Lightweight Protection of Cryptographic Hardware Accelerators against Differential Fault Analysis

Ana Lasheras, Ramon Canal, Eva Rodríguez
Departament d'Arquitectura de Computadors
Universitat Politecnica de Catalunya, Barcelona, Spain
ana.lasheras@est.fib.upc.edu, {rcanal,evar}@ac.upc.edu

Luca Cassano
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Italy
luca.cassano@polimi.it

***Abstract*—Hardware acceleration circuits for cryptographic algorithms are largely deployed in a wide range of products. The HW implementations of such algorithms often suffer from a number of vulnerabilities that expose systems to several attacks, e.g., differential fault analysis (DFA). The challenge for designers is to protect cryptographic accelerators in a cost-effective and power-efficient way. In this paper, we propose a lightweight technique for protecting hardware accelerators implementing AES and SHA-2 (which are two widely used NIST standards) against DFA. The proposed technique exploits partial redundancy to first detect the occurrence of a fault and then to react to the attack by obfuscating the output values. An experimental campaign demonstrated that the overhead introduced is 8.32% for AES and 3.88% for SHA-2 in terms of area, 0.81% for AES and 12.31% for SHA-2 in terms of power with no working frequency reduction. Moreover, a comparative analysis showed that our proposal outperforms the most recent related countermeasures.**

***Index Terms*—AES, Attack Resistance, Cryptographic Cores, Differential Fault Analysis, Hardware Security, SHA-2**

I. Introduction and Related Work

Hardware accelerators implementing cryptographic algorithms are nowadays employed in an wide range of products, such as smart phones and smart cards. These systems not only expose performance and cost requirements but also security ones. Mathematically speaking, modern cryptographic algorithms are robust [1]. Yet, their implementations may suffer from security flaws. In the last years, several cryptographic hardware accelerators demonstrated to be prone to a number of attacks, among which differential fault analysis (DFA) [2]. As a result, the deployed systems may be vulnerable although featuring security-dedicated modules.

DFA consists in: i) injecting faults into the circuit while it is performing encryptions/decryptions, ii) collectin+g incorrect outputs, and iii) analysing the collected outputs to infer secret information. DFA attacks rely on the ability to select the erroneous values to inject or the points of the circuit where to inject them (or even both). This makes the amount of data required to infer the secret information particularly small, thus allowing the attacker to achieve its goal in a reduced time [2].

Several approaches exist to protect cryptographic circuits against DFA. As a general solution, independent of the specific algorithm, it is possible to make the circuit physically inaccessible, e.g., through tamper-proof boxes and on-chip sensors, as for the high-end crypto-core IBM 4764 [3]. This approach is effective but very costly, since it relies on non-standard technologies. Algorithm-specific countermeasures exist and are less costly. In this paper, we focus on AES and SHA-2 (which are two widely used NIST standards).

When looking at AES-specific protection techniques, several proposals exist where time or spatial redundancy [4], [5] or error detection codes [6], [7] is applied. The main issue in these solutions is the area or working frequency overhead. More recent proposals have been presented in [8], [9]. In [8], the computation of each AES round is duplicated and results are compared. In [9], the AES original input is encoded into two different polynomial residue number system (which is not the residue checking referred in the current paper). Then, one nominal AES works on the original input while two replicas work on the encoded inputs. The results produced by the three replicas are then compared to detect the occurrence of a fault.

In the very last years, also SHA (in particular its keyed employment for authentication purposes) received attention from the security community and several protection mechanisms have been proposed [10]–[12]. In [10] parity-checking is employed to detect the occurrence of faults. In [11] the output of each round is rotated by a random number before the subsequent round, and then shifted back, so that the results do not change. Finally, in [12] two replicas are employed and the outputs of each round are scrambled between the two replicas.

In this paper, we propose a novel and lightweight protection technique based on partial selective replication against DFA attacks for hardware accelerators implementing the AES and SHA-2 algorithms. More in details, the *conventional* implementations of AES and SHA-2 are paired with replicas (dubbed the *n-AES*, and *n-SHA*, respectively). The input values fed into the replica circuit are a n-bit subset of the input values fed into the conventional circuit. At the end of the execution, the output of the replica circuit is compared with the corresponding n-bits of the output of the conventional circuit. In case a fault perturbated the processing of the conventional circuit, this check will fail; thus, identifying a possible fault attack. Otherwise, the check will pass. When a fault is detected, a (pseudo-)random output value is produced in order to make differential fault analysis unfeasible.

The remainder of this paper is organized as follows: Section II describes the basics of AES and SHA-2 and then surveys the existing AES- and SHA-specific DFA attacks; Sections III and IV introduce the details of our protection

technique; Section V reports about a set of experiments carried out to measure the effectiveness of the proposed solution and the introduced overhead also presenting a comparison with a set of related solutions; Section VI discusses the limitations of our proposal; Section VII concludes the paper.

II. The AES and SHA-2 algorithms and attacks

We introduce the basics of AES and SHA-2; and then, we survey the available DFA attacks. We do not discuss general attacks not specifically addressing HW implementations.

A. The AES Algorithm and DFA Attacks to AES

The *AES (Advance Encryption Standard)*, presented in [13], is the most widely adopted symmetric-key encryption algorithm. AES is based on the *substitution–permutation network (SP-network)* design principle which employs a network of *substitution boxes (S-boxes)* and *permutation boxes (P-boxes)*. Substitution and permutation are used to make the relation between the input *plaintext* and the output *ciphertext* difficult to understand (*confidentiality* property). Such SP-network takes the plaintext and the 128-, 192-, or 256-bit *key* as inputs, and applies 10, 12, or 14 *rounds* (depending on the key size) of S-boxes and P-boxes to produce the ciphertext. The output of each round is called the *state*. Generally speaking, an S-box (generally implemented as a lookup table) takes m input bits and transforms them into n output bits. P-boxes are then used to spread the output bits of an S-box through the input bits of the subsequent S-box. At the end of each round the *round key* (obtained from the overall key) is combined with the state by using XOR operations (*key addition*). Decryption is performed by reversing the process: using the inverses of the S-boxes and P-boxes and applying the round keys in reversed order.

Attacks to AES: A simple attack to AES has been proposed in [14]. It changes a single bit after the first key addition. The objective is to reset a single bit in the output state of the first round and to observe whether the value of the ciphertext has changed: if yes, the attacker infers that the original value of the bit in the state was 1 (because of the XOR operation), otherwise it was 0. With this attack it is possible to recover one bit of the key, and thus, by changing the injection location, the entire key can be retrieved, one bit at a time. This attack has been demonstrated to be practically infeasible due to the very precise timing required of the fault injection to exactly inject into the output state of the first round.

The advanced version of the previous attack has been proposed in [15], where the injection happens during the last round and causes a single-byte corruption in the output ciphertext. The attacker collects a pair of correct and corrupted ciphertexts, c and $\widetilde{c}$, respectively. In this way the attacker can reduce the number of possible bytes of the key that have been employed to encrypt the corrupted byte of $\widetilde{c}$ by inverting the effect of the last round. By performing several injections in the same byte of the input state the attacker can infer exactly the byte of the key used to encrypt the corrupted byte of $\widetilde{c}$ and then, by extending the procedure to remaining bytes of the input state the attacker can infer the entire key. This attack has been extended in [16] where the same procedure can be applied to any round (instead of the last round), thus making the attack much simpler to be deployed.

A different attack involves the complete blanking or the manipulation of the S-Box lookup tables, achievable through resetting or reconfiguring the memory where they are stored [17], [18]. In this way the attacker is able to immediately recover the last round key. This attack has been proved to be feasible on a number of microcontrollers where the S-Box was stored in the internal flash memory or where the AES algorithm was accelerated on an FPGA device.

A last family of attacks to AES targets the key schedule that is the process executed to generate round keys from the user supplied key [15], [19]. These attacks exploit the regular structure of the key schedule to infer bytes of the key through corrupting one or more bytes during the generation of the last round key. Although effective when the key schedule is precomputed and its results are stored in some permanent memory, this attack is not feasible when the generation of the round keys is performed on the fly.

Our technique represents an effective and lightweight countermeasure against fault attacks targeting the round logic of AES. Fault attacks targeting the content of the S-boxes are not covered by our proposal.

B. The SHA-2 Algorithm and DFA Attacks to SHA-2

SHA-2 (Secure Hash Algorithm 2) is a widely employed hash function still largely suggested by NIST [20]. SHA-2 is built upon the Davies–Meyer structure: the input message m is split into fixed-length blocks m_i (applying padding if necessary) and goes through a number of rounds. At each ith round, block message m_i is fed into a block-cypher as the round key, while h_{i-1}, which is the output of the previous round, is fed in the block cypher as the input message. Each round produces the cyphertext c_i that is then x-ored with h_{i-1} to obtain the round output h_i. In the first round, when there is no previous output value, a constant pre-specified initial value h_0 is employed. SHA-2 ensures by construction that given two messages m_1 and m_2, the corresponding digests d_1 and d_2 are the same if and only if m_1 and m_2 are the same. For this reason, SHA-2 is employed to ensure *integrity* (no malicious manipulation) of transmitted/received data. Moreover, in case a secret key is added to the input message of SHA-2 (*keyed has function*), not only integrity of the message is ensured, but also *authentication* of the sender of the message.

Attacks to SHA-2: Given the relatively short time from its publication, few works presenting fault attacks to SHA-2 have been yet published. Given a message m, a secret key k and digest d obtained by processing $m + k$ with SHA-2, the goal of the attacker is to modify m into m' and generating the corresponding d' without knowing k so that m' will be accepted as an authentic message from the receiver.

The very first DFA-based attack to SHA-2 has been proposed in [21] where random single faults are injected in the same round of the algorithm during the computation of the digest associated with a given message. All the "corrupted"

digests, together with the nominal one, are then used to infer the internal state of the sponge at the injected round, thus retrieving the secret key. This attack has been extended in [22] where up to 16 bits are corrupted.

More recently fault attacks have been combined with SAT problem formulation in the so called *Algebraic Fault Analysis* [23], [24]. These approaches, still based on the injection of faults during the execution of the algorithm, allow to retrieve the secret key requiring a significantly reduced amount of corrupted digests, thus speeding-up the attack.

The protection technique here presented represents an effective and lightweight countermeasure against all the discussed fault attacks families against SHA-2.

C. Threat Model

In this paper, we assume that the attacker knows the functionality implemented by the circuit and has a high-level knowledge of the architecture of the circuit. On the other hand, we assume that the attacker has no detailed knowledge of the circuit layout, e.g., position of the flip-flops. We assume single bit-flip attacks because this represents the worst case scenario for our detection methodology. Should the attacker be able to carry out a multiple bit-flips attack, the proposed methodology would increase (or at least maintain) the detection capabilities. Thus, the analysis in Section V only reports results from such worst-case single bit-flip injection scenario.

III. Applying Partial replication in AES

As we previously discussed, AES performs a set of logical operations for a given number of round iterations. Some of the values used within the loop (the state) are rotated in between each loop iteration. These loop logic operations can be replicated either fully or partially. As a consequence, if partial replication is adopted, only a subset of n-bits of the state are replicated. Since all the operations involved in the loop are logical operations, the n-bits selected for replication do not depend on the remaining bits.

Not all AES internal operations can be protected without adding a big overhead to the original design. For this reason, these operations are not duplicated on our protection mechanism. One unprotected operation in our implementation of AES is the subBytes operation. This operation is not protected as it needs a lookup table to do the substitution of bytes and we do not want to replicate all this table. The other operation is the initial *addRoundKey* phase. It cannot be protected as it directly uses the input key and the state of the main module (i.e. *aes_128*). For this reason, we, hereby, use the term partial replication as we are not replicating the full AES circuitry.

The *shiftRows* operation is used in the nine rounds of the algorithm plus the final round. It is performed along with the *mixColumns* operations and the *addRoundKey* operation. Therefore, as these three operations are executed at the same time, they are protected together. The protection of these operations is performed through n-bit replicas of the original signals and replicating the XOR operation of the last three rows with a constant as the pseudo-code in listing 1 shows.

```
 1 // Partial copy of the state columns
 2 s0_rc = state_in[(96 + MOD_SIZE - 1)..96]
 3 s1_rc = state_in[(64 + MOD_SIZE - 1)..64]
 4 s2_rc = state_in[(32 + MOD_SIZE - 1)..32]
 5 s3_rc = state_in[(MOD_SIZE - 1)..0]
 6
 7 // Divide output of the subBytes operations by cells
 8 p = {p00, p01, p02, p03}
 9     {p10, p11, p12, p13}
10     {p20, p21, p22, p23}
11     {p30, p31, p32, p33}
12
13 // Perform shiftRows and mixColumns to compute the
14 // state_out signal
15 z0_rc = p00 XOR p11 XOR p22 XOR p33 XOR k0_rx
16 z1_rc = p03 XOR p10 XOR p21 XOR p32 XOR k1_rc
17 z2_rc = p02 XOR p13 XOR p20 XOR p31 XOR k2_rc
18 z3_rc = p01 XOR p12 XOR p23 XOR p30 XOR k3_rc
```

Listing 1. Protected *mixColumns*

At the end of the each iteration of the algorithm, the output of the conventional AES circuit is compared with the partial replica. If they do not match, there has been -at least- a fault (flip-flops, wires, logic) in one of the two circuits. In this case, the circuit continues execution and, at the end of the encryption/decryption, a (pseudo-) random text is forwarded to the output of the circuit to make both differential fault analysis and side-channel attacks unfeasible. This forwarding (selection) logic needs to be carefully designed and protected, e.g., through obfuscation, not to become a weakness.

IV. Applying Partial replication in SHA-2

In our implementation of SHA-2, there are four sets of rounds (20 iterations per round) that perform logical operations over the values. At the end of each round, the values for the next round are computed. Consequently, we apply partial replication at the iteration level. We cannot apply it at the round level as left rotation operations are used (and the replica does not have all the bits).

Listing 2 shows a pseudo-code of the protected operations for SHA-2 where the replicated signals are denoted as the name of the original signals concatenated with the suffix *_rc* (for replica circuit). We are able to replicate from 1 to 32 bits (being 32 the width used in the original circuit)

A. Rotary mechanism for higher fault coverage in SHA-2

Partial replication, as we will see in short, has a detection capability proportional to the number of bits replicated. In this section, we introduce a rotary mechanism so that every arbitrary number of iterations, we stop protecting the lower order n-bits and we protect (i.e. replicate) the next n-bit.

From the example in Figure1, we can see that the rotary mechanism rotates taking n-bits to the left in each rotation, where n is the number of bits to protect. The mechanism implemented does not loop around when reaching the highest order bit (i.e. it does not operate with a chunk of the high-order bits and a chunk of the low-order bits). This has been done because it is a considerable overhead for the replica to correctly operate non-consecutive bits. For this reason, the rotary mechanism protects all the possible chunks of n bits

```
1 for counter_val in 0 to 79
2   if (counter_val >= 0 AND counter_val <= 19)
3     f_rc = (b_rc and c_rc) or ((not b_rc) and d_rc)
4     k_rc = 0x5A827999 // Take last n bits
5   else if (counter_val >= 20 AND counter_val <= 39)
6     f_rc = b_rc xor c_rc xor d_rc
7     k_rc = 0x6ED9EBA1 // Take last n bits
8   else if (counter_val >= 40 AND counter_val <= 59)
9     f_rc = (b_rc and c_rc) or (b_rc and d_rc) or (c_rc
          and d_rc)
10    k_rc = 0x8F1BBCDC // Take last n bits
11  else if (counter_val >= 60 AND counter_val <= 79)
12    f_rc = b_rc xor c_rc xor d_rc
13    k_rc = 0xCA62C1D6 // Take last n bits
14
15  temp_rc = (a leftrotate 5) + f_rc + e_rc + k_rc + w[i]
16  e_rc = d_rc
17  d_rc = c_rc
18  c_rc = b leftrotate 30
19  b_rc = a_rc
20  a_rc = temp
```

Listing 2. Protected SHA-2

going from the less significant to the most significant bit until the next chunk is smaller than n. In this particular case, the rotary mechanism uses the n high-order bits and it resets the rotary mechanism to start again from bit zero. An alternative to this implementation would be to forget about the last chunk if it does not have n-bits. Yet, this reduces the protection capabilities of the scheme (and specially, the high-order bits).

Figure 1 shows an example of how this rotation mechanism would work using a replication value (n=14) value non-multiple of the register width for the first three rounds. The fourth round and the following are not represented as they will copy the same behaviour (i.e. the fourth round behaves exactly as the first round shown).

Finally, when an error is detected, the SHA-2 circuitry applies the same procedure described for AES in section III.

V. Experimental Results

A. Experimental Environment

We implemented the conventional AES and SHA-2 algorithms; and, we extended both implementations with the proposed protection mechanism. We implemented a VHDL-level fault injector to measure fault detection capabilities. We run the fault injection experiments on a server equipped with one AMD EPYC 7401P (24-Cores) running at 3GHz with 128 GB of DDR4. To measure the overhead introduced by the proposed protection mechanism we synthesized prototypes of the nominal and protected AES and SHA-2 circuits using the Virtex UltraScale+ FPGA (model: xcvu13p-fhga2104-3-e) as a target device. We used Xilinx Vivado™as a development and synthesis environment and we then employed the accompanying Device Utilization Summary and Power Estimator tools for area, power and time analysis.

B. Effectiveness Analysis

The first analysis we carried out aimed at measuring the effectiveness of the proposed countermeasure in detecting faults in the conventional AES and SHA-2 circuits and the impact of the chosen number of protected bits on the achieved fault detection capability. We considered a worst case attack scenario, where the attacker is able to attack a single flip-flop as this will certainly produce the most statistically significant results for the attack. Note though, that the technique compares the outputs of the conventional circuit with the output of the n-bit replica. Therefore, any fault caused in the wires or any multiple-bit fault that generates a change in the outputs will also be detected by the proposed solution.

Table I
Number of required simulations per confidence and error margin value

Confidence	99%			95%		
Error Margin	10%	5%	1%	10%	5%	1%
SHA-2	589	2354	58846	341	1363	34068
AES	6709	26839	670951	3885	15538	388429

When dealing with simulation, statistically significant results are usually understood as 99% confidence level at 5% error margin. Based on this target, the authors in [25] defined the number of experiments (based on the input signals) needed to reach the selected goal. Table I reports the number of simulations (i.e. different combinations of input values and fault location and clock cycle) needed to achieve the required statistical significance. In this paper, we report the results for 99% confidence and 1% error margin. Consequently, we performed 671K simulations for each replica size (thus, 671K * 32 in total) for the AES algorithm (which took almost 42 hours to complete in our server when using the batch mode of Vivado™) and 60K simulations for each replica size for SHA-2.

Critical Fault Identification: Due to logical and temporal masking, not all faults lead to a modification of the output (i.e. an error). Moreover, when the circuit works in the field, the percentage of masked faults will even increase due to electrical masking (this cannot be simulated with Vivado™). From a differential fault analysis point of view, *critical* faults are only those that eventually propagate to the output of the circuit. By using the fault injection tool, we measured the amount of critical faults for the considered designs[1]. It is worth noting that the AES algorithm when unfolded (i.e. each round has each own copy of the circuitry), just 10% of the faults generate an erroneous output (as most of the faults fall in unused parts at that precise moment). Yet, for the folded version of AES, this number goes up to 88%. In the SHA-2 implementation used 73.5% of the faults injected result in an erroneous output. These faults (now errors) are the ones that have been used to assess the effectiveness of the proposed detection technique.

Error Detection Capability Analysis: In order to analyze the error detection capability of the proposed technique, we implemented several versions of the protected AES and SHA-2, each with a different number of protected bits. For each given number of protected bits we performed a set of fault injection experiments in which only the previously identified errors have been considered.

[1] We did not inject faults in the replica circuits because this does not bring any advantage to the DFA attacker.

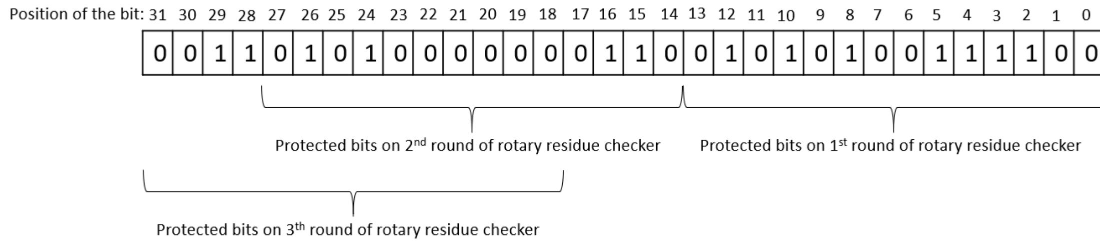

Figure 1. Example of the rotary residue checker protecting 14 bits

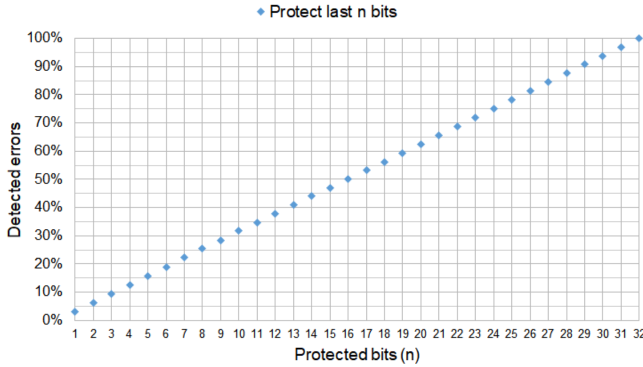

Figure 2. Error detection w.r.t. the number of protected bits for AES

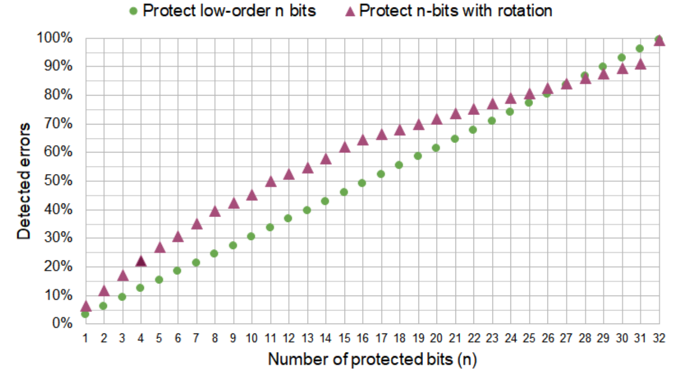

Figure 3. Error detection w.r.t. the number of protected bits for SHA-2

Table II
FAULT DETECTION AND OVERHEAD FOR THE AES PROTECTION

	Fault Detection	Overhead		
		Area	Power	Frequency
Our Approach	100.00%	8.00%	0.81%	0.00%
Mestiri [8]	85.96%	14.45%	N.A.	49.02%
Chu [9]	100.00%	81.99%	N.A.	0.17%

Table III
FAULT DETECTION AND OVERHEAD FOR THE SHA-2 PROTECTION

	Fault Detection	Overhead		
		Area	Power	Frequency
Our Approach	100.00%	3.88%	12.31%	0.00%
Luo [10]	83.60%	27.05%	48.69%	15.62%
Bayat [11]	100.00%	4.40%	N.A.	N.A.
Mestiri [12]	99.99%	68.37%	N.A.	1.32%

Figures 2 and 3 report the percentage of detected errors as a function of the number of replicated bits. For both, AES and SHA-2 the detection capability is proportional to the number of replicated bits. It reaches 100% when replicating all the bits

In the case of SHA-2, Figure 3 also plots the detection capabilities of the rotary mechanism. In this case, the rotary mechanism proves to be useful when the number of protected bits is small. For example, a 50% detection capability is achieved with 11 bits (with the rotary mechanism) but it needs 16 bits for the baseline replication mechanism. Curiously, the baseline outperforms the rotary mechanism between 28 and 31 bits. This is caused by the alternate protection of the high-order n-bits or low-order n-bits which leaves the high-order 32-n bits or low-order 32-n bits unprotected at a time (whereas for the baseline, it is always the 32-n high-order bits). Thus, the rotary mechanism is unable to detect all faults that appear in the low-order bits (as they will be 50% of the time unprotected) plus, being able to protect the high-order 32-n bits 50% of the time does not compensate it.

C. Efficiency Analysis

After assessing the error detection capability of the proposed partial replication, we also measured the introduced overhead in terms of area occupation and power consumption increase and working frequency reduction. We considered the worst-case scenario (w.r.t. the introduced overhead) which is the case of a complete 32-bit replication. Moreover, we compared the results achieved by our technique with several recent related proposals. Tables II and III report the results of this analysis for AES and SHA-2, respectively.

The protected AES requires 7.81% more LUTs than the unprotected design (and also 12 more CLB registers and 15 extra Bounded IOB (not included in Table II). The pseudo-random number generator (PRNG) adds an extra 0.51% area overhead over the unprotected AES; making the total are overhead 8%. We measured the power consumption increase by averaging the all the performed simulations. The unprotected AES dynamic power consumption is about 492 mW while for the protected AES it is 496 mW, thus resulting in a 0.82% overall power consumption increase. The PRNG, which is always on to avoid side-channel analysis is responsible of 50% of the power consumption increase. Finally, the protected AES was able to work at the same maximum frequency (136 MHz) as the unprotected circuit, thus no working frequency reduction is caused by our proposal.

The protected SHA-2 requires 1.85% more LUTs than the base design, 1 more F8 MUX and 17 extra Bounded IOB (not included in Table III). The PRNG adds an extra 2% area overhead over the unprotected SHA-2. Moreover, we left the rotary mechanism in place for a 32-bits protection. Nevertheless, the rotary mechanism does not introduce any area overhead being it a matter of wiring. The dynamic power consumption of the unprotected SHA-2 is 195mW while it is 207mW and 209mW for the protected SHA-2 without and with the rotary mechanism, respectively. Therefore, the worst-case power overhead is 12.31% (5.13% caused by replication, 1.03% by the rotary mechanism; and 6.15% by the PRNG). Finally, the protected SHA-2 works at the same maximum frequency (74 MHz) as the unprotected circuit.

Finally, as shown in Tables II and III, the proposed protection detects 100% of the critical fault attacks for both AES and SHA-2 overperforming the most recent related proposals in terms of area, power and working frequency impact.

VI. Security Analysis and Limitations

Our proposal protects SHA-2 from all possible fault attacks and AES from all the fault attacks targeting the round logic while S-box values are not protected. In case these are computed on the fly, they do not represent a threat, as discussed in [16]. On the other hand, in case these values are precomputed and stored, they can be effectively exploited for DFA. Therefore, the memory that stores these values must be protected.

A severe threat is related to the output forwarding logic associated with our proposal. Indeed, this represents a "common-cause of failure": by exploiting multiple fault injections the attacker may produce an error in the nominal algorithm and then induce the forwarding logic to forward the corrupted value instead of the random one. This scenario may be solved by applying circuit obfuscation/camouflaging to the forwarding logic, making fault injection harder.

A final hypothetical threat would be the case where the attacker is able to inject faults in both the nominal and replicated algorithm and then induce the two replicas to generate the same output. In such case, our solution would totally fail. Nevertheless, we argue that, although it is possible for the attacker to inject faults in both replicas, is is totally unfeasible to make them generate the same output.

VII. Conclusions

We presented a lightweight protection mechanism based on partial redundancy to detect fault attacks in HW accelerators implementing AES and SHA-2 (which are two widely used NIST standards) based on partial redundancy. The presented results demonstrated that selectively replicating portions of the circuits allow to achieve full detection of the critical fault attacks. The overhead introduced is 8.32% for AES and 3.88% for SHA-2 in terms of area, 0.81% for AES and 12.31% for SHA-2 in terms of power with no working frequency reduction. Moreover, a comparative analysis showed that our proposal outperforms the most recent related countermeasures.

References

[1] J. Katz and *et al.*, *Handbook of applied cryptography*. CRC press, 1996.

[2] M. Joye and *et al.*, *Fault analysis in cryptography*. Springer, 2012, vol. 147.

[3] T. W. Arnold and *et al.*, "Ibm 4765 cryptographic coprocessor," *IBM Journal of Research and Development*, vol. 56, no. 1.2, pp. 10–1, 2012.

[4] R. Karri and *et al.*, "Fault-based side-channel cryptanalysis tolerant rijndael symmetric block cipher architecture," in *Proceedings 2001 IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems*, Oct 2001, pp. 427–435.

[5] P. Maistri and *et al.*, "A novel double-data-rate aes architecture resistant against fault injection," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007)*, Sep. 2007, pp. 54–61.

[6] G. Bertoni and *et al.*, "Error analysis and detection procedures for a hardware implementation of the advanced encryption standard," *IEEE Trans. on Computers*, vol. 52, no. 4, pp. 492–505, April 2003.

[7] G. Bertoni and *et al.*, "An efficient hardware-based fault diagnosis scheme for aes: performances and cost," in *19th IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems, 2004. DFT 2004. Proceedings.*, Oct 2004, pp. 130–138.

[8] H. Mestiri and *et al.*, "A high-speed aes design resistant to fault injection attacks," *Microprocessors and Microsystems*, vol. 41, pp. 47–55, 2016.

[9] J. Chu and *et al.*, "Error detecting aes using polynomial residue number systems," *Microprocessors and Microsystems*, vol. 37, no. 2, pp. 228–234, 2013.

[10] P. Luo and *et al.*, "Concurrent error detection for reliable sha-3 design," in *2016 Int. Great Lakes Symposium on VLSI (GLSVLSI)*. IEEE, 2016, pp. 39–44.

[11] S. Bayat-Sarmadi and *et al.*, "Efficient and concurrent reliable realization of the secure cryptographic sha-3 algorithm," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 7, pp. 1105–1109, July 2014.

[12] H. Mestiri and *et al.*, "Efficient countermeasure for reliable keccak architecture against fault attacks," in *2017 2nd Int. Conf. on Anti-Cyber Crimes (ICACC)*, March 2017, pp. 55–59.

[13] NIST-FIPS, "Announcing the advanced encryption standard (aes)," *Federal Information Processing Standards Publication*, vol. 197, no. 1-51, pp. 3–3, 2001.

[14] J. Blömer and *et al.*, "Fault based cryptanalysis of the advanced encryption standard (aes)," in *Int. Conf. on Financial Cryptography*. Springer, 2003, pp. 162–181.

[15] C. Giraud, "Dfa on aes," in *Int. Conf. on Advanced Encryption Standard*. Springer, 2004, pp. 27–41.

[16] A. Barenghi and *et al.*, "Fault attack on aes with single-bit induced faults," in *2010 Sixth Int. Conf. on Information Assurance and Security*. IEEE, 2010, pp. 167–172.

[17] D. Ziener and *et al.*, in *2018 Int. Conf. of BRAM-based AES Implementations on FPGAs*, Dec 2018, pp. 1–7.

[18] Y. Zhang and *et al.*, "Persistent fault injection in fpga via bram modification," in *2019 IEEE Conf. on Dependable and Secure Computing (DSC)*, Nov 2019, pp. 1–6.

[19] C.-N. Chen and *et al.*, "Differential fault analysis on aes key schedule and some countermeasures," in *Australasian Conf. on Information Security and Privacy*. Springer, 2003, pp. 118–129.

[20] NIST-FIPS, "Announcing approval of federal information processing standard (fips) 180-2, secure hash standard; a revision of fips 180-1," *Federal Information Processing Standards Publication*, 2002.

[21] N. Bagheri and *et al.*, "Differential fault analysis of sha-3," in *Int. Conf. on Cryptology in India*. Springer, 2015, pp. 253–269.

[22] P. Luo and *et al.*, "Differential fault analysis of sha3-224 and sha3-256," in *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Aug 2016, pp. 4–15.

[23] S. Nejati and *et al.*, "Algebraic fault attack on sha hash functions using programmatic sat solvers," in *Int. Conf. on Principles and Practice of Constraint Programming*. Springer, 2018, pp. 737–754.

[24] P. Luo and *et al.*, "Algebraic fault analysis of sha-3 under relaxed fault models," *IEEE Trans. on Information Forensics and Security*, vol. 13, no. 7, pp. 1752–1761, July 2018.

[25] "Chapter 3 - architectural vulnerability analysis," in *Architecture Design for Soft Errors*, S. Mukherjee, Ed. Burlington: Morgan Kaufmann, 2008, pp. 79 – 120.