption).

Although these two conditions seem easy to satisfy, there are some details that require attention from the designer. First, the spatial granularity of the fault injection: in principle, the target can be from a single flip-flop up to an entire module. In practice, if we aim at emulating local faults, smaller granularities should be used (such as a few FFDs or a specific register), as our goal is not to emulate global clock glitches. More advanced methods can be envisioned, as suggested in Section IV. Secondly, the glitch may occur either on the rising or the falling edge of the clock signal, or even both: different policies may cause different behaviors, depending on the actual design under attack, and should be taken all into consideration. In order to minimize the modifications to the original design, all the additional blocks (glitch generator, trigger, etc) can be defined separately in a larger wrapper module, containing the target as well, as schematically shown in Fig. 1.

The glitch can be generated and controlled by using two synchronous clocks in the designs: the regular and the faultable clocks. The latter clock will be used to drive only the cells within the target (i.e., the flip-flops where the fault has to be injected), while the former will be used by the rest of the design. The two clock signals can be created through the FPGA Digital Clock Manager (DCM), and managed by the *Fault Injector* module. When the trigger is off no glitch is injected, meaning that the two clocks are identical; during the injection, the faultable clock (and only this clock) is perturbated with the glitch, and the sampling in the corresponding targets is corrupted. After the glitch, the two clocks are again identical, which allows resuming the computation process without any further error. The faulty output can then be recovered for cryptanalysis.

B. Tools and Implementation

Our proposed methodology is largely based on existing FPGA design flows. This is justified by the fact that our goal

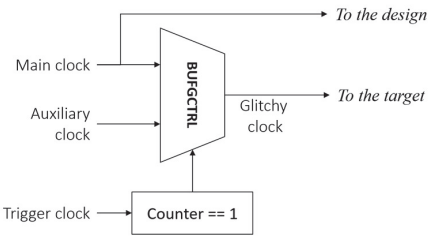

Fig. 2. Clock signals in the design, and how they are generated

is to emulate EM injections to evaluate architectures and countermeasures at RTL or higher level: FPGA platforms are therefore perfectly suited, thanks to their configurability and large spectrum of devices. We have used Xilinx devices, but any other programmable board can be used if the main requirements are met.

In order to generate the glitch internally, we rely on the BUFGCTRL primitive available on Xilinx FPGAs [15]. These components are well known clocking resources that can be used to switch between two input clocks without any glitch when properly configured. It is important to highlight that in our case, the glitch-free property of the primitive ensures that we can finely control the clock output: if a glitch occurs, it is deterministically due to our trigger and not to undefined behavior of the logic itself. This behavior can be achieved by configuring the additional inputs, namely the *Ignore*, *Enable*, and *Selection* input pins, to specific values.

The clock switching is controlled by a decrementing counter, When the counter reaches the activation value, BUFGCTRL is triggered and changes to the secondary input clock. In order to preserve the global synchronization of the computation process, the regular clock is then restored. To simplify this step, the clock switch occurs when the counter reaches 1, and restores the normal clock at 0.

The generic setup for glitch generation is shown in Fig. 2. The main clock is the nominal clock of the system; the auxiliary clock is used for glitch generation, and it is selected by the BUFGCTRL primitive when the trigger is activated. Its actual shape can be chosen to create a glitch with a specific pattern: in our experiments we set it to the same frequency of the main clock, a duty cycle of 25%, and a phase shift that depends on the design under test (270° in our case), but the actual configuration may be tailored to the specific experiment. The trigger is updated on the main clock, but its edge sensitivity depends on the type of glitch (rising, falling, or both edges) that has to be injected.

A tool has been implemented in Python to automate the instrumentation process. It has been designed to accept any netlist in Verilog, which allows to easily process any synthesizable design. The tool identifies all possible targets by scanning the netlist and looking for keywords defining the sequential logic. Hierarchical or flattened designs are equally supported, while the list of keywords can be configured to adapt to different technology targets. The user is then allowed to choose a subset of targets, or the tool can be instructed to go through the list element by element for an extensive emulation campaign. The auxiliary clock, the triggering blocks, and the proper clock connections are handled directly by the tool in a transparent way for the user. The instrumented design can thus be used to replace the original one on the programmable board for the accelerated emulation experiment. In the perspective of comparing the results from this methodology against actual EMFI, the tool can optionally define a separate partition for the target in an automatic way, while the rest of the design is placed far from the target. This will allow easier placement of the EM probe during the attacks and the results will be more representative of the actual sensitivity of the selected target (and not of other glue logic).

C. Validation

The methodology was validated on three different hardware implementations of the Advanced Encryption Standard [16]: a trivial iterative design, an architecture protected by an Error Detecting Code (EDC) [17], and a design protected by hybrid temporal redundancy [18]. At this stage, no extensive campaign was performed to evaluate the countermeasures, as the main goal was rather to confirm the feasibility of the approach on designs having disparate constraints. All the designs were implemented on a Nexys A7-100T board, which is equipped with a Xilinx Artix-7 programmable chip. This board was chosen because of its large availability of clocking resources, and because the FPGA circuit is not protected by any heat spreader, making thus easier to plan later EM fault injection campaigns.

Several experiments were done in order to find the most suitable value for configuring the parameters of the glitch generation, in particular the frequency, phase, and duty cycle of the auxiliary clock, as well as different techniques to create effective glitches. More importantly, several tests were run on all designs to inject errors on different elements of the designs (e.g., all the state bytes) and at different times (i.e., different encryption cycles). Our experiments have shown that faults at byte level can be successfully injected anywhere and anytime in the design, by controlling the clock glitches at local level. As an example, we report the output of an encryption experiment in Fig. 3, where a standard pair of plain text and key from the NIST specifications were used for correct and a faulty encryption on the EDC-protected design. In this experiment, a clock glitch was injected on the output of the *SubBytes* operation on the least significant byte of the state during the 8th round. The figure highlights how the output of the faulty encryption is fully corrupted as expected, and the cryptanalysis by reverse computation of the result correctly identifies the error as occurring on the targeted byte and round, thus confirming the successful injection. Moreover, the injected error value can be guessed as well.

IV. Discussion

This methodology has been proposed to emulate EM fault injection on an FPGA without recurring to an actual experimental EM platform. The designer can then apply classical fault analysis on the experimental results in order to assess the exploitability of the faults.

The short-term objectives are comparing the results with actual EM injected faults, and evaluate architectural countermeasures against fault attacks. Although only

Input msg:	00112233 44556677 8899aabb ccddeeff
Secret key:	00010203 04050607 08090a0b 0c0d0e0f
Expected output:	69c4e0d8 6a7b0430 d8cdb780 70b4c55a
Faulty output:	9338a658 c8de3d85 edfc907e 64cbb5a1
Output error:	fafc4680 a2a539b5 353127fe 147f70fb
8th SubBytes error:	00000000 00000000 00000000 000000**ca**

Fig. 3. Experimental results with NIST test values and fault injection on least significant byte of the state (8th round, after SubBytes).

dedicated designs were presented in this work, the methodology can be easily applied to any synthesizable description of any architecture: for instance, it may be used to analyze the error propagation within the architecture of a general-purpose processor. The main requirement is that the netlist description has to be available and modifiable to allow clock instrumentation, such as in RISC-V architectures. If the netlist is encrypted, our approach cannot be used.

The parameter exploration is currently delegated to the designer, who needs to tune the fault injector module in order to fit the target constraints and find the best configuration. This may be a tedious process, which could be sped up with proper assistance from the tool if the longest path delay would be known for the given target. Functional relationship analysis might be exploited as well in order to identify sets of targets and estimate spatial locality at RTL as in [20]. This methodology, originally proposed to create a laser-based fault model at RTL, could be effectively applied also to EM faults as well, thanks to similar spatial properties of the two attack techniques.

So far, only single glitches have been taken into consideration. On the other hand, current FPGAs have several clocking resources available on chip, which would allow for multiple glitch domains at the same time. The device used in our experiments, for instance, has 6 clock management tiles, which can be used to create independent glitches in parallel. Triggering conditions may be further explored as well. In simple attacks, only one glitch is injected at a specific time, but a more advanced controller could be programmed to schedule more than one glitch for each execution. When combined with independent parallel glitching, cited above, this would allow emulating complex higher-order attacks, with spatial and temporal multiplicity higher than one.

V. Conclusion

Physical attacks are a well-known threat to embedded secure systems, and electromagnetic injections are an effective way to carry out fault attacks. Designers must therefore assess the vulnerability of implementations as early as possible in the design flow. In this paper, we present a methodology that can emulate EM faults by exploiting local timing violations. The proposed approach can help the designer to validate architectural countermeasures at Register Transfer Level against a realistic fault model on an actual prototype implemented on FPGA, without the need of an expensive EMFI platform.

The methodology has been fully implemented and the developed tool is being published and made accessible to the community. Exploitable faults have been injected into several designs. In the short term, the injected faults will be compared to those obtained by an actual EMFI platform, and error detecting countermeasures will be evaluated with the proposed approach.

Acknowledgment

This work has been partially supported by the French National Research Agency in the framework of the "Investissements d'avenir" program (ANR-15-IDEX-02).

References

[1] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. CRYPTO 1996: 104-113.

[2] P. C. Kocher, J. Jaffe, B. Jun. Differential Power Analysis. CRYPTO 1999: 388-397.

[3] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, C. Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. Proceedings of the IEEE 94(2): 370-382 (2006).

[4] J. -M. Dutertre et al., "Sensitivity to Laser Fault Injection: CMOS FD-SOI vs. CMOS Bulk," in IEEE Transactions on Device and Materials Reliability, vol. 19, no. 1, pp. 6-15, March 2019, doi: 10.1109/TDMR.2018.2886463.

[5] Toulemont, J. et al. "A Simple Protocol to Compare EMFI Platforms." IACR Cryptol. ePrint Arch. 2020 (2020): 1277.

[6] NewAE Technology Inc, "ChipShouter User Manual", available at https://media.newae.com/manuals/ChipSHOUTER_PRESS_1.1.pdf, 2019.

[7] J. Toulemont, F. Mailly, P. Maurine and P. Nouet, "Exploring flexible and 3D printing technologies for the design of high spatial resolution EM probes," 2021 19th IEEE International New Circuits and Systems Conference (NEWCAS), 2021, pp. 1-4, doi: 10.1109/NEWCAS50681.2021.9462763.

[8] A. Dehbaoui, J. Dutertre, B. Robisson and A. Tria, "Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES," 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, 2012, pp. 7-15, doi: 10.1109/FDTC.2012.15.

[9] N. Moro, K. Heydemann, E. Encrenaz, and B. Robisson, "Formal verification of a software countermeasure against instruction skip attacks," Journal of Cryptographic Engineering, vol. 4, no. 3, pp. 145–156, 2014.

[10] S. Ordas, L. Guillaume-Sage, and P. Maurine, "Electromagnetic fault injection: the curse of flip-flops," Journal of Cryptographic Engineering, vol. 7, no. 3, pp. 183–197, 2017.

[11] M. Dumont, M. Lisart and P. Maurine, "Modeling and Simulating Electromagnetic Fault Injection," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 40, no. 4, pp. 680-693, April 2021, doi: 10.1109/TCAD.2020.3003287.

[12] A. Menu, J. Dutertre, O. Potin, J. Rigaud and J. Danger, "Experimental Analysis of the Electromagnetic Instruction Skip Fault Model," 2020 15th Design & Technology of Integrated Systems in Nanoscale Era (DTIS), 2020, pp. 1-7, doi: 10.1109/DTIS48698.2020.9081261.

[13] M. Ghodrati, B. Yuce, S. Gujar, C. Deshpande, L. Nazhandali and P. Schaumont, "Inducing Local Timing Fault Through EM Injection," 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), 2018, pp. 1-6, doi: 10.1109/DAC.2018.8465836.

[14] G. Surya, P. Maistri and S. Sankaran, "Local Clock Glitching Fault Injection with Application to the ASCON Cipher." 2020 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS) (2020): 271-276.

[15] Xilinx Inc, "7 Series FPGAs Clocking Resources. User Guide," UG472, v1.14, available at www.xilinx.com/support/documentation/-user_guides/ug472_7Series_Clocking.pdf, 2018.

[16] National Institute of Standards and Technology (NIST), "FIPS-197: Advanced Encryption Standard," Nov. 2001.

[17] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "Error analysis and detection procedures for a hardware implementation of the advanced encryption standard," in IEEE Transactions on Computers, vol. 52, no. 4, pp. 492-505, April 2003, doi: 10.1109/TC.2003.1190590.

[18] P. Maistri and R. Leveugle, "Double-Data-Rate Computation as a Countermeasure against Fault Analysis," in IEEE Transactions on Computers, vol. 57, no. 11, pp. 1528-1539, Nov. 2008, doi: 10.1109/TC.2008.149.

[19] Digilent Inc, "Nexys A7™ FPGA Board Reference Manual", available at https://digilent.com/reference/_media/reference/programmable-logic/nexys-a7/nexys-a7_rm.pdf, 2019.

[20] A. Papadimitriou, D. Hély, V. Beroulle, P. Maistri and R. Leveugle, "A multiple fault injection methodology based on cone partitioning towards RTL modeling of laser attacks," 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014, pp. 1-4, doi: 10.7873/DATE.2014.219.