Hamming-code based fault detection design methodology for block ciphers

F.E. Potestad-Ordóñez[1], E. Tena-Sánchez[1], R. Chaves[2], M. Valencia-Barrero[1], A.J. Acosta-Jiménez[1], C.J. Jiménez-Fernández[1]

[1] Instituto de Microelectrónica de Sevilla/Universidad de Sevilla (IMSE-CNM-CSIC/US)
[2] INESC-ID, IST, Universidade de Lisboa
Email: {potestad,erica,manolov,acojim,cjesus}@imse-cnm.csic.es,{ricardo.chaves}@inesc-id.pt

***Abstract*—Fault injection, in particular Differential Fault Analysis (DFA), has become one of the main methods for exploiting vulnerabilities into the block ciphers currently used in a multitude of applications. In order to minimize this type of vulnerabilities, several mechanisms have been proposed to detect this type of attacks. However, these mechanisms can have a significant cost or not adequately cover the implementations against fault attacks. In this paper a novel approach is proposed, consisting in generating the signatures of the internal state using a Hamming code. This allows to cover a larger amount of faults allowing to detect even or odd bit changes, as well as multi-bit and multi-byte changes, the ones that make ciphers more vulnerable to DFA attacks. As case of study, this approach has been applied to the Advanced Encryption Standard (AES) block cipher implemented on FPGA using T-boxes. The results suggest a higher fault coverage with an overhead of 16% of resource consumption and without any penalty in the frequency degradation.**

Keywords*—Fault Attack, AES, Countermeasure, FPGA implementation, Hamming Code, DFA.

I. Introduction

While the existing encryption standards have proven to be mathematically secure, there are techniques that allow to attack the physical implementation of such algorithms. These attacks include Side Channel Analysis (SCAs) and Active Faults Analysis Attacks. In the first case, the attacker tries to obtain information from the cryptographic algorithm during encryption in a passive way (measuring its power consumption or electromagnetic emissions among others). The most known SCAs are the Differential Power Analysis (DPA) and Correlation Power Analysis (CPA). In the second case, the attacker tries to manipulate the circuit in a non-permanent way to generate transient operating errors (faults) and thus obtain the secret information contained by the device. This paper focuses on Differential Fault Analysis (DFA) and non-invasive active fault injections, due to their effectiveness and the real threat they pose to the security of cryptographic devices.

DFA has led the international community to focus on the study and development of techniques to prevent these types of attacks. The so-called countermeasures or detection schemes try to minimize the vulnerabilities of cryptocircuits against the numerous attack techniques. There are therefore numerous proposals for detection schemes reported in the literature, such as hardware redundancy [1], temporal redundancy [2], information redundancy [3] or the combination between them [4], [5]. Within the different schemes, in this paper we will focus on information redundancy schemes. The main contribution of the paper is the proposal of a methodology for designing a fault detection scheme using the Hamming codes. The proposed approach is particularly targeted at block cipher implementations taking advantage of memory blocks for the data transformation. With the use of Hamming codes it is possible to protect the data being processed in a more comprehensive way at a lower cost. As case of study, the Advanced Encryption Standard (AES) block cipher [6] is considered, using a T-boxes based implementation. Note that, while T-Box based implementations on software tend to be vulnerable to timing attacks, hardware based implementations are not and result in particularly efficient designs. The obtained implementation results suggest that using this methodology it is possible to detect both even/odd-bit and single/multi-byte faults, thus protectecting against DFA attacks like [7]–[10], at a cost of 16% LUT usage increase and without affecting the original achievable frequency.

The rest of the paper is organized as follows. Section II introduces the AES block cipher and related T-Box implementations. Section III presents a overview of the main DFA attacks and related countermeasures. Section IV describes the proposed detection approach and its application to the AES algorithm. Section V presents the obtained experimental results on a FPGA, discussing the fault coverage, area and performance impact, and state of the art comparison. Final concluding remarks are presented in Section VI.

II. AES algorithm

The AES cipher is the NIST standard, selected to replace the DES, using the Rijndael algorithm [6] with 128 bit input blocks. Depending on the key size (128, 192, or 256 bits) AES performs the input transformation over multiple rounds (10, 12, or 14, respectively). The round process consists of processing the (16 byte) input state S though the operations SubBytes(), ShiftRows(), MixColums() and AddRoundKey(), as illustrated in Figure 1a) for a 128 bit key.

The SubBytes() operation is a non-linear transformation that replaces one byte with another. The ShiftRows() function rotates each of the bytes of state S to the left 0, 1, 2 or 3 positions as a function of the row of state S. The MixColumn() transformation multiplies each column of the state S by a fixed polynomial. Finally, the AddRoundKey() operation performs the bitwise-XOR operation between the state S and the expanded key of that round.

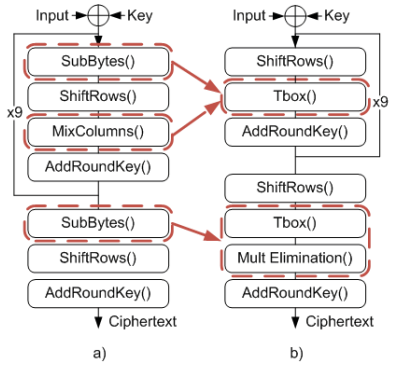

Fig. 1. Schematic representation of AES: a) Standard, b) T-box based.

Hardware implementation wise, the computation can be performed by discrete logic operations or by lookup tables, mapping part of the computation to memory blocks. This mapping can range from performing part of the bytes substitution operation to perform both the bytes substitution and part of the MixColumn operation, using T-Boxes [6], [11], as illustrated in Figure 1. Note that, T-Box implementations are possible since the byte substitution operation is independent of the byte position, allowing the ShiftRows() to be performed before the SubBytes(). Particularly in FPGA devices, the use of T-Box based implementations can take advantage of existing embedded memory blocks, allowing for more efficient designs [12].

While the vulnerability of T-Box based implementations against fault injection attacks has not been deeply analyzed, the S-box based implementations have been analyzed suggesting different vulnerabilities [7]–[10], as described in the next section.

III. DFA State of the Art

The DFA technique is one of the most influential techniques within the field of cryptanalysis. It consists of recovering the secret key by the mathematical analysis of outputs generated by the cipher in both its correct and faulty mode of operation.

The two main works on DFA applied to AES are those presented by Giraud [7] and Dussart [8]. The first one establishes that if an attacker is able to inject a single bit-level fault into the state matrix before the operations of the ninth round, namely, during the operations of the eight round, it is possible to recover the secret key. The latter one describes how to recover the key by injecting different faulty bits in the same byte of the state matrix after the seventh round and before the ninth operation of MixColumn(). Another recently DFA attack on AES is presented in [9], called Incremental Fault Analysis (IFA) attack and represents a good example of the continuous interest in designing new techniques to attack the implementations. Another theoretical attack on the KeySchedule() matrix is also described in [7], where by injecting a complete faulty byte in the last rounds it is possible to determine the secret key. Other techniques based on similar fault models need the injection of a single byte or multiple faulty bytes in the state matrix or KeySchedule() in the last rounds to compromise the security of the AES cryptocircuit [10].

The reported vulnerabilities of the AES cipher show that the faults injected during the SubBytes() and MixColumn() operations represent a very important leakage of information. Since these operations are performed through T-boxes, the reported vulnerabilities are extensible to this type of implementations with memories and therefore must be protected.

In regard to fault coverage some approach are limited, detecting only odd or even bit faults, while other more complete solutions are able to cover single or multi-byte faults.

Considering these types of fault coverage, different countermeasure approaches can be found in the literature, namely: hardware redundancy, which consists in duplicating the whole or part of the cryptographic circuit and running it in parallel; or temporal redundancy, which consists in repeating cryptographic operations or encrypt/decrypt every data. In both cases the duplicated result is then compared to check its correctness. Hardware redundancy schemes are those that offer the greatest protection, since they can detect any type of fault injected, however, the cost in area tend to be very high. For example, in [1] a hardware redundancy scheme is proposed that is able to detect all type of faults but at area overhead of 86%. Temporal redundancy schemes have the lowest area degradation, however they present high throughput degradation. In [2] a more efficient solution is presented with an additional 7% resource usage and a frequency degradation of 17%, but the fault coverage only considers single bit faults. Information redundancy schemes are those that add some type of information to the processed data, such as parity bits. This type of approach presents the lowest area overhead and the lowest performance degradation, depending on the design technique. For example, in [3] a scheme is proposed with an area overhead of 8% but it is only able to detect odd faulty bits. Other solution consider a combination of these approaches, such as the one presented in [4], able to detect all type of faults but at a 38% resource usage increase, or the presented in [5] that use the polinomial residue number systems (PRNS) with redundant AES modules to detect the faults.

Approaches exploiting information redundancy tend to be more efficient, presenting the best tradeoff between area/performance penalties, however sometimes at a cost of lower fault coverage [3], [13]–[18]. The solutions presented in [3], [13]–[15], only protect against odd faulty bits. For example, in [3] one parity bit per byte in the S-box() and two checkers are implemented. In [13] a similar approach with one parity bit is used, but in this case, the scheme implements a backup redundant component to correct the possible faults, being also applicable to the KeySchedule(). In [14] different cyclic redundancy checkers over $GF(2^8)$ are implemented with the penalty that they are not able to detect even-bit or byte faults. In [15] an implementation of the S-box() using composite fields to improve the robustness of the S-box() is presented, needing a parity bit to protect the rest of the cipher operations. The scheme presented in [16] is an improvement of [15] where the scheme is able to detect even-bit faults but not single/multi bytes faults. On the other hand, the solutions proposed in [17] and [18] are able to detect even/odd-bit and single/multi-byte faults. In [17] the fault detection is based on linear predictors improved with (n, k) codes and robust cubic network. Finally, in [18] the scheme protects only the S-box() by relating the input byte to the output byte by means of a

difference and using a checker. For the rest of the processes they use a parity bit. This scheme tends not to detect all possible even-bit faults.

IV. Proposed detection scheme

DFA attacks can target different point of the computation by injecting different types of faults, from single even or odd bit faults to single and multi-bytes.

Therefore, one of the main consideration when designing a fault detection scheme is to consider the possibility of detecting faults during the data transformations. Taking this into account, the proposed approach is based on the fact that the more information is known about the data being processed, the harder to modify them without being detected.

Our proposal does not consider the vulnerabilities presented by these works on the KeySchedule(), using the transformation rounds as application example, although our detection scheme is equally applicable to protect the KeySchedule().

A. Hamming code as signature generator

The proposed scheme is based on the Hamming code as a signature generator of the data used by the cipher. The Hamming codes are well known as a family of linear error-correcting codes invented by Richard W. Hamming in 1950 [19]. The main use of these codes is to detect faults in the data transmission and correct them. Within the fault detection context, it is possible to use these codes to obtain as much information as possible about the data being processed. The use of Hamming codes to protect the AES cipher is not new. In [20], it is used to correct errors in the state matrix when a parity bit detects an error. In the proposed approach Hamming codes are used as a signature generator of the data being processed throughout the entire round.

Hamming codes add additional bits to the original data, that are able to detect/correct errors. With these codes, it is possible to protect d bits, by adding m bit to the original message, resulting in a total of n bits. k is the maximum number of bits that could be encoded. Equations (1) to (4) illustrate the Hamming code characteristics.

$$Hamming\ Code\ (n,k) \tag{1}$$
$$d \leq k \tag{2}$$
$$k = 2^m - m - 1 \tag{3}$$
$$n = 2^m - 1 \tag{4}$$

For example, to protect 8 bits of information using 4 additional bits, it has $d = 8$ and $m = 4$. Therefore $k = 11$, and thus 11 bits of information are encoded resulting in a total data length of $n = 15$. Then $d \leq k$ and therefore with $m = 4$ extra bits is enough to protect the data.

By applying a Hamming code as a signature generator for the fault detection in the T-box based AES, and using the signatures depicted in (5), it is possible to obtain a signature composed of 4 additional bits (M_{0-3}) to protect 8 bits of data (D_{0-7}). M_{0-3} denotes the 4 bit signature added to the 8 bit of processed data.

$$\begin{aligned} M_0 &= D_0 \oplus D_1 \oplus D_3 \oplus D_4 \oplus D_6 \\ M_1 &= D_0 \oplus D_2 \oplus D_3 \oplus D_5 \oplus D_6 \\ M_2 &= D_1 \oplus D_2 \oplus D_3 \oplus D_7 \\ M_3 &= D_4 \oplus D_5 \oplus D_6 \oplus D_7 \end{aligned} \tag{5}$$

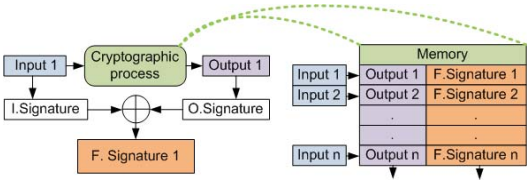

Fig. 2. Representation of the signature calculation and memory usage.

By applying the signature generation on the input data of a cryptographic operation ($I.Signature$) and on the output of this same operation ($O.Signature$), it is possible to merge and check the join signature ($F.Signature$) using a XOR operation, as depicted by (6):

$$F.Signature = I.Signature \oplus O.Signature \tag{6}$$

With this approach it is only required to store and test a single 4 bit value for each byte to be protected. This is the only additional value to be stored, as described in the next section, in order to check if the value was corrupted by a fault injection during the cryptographic operation, as depicted in Figure 2.

B. Scheme application to AES T-box

The proposed solution is well suited to T-Box based implementations, in particular when targeting FPGA devices with embedded memories. These embedded memories blocks (BRAMs) output upto 18 or 36 bits, depending on the selected device and technology.

The T-box() has an 8-bit input and outputs 32 bits. This 32-bit output is the result of the Galois multiplication of the constant values with the value obtained after the S-box() operation, as depicted in Figure 1b. For the encryption the 4 constant values are $\{1, 1, 2, 3\}$ and $\{9, e, b, d\}$ for the decryption. For simplicity sake, the following discussion only focuses on the implementation for the fault detection for the encryption process. Nevertheless, the following solution can be directly applied for decryption. In the following discussion the output of the T-box(), depicting the constant multiplication of the S-box() output, is represented by $\{1S, 1S, 2S, 3S\}$ for the encryption.

As described in [12], the 4 bytes of each AES T-box() output can be combined to derive the S-box() output (both in encryption and in decryption), has:

$$\begin{aligned} 1S \oplus 1S &= 0 \\ 2S \oplus 3S &= 1S \\ 1S \oplus 1S \oplus 2S \oplus 3S &= 1S \end{aligned} \tag{7}$$

With this, instead of generating a Hamming code to protect 32 bits, which would require a high number of protection bits, only a 8-bit value needs to be considered, namely $1S$.

Knowing the value of the T-box() input, it is possible to generate the signature of the input and Xored with the $1S$ signature, obtaining the final signature using (6). The final signature is added to each data contained by the BRAM, as it is shown in Figure 2 and is checked after the T-box() process. This means that 4 extra bits of information are needed for each byte of the State. Given that the available BRAMs already output 18 or 36 bits, this additional redundant information can

TABLE I. COMPARISON WITH DIFFERENT DETECTION SCHEMES.

Solution	Type of redundancy protection	Area Overhead	Frequency Degradation	Type of Fault detected				Technology
				Odd Bit	Even Bit	Single byte	Multi Byte	
Unprotected	none	1	1	✗	✗	✗	✗	Spartan 6
[3]	Information	1.08	0.70	✓	✗	✗	✗	Virtex 1000
[13]		1.44	NIA	✓	✗	✗	✗	NIA
[14]		1.73	0.64	✓	✗	✗	✗	NIA
[15]		1.40	NIA	✓	✗	✗	✗	NIA
[16]		1.32	0.97	✓	✓	✗	✗	Virtex II
[17]		1.77	0.86	✓	✓	✓	✓	Virtex E
[18]		1.25	0.88	✓	✓	✓	✓	Virtex 5
Proposed		**1.16**	**1**	✓	✓	✓	✓	**Spartan 6**
[1]	Hardware	1.87	1	✓	✓	✓	✓	Virtex 5
[2]	Temporal	1.07	0.83	✓	✗	✗	✗	Virtex 4
[4]	Combination	1.38	0.78	✓	✓	✓	✓	Virtex 6
[5]		1.58	0.83	✓	✓	✗	✗	Spartan 3

[1] NIA=No Information Available.

be stored for free. The only additional area cost comes from the signature generation and comparison logic.

With this approach, and considering (7) and (8), it is possible to cover the computation of the T-box(), namely any fault produced during the memory access/read (denoted by e), and in the input/output of the T-box() process.

$$\begin{aligned} &(1S \oplus 1S) \oplus e \neq 0 \\ &(2S \oplus 3S) \oplus e \neq 1S \\ &(1S \oplus 1S \oplus 2S \oplus 3S) \oplus e \neq 1S \end{aligned} \tag{8}$$

V. RESULTS

In order to properly evaluate the proposed solution, the fault detection approach was applied to the T-Box based AES design descried in [12] and implemented on a Xilinx Spartan 6 XC6SLX75 FPGA using Xilinx ISE 14.7. This implementation uses the devices dual port BRAMs, allowing to map two T-Box() functions into two single memory block, each mapping 18 bits of the output.

A. Fault coverage

In order to test the fault detection proposal, different fault simulation tests have been implemented where even/odd-bit and single/multi-byte faults have been considered. The faults have been injected at the input of the T-box(), inside the T-box(), and in the output data of the T-box(). To perform the fault simulations, a fault is injected during the encryption process in each test and an error signal is activated in case of fault detection. All the injected faults were detected by our scheme.

B. Comparison with other schemes

In order to properly compare and evaluate the cost of the proposed solution, differential metrics were obtained, presenting the ratio between the unprotected and protected implementations. The obtained results show a cost of 645 Slice Registers and 720 LUTs for the proposed protected solution. In regard to the unprotected design, this represents a total resource usage increase of 25.5% more Slice registers and 16% more LUTs, and no additional BRAM usage. This is achieved with no frequency degradation nor increase in the number of computation cycles, i.e. achieving the same throughput. While the amount of registers increases more than the LUTs, LUTs are the resources imposing the overall device occupation (since more LUTs than registers are required). Particularly, the signature calculation and fault detection logic, depicted in Figure 2 requires 55 LUTs per each 4 bytes, i.e. 13 LUTs per state byte.

In order to compare the proposed method with those proposed in the most relevant state of the art, the relative cost and provided fault coverage of these solutions are summarized in Table I. This table depicts the area overhead, frequency degradation, and type of faults detected for the AES implementations. This comparison mostly considers the information redundancy solutions. For completeness, hardware, temporal, and combined redundancy schemes are also presented. The type of faults detected by each solution is divided by $Odd\ Bit$, $Even\ Bit$, $Single\ Byte$ and $Multi\ Byte$. The ability to detect each type of fault is represented by a check or cross mark, respectively.

In regard to information redundancy approaches, the solution in [3] presents the lowest impact in terms of area, but has a small fault coverage and imposes a significant performance degradation. The solutions proposed in [17] and [18], provide a good fault coverage but at a higher resource usage and with a performance degradation. In particular the scheme in [18] is able to detect even/odd-bit and single/multi-byte faults and it only protects the S-box() computation. Hardware redundancy approaches such as the one in [1] provide a complete fault coverage without impacting the performance, but almost doubles the required hardware resources.

VI. CONCLUSIONS

This paper proposes a novel approach for signature generation using a Hamming code to protect block ciphers. This approach has been implemented into the AES implementation supported by T-Boxes on a reconfigurable device. The obtained results suggest that the proposed solution can be deployed with a LUT overhead of about 16% and without any performance degradation. The fault coverage evaluation indicates that the proposed scheme is able to detect all the types of faults targeted by DFAs: even and odd faulty bits in the same or in different bytes. Finally, the comparison with the related state of the art suggests that the proposed solution is able to provide a higher fault coverage at a overall significant lower resource cost, and without a throughput degradation.

ACKNOWLEDGMENTS

This work was supported by the projects: Erasmus+, COST Action CRYPTACUS, INTERVALO (TEC2016-80549-R), FCT (Fundação para a Ciência e a Tecnologia, Portugal) and the ERDF (European Regional Development Fund, EU) through the projects FCT: UIDB/50021/2020 and LISBOA-01-0145-FEDER-031901 (PTDC/CCI-COM/31901/2017, HiPErBio).

REFERENCES

[1] M. Joye, P. Manet, and J.-B. Rigaud, "Strengthening hardware AES implementations against fault attacks," *IET Information Security*, vol. 1, no. 3, pp. 106–110, 2007.

[2] J. Rajendran, H. Borad, S. Mantravadi, and R. Karri, "Slide-based concurrent error detection technique for symmetric block ciphers," *IEEE International Symposium on Hardware Oriented Security and Trust (HOST'10)*, pp. 70–75, 2010.

[3] K. Wu and M. Goessel, "Low Cost Concurrent Error Detection for the Advanced Encryption Standard," in *International Test Conference (ITC'04)*, pp. 1242 – 1248, 2004.

[4] R. Karri, K. Wu, P. Mishra, and Y. Kim, "Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 21, no. 12, pp. 1509–1517, 2002.

[5] J. Chu and M. Benaissa, "Error detecting AES using polynomial residue number systems," *Microprocessors and Microsystems*, vol. 37, no. 2, pp. 228 – 234, 2013.

[6] J. Daemen and V. Rijmen, "AES proposal: Rijndael," 1999.

[7] C. Giraud, "Dfa on aes," in *International Conference on Advanced Encryption Standard*, pp. 27–41, 2004.

[8] P. Dusart, "Differential Fault Analysis on A.E.S.," *Applied Cryptography and Network Security (ACNS'03)*, pp. 293–306, 2003.

[9] T. E. Pogue and N. Nicolici, "Incremental fault analysis: Relaxing the fault model of differential fault attacks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019.

[10] G. Piret and F. Quisquater, "A Differential Fault Attack Technique against SPN Structures, with Application to the AES," *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'03)*, pp. 77–88, 2003.

[11] V. Fischer and M. Drutarovský, "Two methods of rijndael implementation in reconfigurable hardware," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES'01)*, pp. 77–92, 2001.

[12] R. Chaves, G. Kuzmanov, S. Vassiliadis, and L. Sousa, "Reconfigurable memory based AES co-processor," *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS'06)*, p. 8, 2006.

[13] L. Breveglieri, I. Koren, P. Maistri, and P. Milano, "Incorporating Error Detection and Online Reconfiguration into a a Regular Architecture for the Advanced Encryption Standard," *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05)*, pp. 72 – 80, 2005.

[14] C.-H. Yen and B.-F. Wu, "Simple Error Detection Methods for Hardware Implementation of Advanced Encryption Standard," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 720–731, 2006.

[15] M. M. Kermani and A. Reyhani-Masoleh, "Parity-based fault detection architecture of S-box for advanced encryption standard," *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'06)*, pp. 572–580, 2006.

[16] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A lightweight high-performance fault detection scheme for the advanced encryption standard using composite fields," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 1, pp. 85–91, 2011.

[17] M. Karpovsky, K. J. Kulikowski, A. Taubin, and S. Member, "Robust Protection Against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard," *International Conference on Dependable Systems and Networks (DSN'20)*, pp. 93 – 101, 2004.

[18] H. Mestiri, N. Benhadjyoussef, M. Machhout, and R. Tourki, "High performance and reliable fault detection scheme for the advanced encryption standard," *International Review on Computers & Software (IRECOS'13)*, vol. 8, no. 3, pp. 730–746, 2013.

[19] R. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[20] C. Moratelli, F. Ghellar, E. Cota, and M. Lubaszewski, " A fault-tolerant, DFA-resistant AES core," *IEEE International Symposium on Circuits and Systems (ISCAS'08)*, pp. 244 – 247, 2008.