A Combined Design-Time/Test-Time Study of the Vulnerability of Sub-Threshold Devices to Low Voltage Fault Attacks

Alessandro Barenghi, Cédric Hocquet, David Bol, François-Xavier Standaert, Francesco Regazzoni *Member, IEEE*, and Israel Koren *Fellow, IEEE*

1 Introduction

Radio Frequency Identification (RFID) devices are nowadays used in a wide range of applications, such as health care, supply chain management, and pet identification. Such a pervasive diffusion raises concerns regarding privacy as RFID tags often store sensitive information. The design of RFID devices is commonly constrained by a very strict power and area budget. Thus, incorporating the circuitry needed to guarantee a sound security margin against attackers is a challenging task, as security primitives, if not properly implemented, are quite demanding in terms of area and power.

A particularly appealing solution to meet the power consumption constraints is to exploit nanometer CMOS technologies, while adopting known aggressive power saving techniques, and operating the device at a subthreshold voltage [1]. Typical supply voltages employed in this context range between 0.3V and 0.5V, significantly lower than the common ones needed to work in the saturation region (1-1.2V). Nanometer CMOS technologies also allow to meet the area constraints when implementing standard cryptographic algorithms such as AES. As the manufacturing processes for nanometer CMOS technologies become more widespread, a larger number of commercial applications will be able to afford the cost of using RFID tags [2]. Using low power cell libraries and operating the device at a subthreshold voltage, result in a significant reduction of the power consumption but at the cost of a considerably lower clock frequency at which the device will operate. This, however, is acceptable since, even if speed in RFIDs is an important design parameter, it is not commonly a critical one.

A key concern for every secure cryptographic implementation is its vulnerability to both active and passive side-channel attacks. When designing RFIDs, it is particularly important to evaluate their resistance against low cost physical attacks. A well-known such attack is a fault injection carried out by simply decreasing the supply voltage. It has been shown that cipher implementations may experience functional failures if the Vdd is reduced below the reference supply voltage. In [3] we demonstrated the practical feasibility of these attacks against nanometer CMOS devices working at a subthreshold voltage. The considered AES co-processor design has a data path of 8 bits and is implemented using a 65nm low power library [4]. In particular, it was shown that it is possible to

- *A. Barenghi is with the Department of Electronics, Information and Biotechnology, Politecnico di Milano, Milano, Italy.*
- *C. Hocquet is with National Instruments, Zaventem, Belgium.*
- *D. Bol is with ICTEAM institute, Université catholique de Louvain, Louvain-la-Neuve, Belgium*
- *F. X. Standaert is with ICTEAM institute, Université catholique de Louvain, Louvain-la-Neuve, Belgium.*
- *F. Regazzoni is with Delft University of Technology, Delft, The Netherlands, and ALaRI - University of Lugano, Lugano, Switzerland.*
- *I. Koren is with University of Massachusetts at Amherst, Massachusetts, USA.*

generate timing faults (violations of the flip-flops' setup time) localized within a single byte of the state of the AES cipher. In the same paper, we also showed that it is possible to inject such faults even if the effects of process variations may shift the exact voltage level to which the power supply of the device should be lowered.

In this paper we address an important question in the design of secure, low-power, low-area cryptographic implementations: can the locations of potential setup-time violations (resulting from lowering the supply voltage) be predicted at design time, through industry standard EDA toolchains? Such predictions can allow the designer to assess the vulnerability of the circuit to fault injection attacks and modify the circuit design accordingly. We tackle this issue through extending the fault location and timing characterization of [3] down to the single bit level, for all the faults which can be used for attack purposes. By exploiting the detailed results of this analysis we investigate the feasibility of gathering information regarding the vulnerability of the chip at design time. In particular, we consider the feasibility of reproducing the faults, which we measured experimentally, using simulations that can be performed prior to manufacturing. In addition, we investigate the usefulness of static timing analysis to predict which state bits of the cipher are more likely to be affected by setup time violation induced faults.

The remainder of the paper is organized as follows. In Section 2 we provide background information on the AES cipher, briefly discuss several previously proposed fault attacks on AES and describe the one considered in our study. We then present the architecture of our AES design in Section 3. Section 4 describes the experimental setup that we employed to collect the fault measurements on the chip samples and the bit-level characterization of the faults. Finally, Section 5 reports our results concerning the feasibility of predicting the setup time violation faults at design time by employing an industry grade EDA toolchain, and discusses an efficient countermeasure which can be applied at design time. Section 6 presents our conclusions and points out future research directions.

2 BACKGROUND

In this section we provide a brief overview of the standard block cipher AES that is used as a case study in our experiments and simulations. We then review the known fault injection attacks against AES, focusing on those that can be mounted with a limited budget. This section also includes a detailed description of the basic differential fault attack that we have mounted in our experiments. We stress the importance of investigating faults induced through setup time violations caused by underfeeding the cipher implementation, as they are, non-destructive and easy to perform, thus representing a concrete threat to secure digital devices.

The AES Cipher: The cipher considered in this work is the Advanced Encryption Standard [5] due to its wide spread adoption. The selected variants of the Rijndael [6] algorithm that the AES standard supports include a plaintext block size of 128 bits and three key sizes of 128, 192 and 256 bits.

AES is based on the iteration of a round function composed of four primitives: SUBBYTES, SHIFTROWS, MIXCOLUMNS and ADDROUNDKEY. The number of times the round function is iterated, N_r, is 10, 12 or 14 times depending on the key length. The exception to the repetition of the four primitives are: the last round of the encryption is missing the MIXCOLUMNS primitive and an extra ADDROUNDKEY is performed before the first round as a pre-whitening of the input.

The inner state of the AES cipher after round r, denoted by S_r, is represented as a 4×4 matrix, where each element is 8-bit wide. We denote the n-th byte, counting from left to right, from top to bottom as S_r^n Each primitive of the AES cipher contributes either confusion or diffusion effects to the cipher, or adds a dependency on the value of the key. The SUBBYTES primitive is a non-linear mapping over $\mathbb{Z}_{2^8}$ that introduces a non-linear confusion effect. This mapping is applied to a single byte at a time, S_r^n, and can be implemented either as a lookup table or computed on the fly. The SHIFTROWS primitive provides a row-wise diffusion effect to the inner state of AES. It rotates the four rows of the state S_r by 0,1,2 or 3 byte positions, respectively. The MIXCOLUMNS primitive provides column-wise diffusion of the state by considering the column as a vector of values over $\mathbb{Z}_{2^8}$, and multiplying the vector by a constant matrix. The last operation, ADDROUNDKEY , combines the state of the cipher with a 4×4 key matrix through bitwise exclusive or (`xor`).

Since ADDROUNDKEY is repeated $N_r + 1$ times, there is a need to expand the initial key into $N_r + 1$ round keys through a key schedule routine. The key schedule process is non-destructive, i.e., all the operations performed are bijective. As a result, if a person is in possession of 4, 6 or 8 contiguous 32-bit words of the key schedule, he is able to reconstruct the full 128, 192 or 256 bit secret key.

Active side-channel attacks against AES: A number of fault injection attacks on the AES cipher have been reported in the literature. Although some of them were not experimentally validated at the time they were presented [7], [8], [9], [10], several other were successfully mounted on real world implementations. Among the ones viable with low cost equipment is the work of Hutter *et al.* [11] where the authors were able to mount a successful attack by causing temporary brown outs and glitches on the power supply line of an 8-bit microcontroller running a software implementation of AES. In [12], Schmidt *et al.* attacked an implementation of AES by blanking selectively the memory where the SBoxes are held, effectively reducing the entire AES algorithm to the last ADDROUNDKEY, performed on a zero-filled state. A technique which has proven effective in inducing controlled faults is through causing setup time violations by lowering the supply voltage below the level the circuit was designed for. In [13], Selmane *et al.* report the effects of attacking a commercial grade ASIC implementation of AES in a smart card, using this fault induction technique, while in [14] the authors successfully applied the technique to a full ARM9 core running a software implementation of AES. We refer the interested reader to a comprehensive survey on fault attacks and countermeasures in [15]. For the sake of completeness,

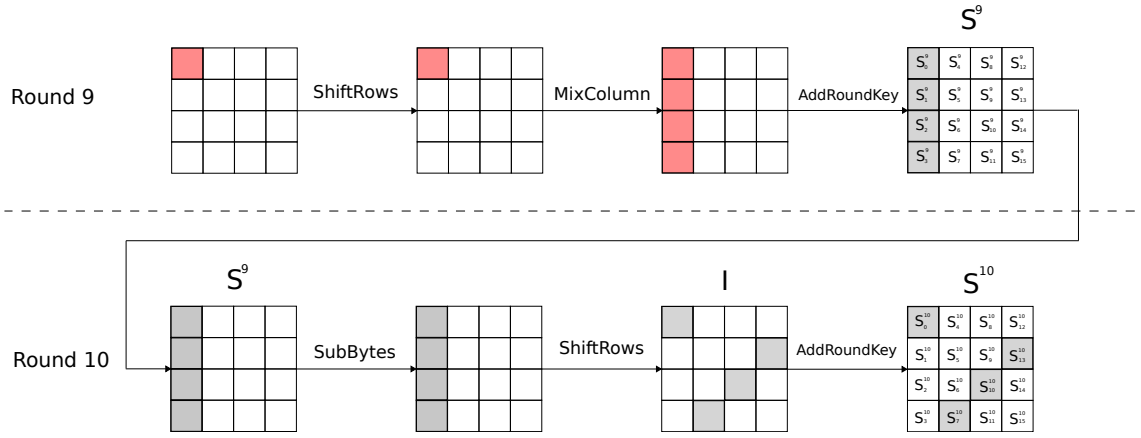


Fig. 1. Effects of a single fault injected between the MIXCOLUMNS operations of the eighth and ninth rounds. The fault propagates to only a quarter of the state, allowing the attacker to detect such a situation.

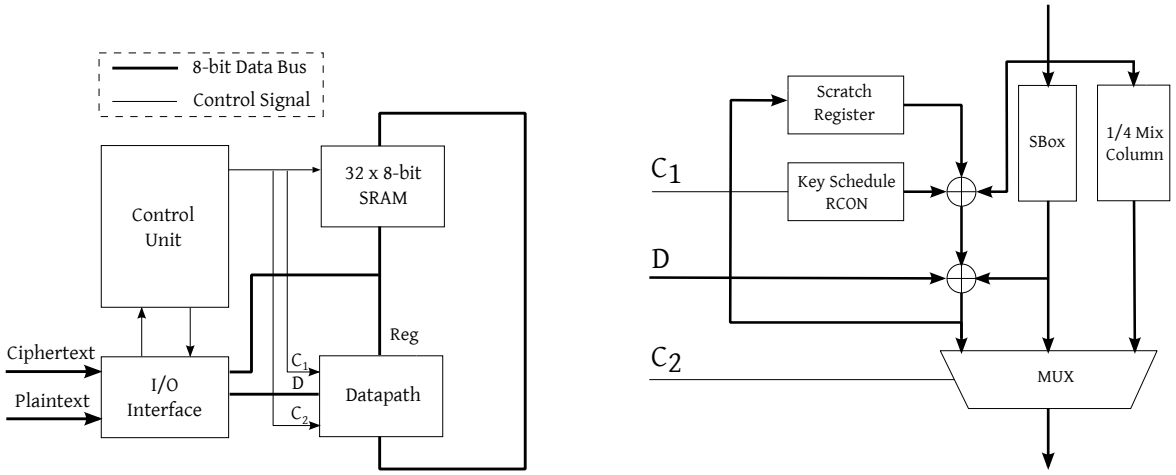
we also recall that passive side-channel analysis techniques have been successful in attacking real world implementations of RFID chips [16].

The attack under consideration: We now present the attack methodology considered in this paper to determine which faults represent a threat to the security of the AES implementation. Dusart *et al.* [17] have shown that it is possible to successfully retrieve the whole secret key of an AES-128 cipher, through the injection of a single byte-wide fault during the regular functioning of the cipher. The proposed attack relies on the injection of a single byte fault between the MIXCOLUMNS operation of the eighth round and the MIXCOLUMNS of the ninth round, as depicted in Figure 1. Due to the lack of the MIXCOLUMNS operation during the tenth round, the effect of the fault is spread only over 4 of the 16 bytes of the state. Since the key addition is performed byte-wise, the values of these 4 bytes are influenced only by 4 bytes of the last round key. Exploiting this fact and assuming that the injected fault has corrupted only one byte, the attacker may proceed to recover the 4 bytes of the key by comparing the correct and faulty ciphertexts.

To this end, the attacker makes an hypothesis on the unknown part of the key and proceeds to invert the effect of the last ADDROUNDKEY on the part of the cipher that was affected by the fault (greyed out in the figure) obtaining four values belonging to the state marked as I in Figure 1. This operation is performed on both the erroneous and the correct values of the ciphertext, yielding two groups of 4-byte values. Subsequently, the attacker inverts the effect of both the SHIFTROWS and SUBBYTES primitives, since their effect is fully known, obtaining successfully a faulty and a correct hypothetical values for four bytes of the state S^9, denoted respectively by $\widetilde{w} = \{\widetilde{S^9_0}, \widetilde{S^9_1}, \widetilde{S^9_2}, \widetilde{S^9_3}\}$ and $w = \{S^9_0, S^9_1, S^9_2, S^9_3\}$. Bypassing the effect of the ADDROUNDKEY operation, to further roll back the cipher, the attacker computes the exclusive or of $\widetilde{w}$ and w. Doing so, the effect of the ADDROUNDKEY function is canceled since in computing $\delta = w \oplus \widetilde{w}$ the key values are added twice. This allows the attacker to effectively compute the difference between the correct and the erroneous state of the cipher right before the MIXCOLUMNS operation. Since the MIXCOLUMNS operation is linear with respect to the exclusive or, it is possible to map the 4-byte difference δ into the difference before the operation by multiplying the value by the inverse of the matrix employed in the regular MIXCOLUMNS. At this point, the attacker may check if the obtained difference is actually composed of a single byte, as the fault model required by the attack mandates. Depending on whether the difference matches the fault model or not, the attacker decides if the key hypothesis made at the beginning of this rollback procedure is a valid one or not.

The attacker iterates the same difference analysis procedure for all the possible 2^{32} values of the four unknown bytes of the key and stores only the ones which produce a single byte difference before the last MIXCOLUMNS operation. A single pass of this procedure yields roughly a thousand valid candidates for the 32-bit wide key slice, and may be repeated if more than one faulty ciphertext caused by a single byte fault is available to the attacker. With a second sweep of the procedure the number of key candidates is reduced to one with a reasonably high probability [18]. Since it is possible for the attacker to discern, looking at the bytes that are affected by the faults, which slice of the key is the one under consideration, he can reconstruct the whole last round key with 4 injected faults and a brute force effort of $1000^4 \sim 2^{40}$ AES encryptions, which takes about a couple of minutes on a modern desktop computer, or with 8 injected faults and no brute force effort at all.

Further reduction of the number of faults needed for the attack can be achieved by injecting a single fault between the MIXCOLUMNS of the seventh round and the MIXCOLUMNS of the eighth round (instead of four single-byte faults injected simultaneously on each column of the state before the last MIXCOLUMNS), thanks to the diffusing effect of the SHIFTROWS operation of the ninth round. This way, it is possible to retrieve the complete last round key with a single fault and a modest brute force effort or two faults and no brute force effort. The main drawback of this method is that it is not possible to determine whether a faulty ciphertext has been the



(a) AES Component

(b) AES Component Datapath

Fig. 2. Block diagram of the AES module proposed by Feldhofer *et al.* [19] and implemented with a 65nm subthreshold cell library.

result of a fault complying with the required timing hypothesis, since the entire ciphertext value is altered. Nonetheless, if a fault which does not match the fault hypothesis is employed in the key recovery procedure, the number of valid key candidates drops to zero during the first iteration of the procedure, thus allowing the attacker to be aware of the issue.

After performing the aforementioned procedure, the attacker has all the bits of the last round key and is thus able to reconstruct the original key, in case a key size of 128 bit is employed. If larger key sizes are used, it is necessary to recover more key material in order to successfully break the cipher. An extension of the above attack is reported in [20], and requires twice the number of faults in order to recover the entire AES-256 key. The fault model assumed is the same, but the attacker has to inject faults also during the round before the one where the aforementioned attack takes place. The same paper also mentions the possibility of attacking even an AES cipher where no key schedule has been employed, and the 1408-bit key material is filled with random key bits. To this end, the attacker should be able to inject single byte faults during all the rounds of the cipher, thus retrieving all the round keys one by one.

3 Architecture of the Coprocessor

In this section we describe the structure of the AES co-processor chosen to implement the cipher, given the tight area and power constraints [4].

The selected 8-bit AES implementation is tailored to be used in low cost, low power devices such as RFIDs and follows the high-level structure proposed by Feldhofer *et al.* [19], supporting only a 128-bit key. The block diagram of this design is depicted in Figure 2. The chip communicates with the user through an 8-bit wide data bus managed by an I/O interface module. The state and the initial key of the cipher are stored in a 32-byte wide register file, connected to the 8-bit wide data-path of the chip, and driven by the finite-state control unit.

The compute portion of the design is composed of three components: a module to compute the non-linear transformation (S-box) that is used by the *SubBytes* primitive and the key expansion routine, a module to compute one quarter of the *MixColumn* primitive per clock cycle, and an exclusive-or module to perform the round key addition and the linear combinations of subkeys which are needed to execute the AES key schedule. The *ShiftRows* primitive is performed through a proper access pattern in the load and store operations.

The implemented AES co-processor differs from the one in [19] in terms of the technique used to implement the S-Box non-linear transformation. To achieve a lower gate count, we followed the guidelines of the lightweight S-box implementation proposed by Satoh *et al.* [21]. This implementation maps the input byte of the S-Box into the composite Galois field $\mathbb{Z}(((2^2)^2)^2)$, effectively reducing the input size of the nonlinear primitives to be computed to 2 bits. Then, it performs an efficient inversion of the four elements over $\mathbb{Z}(2)$ obtained from the previous mapping, and maps back the resulting inverted elements to $\mathbb{Z}(2^8)$, subsequently performing the affine transformation as a sequence of bitwise `xor` operations. This design technique that has a significantly lower gate count, was further optimized by Mentens *et al.* in [22], yielding the implementation technique that we employ.

The HDL code of the described AES design was synthesized for a target clock frequency of 100 kHz using Synopsys Design Compiler. The target technology was the STMicroelectronics 65nm LP CMOS technology with seven interconnect metal layers. Considering the fact that the device would operate in the subthreshold regime, we manually removed from the standard-cell library the cells which may not operate correctly, i.e., the ones with the longest transistor stack, such as NAND3 or NOR3. We set the timing condition to the worst-case process corner (slow NMOS and slow PMOS transistors), low temperature (-15 C), and operating supply voltage of 0.4V to achieve the desired 100KHz clock frequency. The optimized design was then placed and routed using Cadence

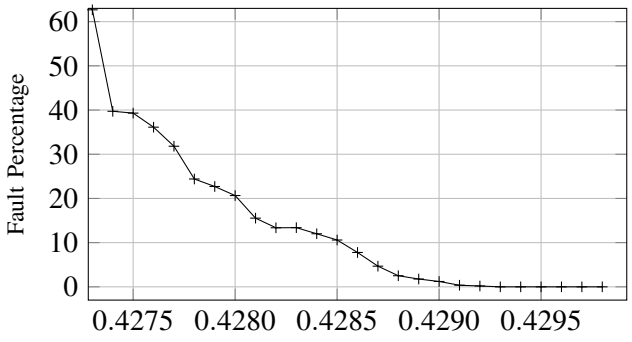


Fig. 3. Percentage of faulty outputs at different supply voltages for a single sample chip.

SoC Encounter and manufactured in silicon. Twenty dies were encapsulated in a 44-pin Ceramic Quad Flat Package (CQFP) and were tested to verify the correct execution of encryption and decryption. All twenty tested chips were found to operate correctly at a frequency of 1.3MHz with a power supply of 0.45V and 400KHz at 0.4V. By relaxing the clock frequency further, it is possible to keep the AES circuit operating correctly down to 0.25 V, which represents thus the functional limit of the design.

4 Experimental setup and Fault Characterization

This section describes the measurement setup used and the experiments conducted in order to profile the behavior of the low-power AES implementation and investigate the feasibility of low-cost fault injection attacks based on voltage throttling. It also provides a precise characterization of the setup time violation faults which can be exploited by an attacker.

4.1 Experimental setup

To perform the required measurements, the packaged chips were mounted on suitable sockets and dedicated PCBs were built. The chips under test were connected to a Keithley K236 tunable voltage generator, which is sufficiently precise to allow modifying the supply voltage by as little as 0.1mV. The power supply was connected directly to the power pin of the chips under test. All the tested chips were clocked at a frequency of 1.3MHz by means of an external clock generator and each encryption required 1100 clock cycles (less than 1ms). These clock rates and encryption times are within the typical range of RFID systems.

To carry out the experiments we connected the inputs/outputs of the PCB to a logic generator/analyzer, the National Instruments NI6552, which was connected to a PC where the plaintexts and keys were stored. The entire acquisition system, including the scaling of the supply voltage, was controlled by a Labview program, allowing to automate the acquisition process.

4.2 Performing the Attacks

The first experiment, with the objective of finding out whether the performance of the circuit degrades gracefully, was conducted by testing how many different encryptions the chip was able to perform while lowering the supply voltage by 0.1mV at each step. The voltage reduction was carried out in order to exactly identify the voltage level at which it is more likely that only a small number of setup time violations occur but no functional errors happen, which is the desired situation for an attacker. At each voltage step, we collected the results of ten thousand encryption operations and compared them to the correct ones. Figure 3 depicts the results of the experiments in terms of percentage of faulty ciphertexts versus supply voltage. As can be seen from the figure, there is a 0.8mV interval where the fault occurrence is limited to less than 10% of the encryptions and the fault occurrences gradually increase when the voltage is further lowered. The 0.8mV zone where the probability of a fault occurrence is low is relevant when performing fault attacks as it maximizes the likelihood of injecting a single fault into the computation, as opposed to the catastrophic behavior experienced when the supply voltage is considerably lower, as reported in [13], [23]. The 0.8mV supply voltage interval is well within the reach of the precision of the employed tunable voltage generator.

Circuits implementations in current CMOS technologies are known to experience process variations. These variations may have a noticeable impact on the behavior of the chip especially when it is operated at subthreshold voltage levels. An important question here is whether these process variations significantly reduce the already small exploitable voltage interval (where very few faults occur). To this end, we tested the impact of process variations on the position and width of the desired voltage interval. Figure 4 reports the results of conducting the same campaign on five different sample chips with the same AES design. As can be seen, process variations result in a noticeable offset of the fault injection threshold for the sampled chip, due to their significant impact on subtreshold timing characteristics [1]. However, it can be noticed that the rate of the degradation of the circuit performance is similar for all the samples and consequently, the width of the voltage interval stays about the same. This in turn implies that, regardless of the different offsets in the fault onset zone, it is possible to inject successfully single byte faults with the same low cost equipment. Note that an attacker may identify the proper voltage interval by lowering the supply voltage of the device using first large steps, and then much smaller steps once a faulty behavior is exhibited.

A related question is whether operating the chip at different temperatures may impact the success rate of the fault injection attacks since temperature changes are known to affect the working voltage of a circuit. Figure 4 also shows the effect of operating a chip instance at different temperatures achieved by placing the chip in a thermostatic chamber. As it can be seen, raising the working temperature of the chip lowers the voltage point where a significant amount of faults starts to happen. However, the degradation curve shows that it is still possible to identify a reasonable interval where the faults are

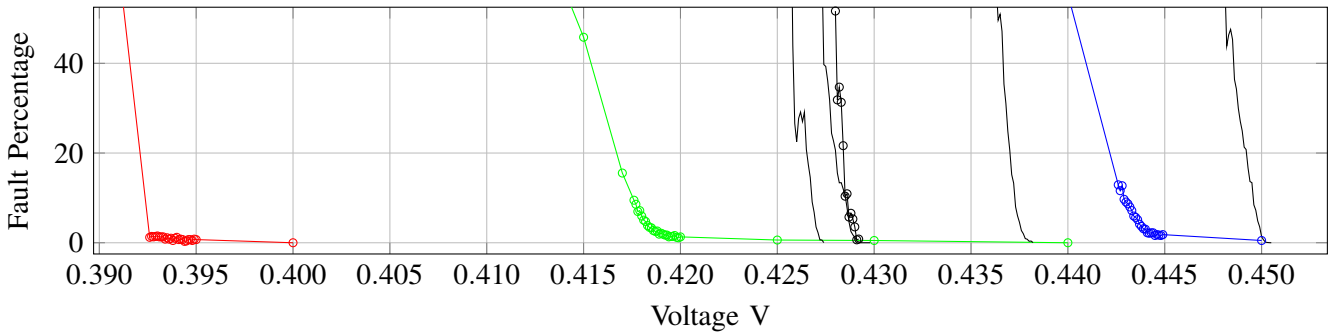


Fig. 4. Comparison of the voltage threshold of the first appearance of faults among different sample chips implementing the same AES co-processor design. The five chip degradation curves are depicted in black. The impact of temperature variations on the degradation curve of the chip marked by round plotmarks is depicted by the blue (T= 25C), green (T = 50C), and red (T= 75C) curves.

exploitable by an attacker.

After characterizing the graceful degradation of the chips, in terms of fault occurrence frequency, we investigated the actual fault patterns to find out whether single byte faults were present in the erroneous ciphertexts which were collected. To this end, we collected roughly 670k faulty ciphertexts while running the chip under test within the 10% faulty ciphertext region mentioned before. In order to uniformly stress the AES implementation, the plaintexts used during this campaign were selected from the NIST standard AES test vectors. The goal of this analysis was to understand the variations in the observed fault patterns. By examining the faulty ciphertexts produced by the device, we found out that the errors induced by setup time violations caused only 822 unique faults. This implies that the positions where the faults occur are very regular, since there was an approximate repetition rate of 1000 for each fault pattern. The fault repetition rate was uniform with respect to the different plaintexts.

The last step in the characterization of attacks on this AES implementation was to find out how many faults out of the ones obtained, were practically usable for carrying out the attack. To this end, we computed all the correct inter-state values for the ten thousand plaintext/key pairs employed in the experiments and the ones resulting from the decryption of the faulty outputs with the correct key. The inner states of the cipher at the beginning of each round obtained in this way were compared with the correct values and all the per-state differences were analyzed. We identified which faults were likely to have been generated by a single byte difference between the correct inner state and the faulty one. We note that, although we are not able to record the actual inner state of the chip, it is very unlikely that a multi-byte difference induces the same effect as a single byte fault in an AES encryption, due to the fast diffusion properties of the cipher. We also note that an attacker does not need to perform this analysis (which would be impossible for him due to the lack of the key knowledge), since all the mentioned fault attack techniques on AES are able to successfully discard faults that do not fit the desired fault pattern.

In order to avoid possible fault pattern repetitions due

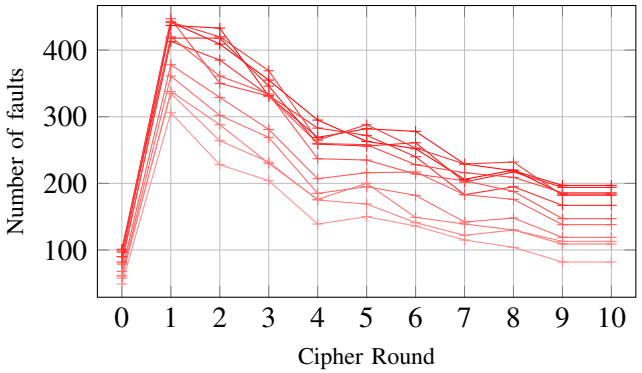
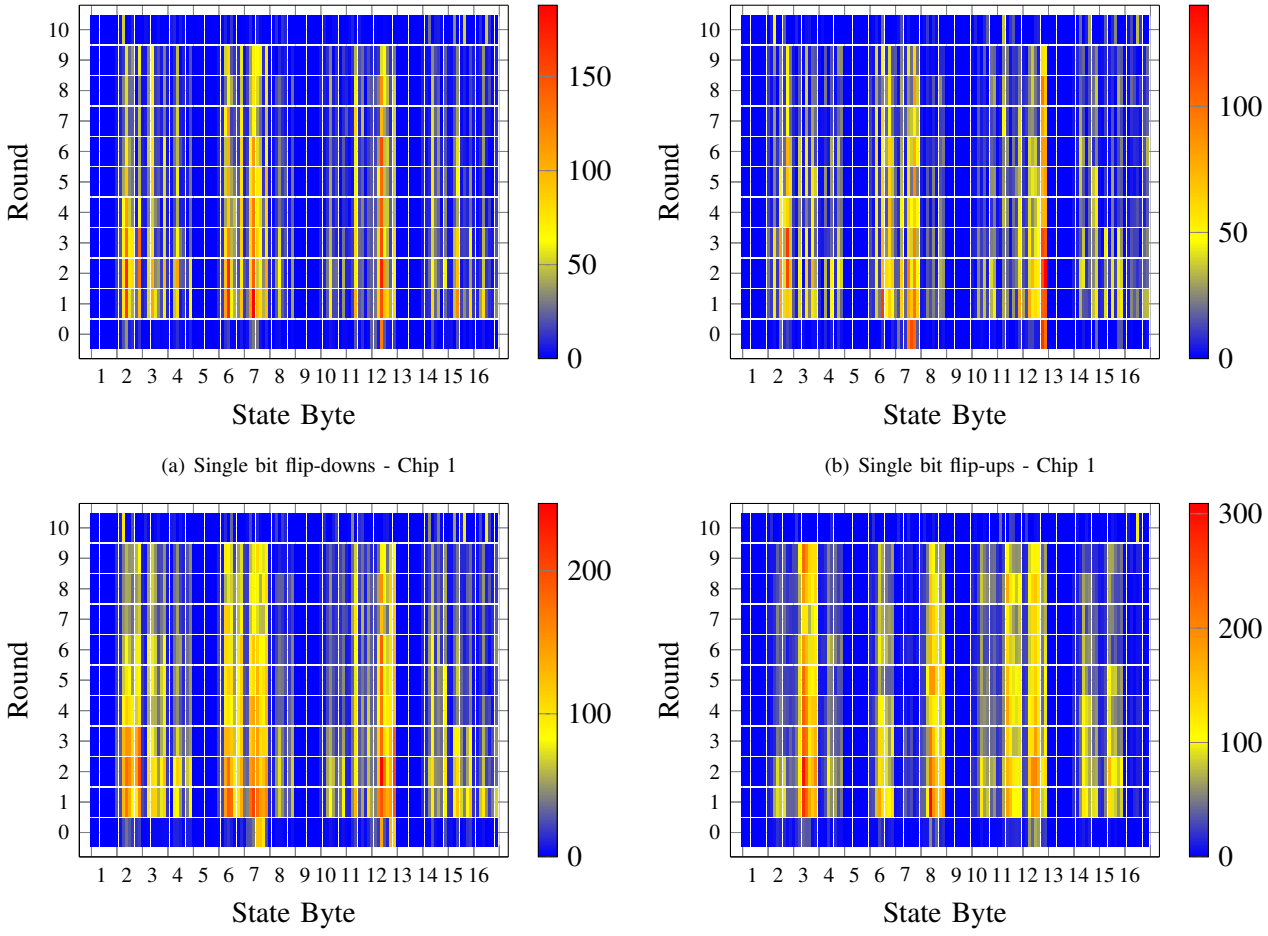


Fig. 5. Per-round distribution of the single byte faults in the states of the cipher on a chip sample. Round 0 indicates the faults occurring before the cipher started, allegedly during the load operations. Each line is obtained with a fixed voltage range, sweeping the $[426.6 - 427.7]$mV interval in 0.1mV steps, fainter colors represent lower voltages.

to the same plaintext or key, 10000 different plaintext and key combination were used as input to each experiment. We performed 10 different measurements lowering the voltage level by 0.1mV each time, while keeping the voltage within the low fault rate region, to determine how wide is the actual voltage window to achieve usable single bit faults. Based on these results we found out that, out of 39881 faulty results collected over the 10 runs, 30386 were the outcome of a single byte fault, thus resulting in an average 76% of the injected faults fitting the desired fault model (single byte modification). The percentage of exploitable faults ranges from 61% (lowest voltage) to 82% (highest voltage), supporting the observation that a bigger voltage drop induces gradually more catastrophic faults in the computation. This result implies that the actual exploitable window for the fault injection is at least 1mV wide. The last step to confirm the feasibility of mounting fault injection attacks on the chip is to verify that the faults are hitting the specific round positions required by the attack of [17]. Figure 5 shows the distribution of the single byte faults



(a) Single bit flip-downs - Chip 1

(b) Single bit flip-ups - Chip 1

(c) Single bit flips - Chip 1

(d) Single bit flips - Chip 2

Fig. 6. Bit level characterization of the single byte faults in the state of the cipher.

in the state of the cipher for each voltage level employed in the measurements.

As can be seen, the faults tend to hit all the rounds of the cipher, albeit in larger numbers for the earliest rounds. This different sensitivity to single byte faults can be ascribed to the critical paths in the control unit allowing slightly different slack times among different rounds due to the specific encoding of the state register.

The very low fault rate for the first state of the cipher is to be attributed to the fact that the architecture has just loaded the values and has not performed any significant operation on the plaintext yet. These results show that it is possible to generate successfully the required faults in order to attack the AES implementation under consideration. In particular, since the position of the single byte fault in the inner state is almost uniformly distributed, the hypotheses made in [17], [18] on the required number of faults hold.

To confirm the feasibility of an attack on the chip we executed the aforementioned attacks on an ad-hoc C software implementation of the algorithm on a Core 2 Quad Q6600 based desktop.

To provide an in-depth characterization of the faults, we examined all the single byte faults which we were able to measure, counting the number of their occurrences on a bit-level. Figure 6 shows a bit-level characterization of the single byte faults which we have observed in our chips. In particular, the state bit flips that occurred during the computation of each round of the AES are shown. Figures 6(a) and 6(b) depict the number of single bit flips, split into bit flip-ups (effectively a transient bit set fault) and bit flip-downs (transient bit reset). As shown in the figure, both flip-ups and flip-downs are present, thus yielding a fault model which differs from the one observed on a software-based AES by [23], where only bit resets were reported. We are not able to compare with the fault patterns obtained through setup time violations on a smart card in [13] as the paper does not report them.

Figures 6(c) and 6(d) show the cumulative single bit flipping statistics for two different chips of our design, to examine the impact of process variations on the fault patterns. As can be seen from the figures, the values of some state bytes are consistently more sensitive to faults across different instances of the chip (e.g., byte 12), while some bytes are never hit by a fault, namely bytes 1,5,9 and 13. The observations regarding the fault immunity of these four bytes were consistent in all the results we collected, and point to the possibility that the paths driving these bytes are effectively shorter than the other

ones. However, we note that, among the fault-sensitive bytes, the amount of sensitivity may vary significantly: for instance, byte 7 is affected by a large number of faults in Chip 1, while it is almost never hit by them in Chip 2.

5 Fault Predictability at Design Time

This section describes the EDA flow used to design the circuit under test and discusses how an early assessment of the potential fault attacks is carried out. In particular, we discuss how an early prediction of the points in the circuit which are more sensitive to setup-time violations can be identified using static timing analysis. Finally, we discuss to what extent the results obtained in simulations match the ones measured on the real chip, and we exploit them to propose efficient countermeasures against this attack strategy.

5.1 Design Toolchain description

The design flow used to assess the viability of fault attacks against the device under test is composed of common EDA tools. The inputs of the design flow are an HDL definition of the cipher implementation, in our case described in VHDL, and the target standard-cell library, in the STMicroelectronics 65nm LP CMOS technology.

As previously mentioned, we first recharacterized the library at the desired voltage, then we synthesized our design setting the appropriate target frequency.

The synthesized netlist was then placed and routed using Cadence Design Systems *SoC Encounter*. Also produced were a parasitics file (in spef format), the final netlist of the circuit (in verilog format), and the back annotation of the delay (in sdf format).

Once the design was completed, we carried out a post place-and-route simulation using the netlist as well as the corresponding delays. This simulation is used to produce the test vector, in VCD format, which will be used in the following steps of the evaluation process. We performed two different analyses with the aim of discerning if the tools available in the common EDA flow for transistor level simulation and static timing analysis are effective in predicting the fault patterns of the chip under test.

5.2 Predicting faults with transistor level simulations

The first analysis was aimed at understanding if a transistor level simulator is capable of correctly predicting the fault patterns we have measured in practice. To this end, a transistor level simulation campaign was carried out using *Synopsys Nanosim*, a fast SPICE simulator, employing the aforementioned data files as input. In addition to the data files describing the circuit, Nanosim requires a transistor-level model of the gates available in the library. The device behavior was simulated for a working voltage between 0.3V and 0.5V, in 1 mV steps. The simulation resolution was set to 100ns, and we employed a clock signal with the same frequency as during the real measurements, yielding a number of faulty ciphertexts reasonably close to the numbers observed in the experiments. We can thus speculate that Nanosim is an effective tool to predict the setup time violations which we were able to measure in practice.

However, performing an extensive characterization of the chip through transistor level simulation, to find which are the paths that are more likely to fail through setup time violations, is very time consuming. In fact, such a characterization would require to perform a significant amount of simulations with different input-key pairs in order to provide a good circuit coverage. As one transistor level simulation requires 10 minutes in our case, performing a simulation matching the experimental collection campaign would exceed 10k hours of computation time. We therefore, followed another methodology to predict which paths are more likely to fail using static timing analysis.

5.3 Predicting faults with static timing analysis

To perform the static timing analysis, *Synposys PrimePower* was used, employing the spef and sdf files, the relative Verilog netlist, and the file describing the timing characteristics of the library used during the synthesis.

The technology library file is not the original one, but the one storing the power consumption and the timing information of the library after the recharacterization at the appropriate voltage.

The result of a static timing analysis is a very detailed timing characterization of the paths inside our design. We extracted the worst-case delays associated with the input connections of the state and key registers. Examining the delays, we noticed that the ones characterizing the cipher state and the ones characterizing the key bytes were the same, bitwise. We ascribe this behavior to the fact that both the cipher state and the key are stored in the same 32-byte SRAM bank, making it thus likely for them to have the same delays. Figure 8(a) depicts the delays characterizing the input lines of the latches storing the cipher state: it can be noticed that the bytes in position 1, 5, 9 and 13 have a significantly lower worst-case delay for their input signal. Figure 8(b) shows the number of faults per bit measured on chips 1 and 2, adding together all the faults occurring throughout the whole 10 rounds of the cipher, with fainter colors representing lower input voltages. It can be seen that while lowering the voltage, the number of faults per bit increases, but the percentage of faults hitting a single bit are roughly the same. This in turn points to an increase in the number of single byte faults when lowering the voltage, while maintaining the faulty bit pattern even at lower voltages. Comparing the two figures, one can notice that the bits having an input delay very different from the critical path delay (e.g., in bytes 1,5,9 and 13) are completely fault-free. In contrast, the bits characterized by an input delay close to the critical path are affected by a significant amount of faults. Also notice that, among them, the number of faults hitting a bit is strongly influenced by process variations in some cases. For instance, the bits in the eighth byte of the state are affected by only a few faults in chip 2, while they are the second most affected in chip 1.

Based on these results we can conclude that static timing analysis provides an effective way for the designer to predict

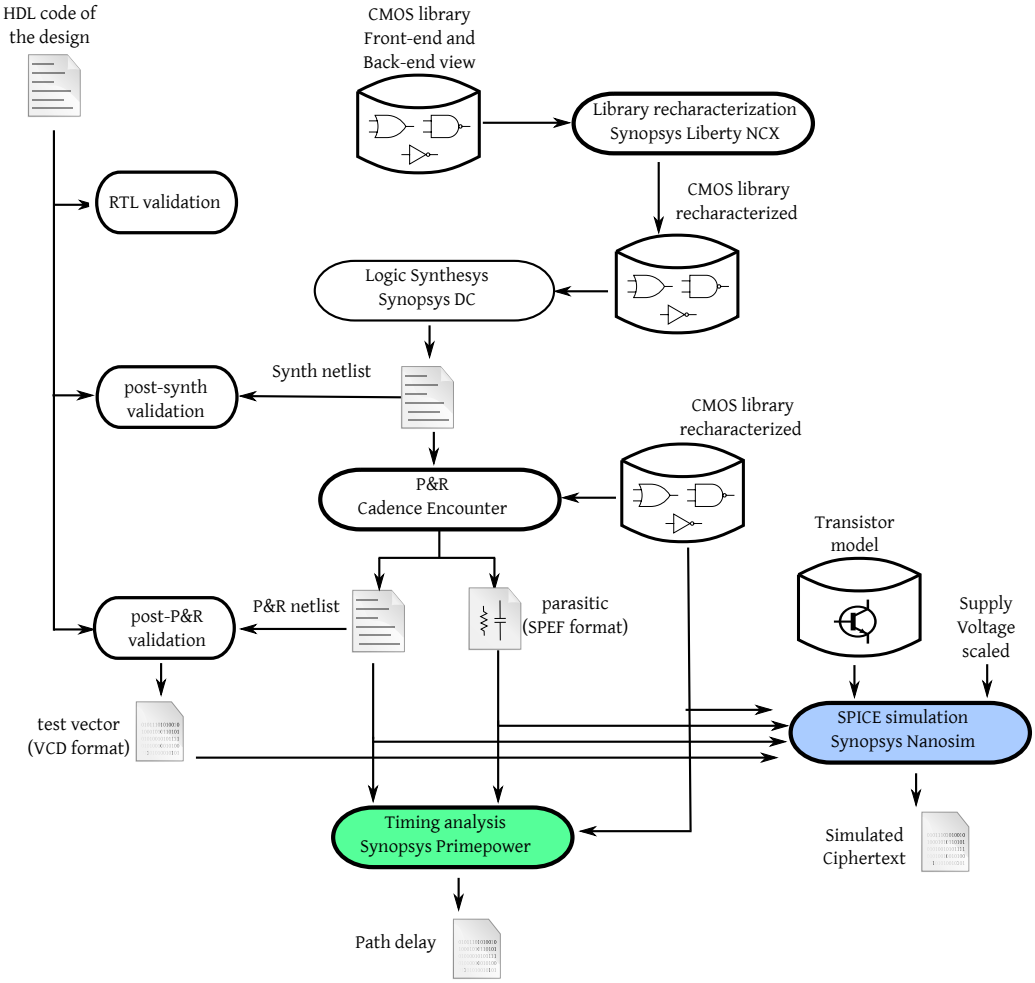


Fig. 7. Outline of the full simulation flow employed for both SPICE simulations and static timing analysis.

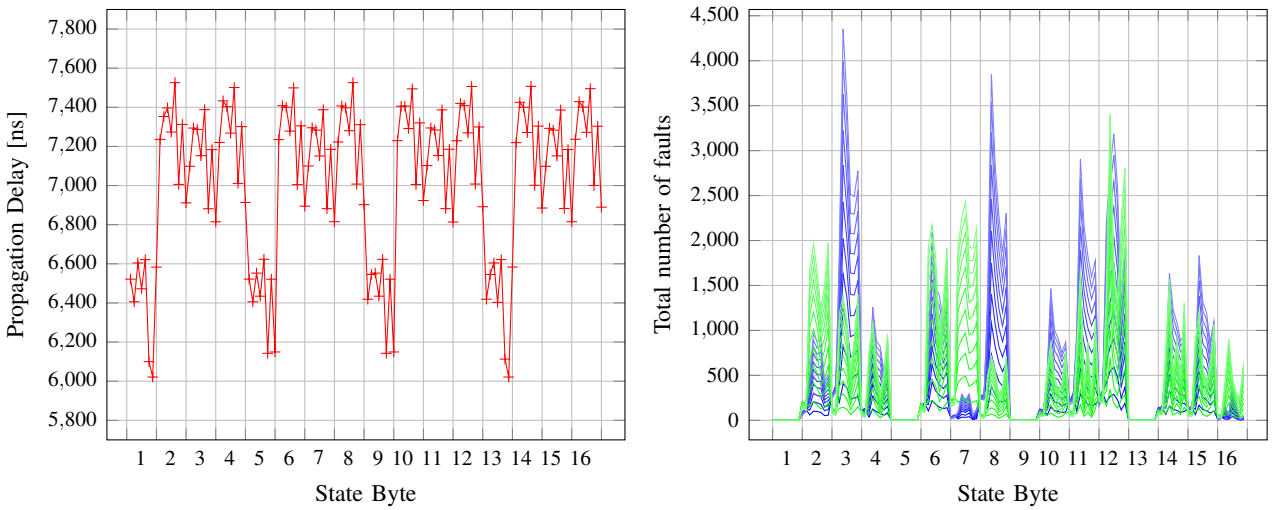
circuit paths which are likely to not experience setup time violations upon an attack, while among the ones at risk, the effective success rate of an attack is influenced by chip-to-chip process variations. In an attempt to propose an effective approach to prevent setup-time violation based fault attacks, we note that the attacker relies on the early failures of some particular lines to achieve a meaningful faulty result. A viable approach for a designer to protect the chip against such attacks could be through introducing an extra logic path, having a total delay sensibly longer than the critical path of the chip. Such a path will be acting as a canary, due to its extra length, in case a setup time violation is induced by the attacker. If a setup time violation on the canary path is detected, proper countermeasures against the fault attack, e.g., output random values as the ciphertext, and/or erase the key, can be activated. According to the delays reported in Figure 8(a) and the fault rates depicted in Figure 8(b), we note that paths shorter than 20% with respect to the critical path, are never hit by setup time violations, regardless of the process variations. It is thus sensible to assume that a canary path longer than the critical path by about 20% will be adequate.

6 Conclusions

In this paper we have presented a detailed characterization of an AES coprocessor realized with 65nm technology and operating in a subtreshold voltage region. The characterization showed that it is possible to effectively perform setup time violation attacks on the implemented ciphers through reducing the supply voltage in 0.1mV steps. We were able to provide a bit-level description of the fault frequencies and provide insights regarding their predictability with common EDA tools. In particular, we report that by employing static timing analysis tools, it is possible to obtain reliable estimates of the worst case timings for the input lines of the cipher state registers and pinpoint which ones are more likely to be vulnerable to setup time violation attacks. Finally, we proposed a countermeasure that designers can use in order to protect their design against these attacks.

Acknowledgements

This work was partially supported by the Walloon Region (E-USER and S@T Skywin projects), by the Nanotera program (SecWear project), and by the Swiss Commission for Technology (KTI/CTI project 12079.1 PFES-ES TRA.S.P.CH). François-Xavier Standaert is an associate researcher of the



(a) Worst case propagation timings and hold propagation timings for all the input lines of the state register

(b) Fault rates measured on Chip 1 (green) and Chip 2 (blue), adding together all rounds

Fig. 8. Input delays (worst-case) for all the bits of the cipher 8(a); state and amount of faults hitting every state bit 8(b).

Belgian Fund for Scientific Research (FNRS-F.R.S.). We acknowledge the support of Lubos Gaspar and François Stas from Université Catholique de Louvain, Louvain-la-Neuve, for performing the temperature-controlled measurements.

REFERENCES

[1] D. Bol, "Robust and energy-efficient ultra-low-voltage circuit design under timing constraints in 65/45 nm cmos," *J. of Low Power Electronics and Applications*, vol. 1, no. 1, pp. 1–19, 2011. [Online]. Available: http://www.mdpi.com/2079-9268/1/1/1

[2] D. Bol, R. Ambroise, D. Flandre, and J. Legat, "Interests and limitations of technology scaling for subthreshold logic," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 10, pp. 1508–1519, 2009.

[3] A. Barenghi, C. Hocquet, D. Bol, F.-X. Standaert, F. Regazzoni, and I. Koren, "Exploring the Feasibility of Low Cost Fault Injection Attacks on Sub-threshold Devices through an Example of a 65nm AES Implementation," in *RFIDSec*, ser. LNCS, A. Juels and C. Paar, Eds., vol. 7055. Springer, 2011, pp. 48–60.

[4] C. Hocquet, D. Kamel, F. Regazzoni, J.-D. Legat, D. Flandre, D. Bol, and F.-X. Standaert, "Harvesting the potential of nano-cmos for lightweight cryptography: an ultra-low-voltage 65 nm aes coprocessor for passive rfid tags," *J. Cryptographic Engineering*, vol. 1, no. 1, pp. 79–86, 2011.

[5] NIST, "Announcing the advanced encryption standard aes," FIPS Publication 197, Tech. Rep., 2001.

[6] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[7] C.-N. Chen and S.-M. Yen, "Differential fault analysis on aes key schedule and some countermeasures," *in Proc. Information Security and Privacy*, pp. 217–217, 2003.

[8] C. Giraud, "DFA on AES," *Advanced Encryption Standard - AES, 4th International Conference*, vol. 3373, pp. 27–41, 2005.

[9] C. H. Kim and J.-J. Quisquater, "New Differential Fault Analysis on AES Key Schedule: Two Faults Are Enough," *in Proc. International Conference on Smart Card Research and Advanced Applications*, pp. 48–60, 2008.

[10] A. Moradi, M. T. M. Shalmani, and M. Salmasizadeh, "A generalized method of differential fault attack against AES cryptosystem," *in Proc. International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 91–100, 2006.

[11] M. Hutter, T. Plos, and J.-M. Schmidt, "Contact-Based Fault Injections and Power Analysis on RFID Tags," *in Proc. IEEE European Conf. on Circuit Theory and Design*, pp. 409–412, 2009.

[12] J.-M. Schmidt, M. Hutter, and T. Plos, "Optical Fault Attacks on AES: A Threat in Violet," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 13–22, 2009.

[13] N. Selmane, S. Guilley, and J.-L. Danger, "Practical Setup Time Violation Attacks on AES," in *EDCC-7 '08: Proc. Seventh European Dependable Computing Conference*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 91–96.

[14] A. Barenghi, G. M. Bertoni, L. Breveglieri, M. Pellicioli, and G. Pelosi, "Low Voltage Fault Attacks to AES," in *HOST*, M. Tehranipoor and J. Plusquellic, Eds. IEEE Computer Society, 2010.

[15] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures," *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–3076, 2012.

[16] T. Kasper, M. Silbermann, and C. Paar, "All you can eat or breaking a real-world contactless payment system," in *Financial Cryptography*, ser. LNCS, R. Sion, Ed., vol. 6052. Springer, 2010, pp. 343–350.

[17] P. Dusart, G. Letourneux, and O. Vivolo, "Differential Fault Analysis on A.E.S," *CoRR*, vol. cs.CR/0301020, 2003.

[18] G. Piret and J.-J. Quisquater, "A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD," in *CHES*, ser. LNCS, C. D. Walter, Çetin Kaya Koç, and C. Paar, Eds., vol. 2779. Springer, 2003, pp. 77–88.

[19] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "AES implementation on a grain of sand," *Information Security, IEE Proceedings*, vol. 152, no. 1, pp. 13–20, 2005.

[20] A. Barenghi, G. M. Bertoni, L. Breveglieri, and G. Pelosi, "A fault induction technique based on voltage underfeeding with application to attacks against AES and RSA," *Journal of Systems and Software*, vol. 86, no. 7, pp. 1864–1878, 2013.

[21] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," in *ASIACRYPT 2001*, ser. LNCS, no. 2248, 2000, pp. 239–254.

[22] N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede, "A Systematic Evaluation of Compact Hardware Implementations for the Rijndael S-Box," in *CT-RSA*, ser. LNCS, A. Menezes, Ed., vol. 3376. Springer, 2005, pp. 323–333.

[23] A. Barenghi, G. Bertoni, E. Parrinello, and G. Pelosi, "Low Voltage Fault Attacks on the RSA Cryptosystem," in *Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2009, pp. 23–31.