IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 1, 2022, accepted June 12, 2022, date of publication June 16, 2022, date of current version June 23, 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3183764*

# Design and Evaluation of Countermeasures Against Fault Injection Attacks and Power Side-Channel Leakage Exploration for AES Block Cipher

**F. E. POTESTAD-ORDÓÑEZ[1,2], E. TENA-SÁNCHEZ[1,2], A. J. ACOSTA-JIMÉNEZ[1,3], C. J. JIMÉNEZ-FERNÁNDEZ[1,2], AND RICARDO CHAVES[4], (Senior Member, IEEE)**

[1]Seville Institute of Microelectronics IMSE-CNM (CSIC/US), 41092 Sevilla, Spain
[2]Departamento de Tecnología Electrónica, Escuela Politécnica Superior, University of Seville, 41011 Sevilla, Spain
[3]Departamento de Electrónica y Electromagnetismo, Facultad de Física, University of Seville, 41012 Sevilla, Spain
[4]INESC-ID, IST, Universidade de Lisboa, 1049-001 Lisboa, Portugal

Corresponding author: F. E. Potestad-Ordóñez (potestad@imse-cnm.csic.es)

This research was funded by Grant PID2020-116664RB-I00 funded by MCIN/AEI/10.13039/501100011033.

**ABSTRACT** Differential Fault Analysis (DFA) and Power Analysis (PA) attacks, have become the main methods for exploiting the vulnerabilities of physical implementations of block ciphers, currently used in a multitude of applications, such as the Advanced Encryption Standard (AES). In order to minimize these types of vulnerabilities, several mechanisms have been proposed to detect fault attacks. However, these mechanisms can have a significant cost, not fully covering the implementations against fault attacks or not taking into account the leakage of the information exploitable by the power analysis attacks. In this paper, four different approaches are proposed with the aim of protecting the AES block cipher against DFA. The proposed solutions are based on Hamming code and parity bits as signature generators for the internal state of the AES cipher. These allow to detect DFA exploitable faults, from bit to byte level. The proposed solutions have been applied to a T-box based AES block cipher implemented on Field Programmable Gate Array (FPGA). Experimental results suggest a fault coverage of 98.5% and 99.99% with an area penalty of 9% and 36% respectively, for the parity bit signature generators and a fault coverage of 100% with an area penalty of 18% and 42% respectively when Hamming code signature generator is used. In addition, none of the proposed countermeasures impose a frequency degradation, in respect to the unprotected cipher. The proposed work goes further in the evaluation of the proposed DFA countermeasures by evaluating the impact of these structures in terms of power side-channel. The obtained results suggest that no extra information leakage is produced that can be exploited by PA. Overall, the proposed DFA countermeasures provide a high fault coverage protection with a low cost in terms of area and power consumption and no PA security degradation.

**INDEX TERMS** Countermeasure, FPGA implementation, Hamming code, parity, AES, DFA, fault attack, power analysis.

## I. INTRODUCTION

Nowadays, the algorithms used to ensure the privacy of user data have proven to be mathematically secure due to the fact that compromising their security would be extremely time and resource consuming. Taking into account recent research work on the different stages of Internet of Things (IoT) security solutions, it is possible to observe the importance of IoT security analysis [1]. One of the proposals in [2], [3] is the use of so-called lightweight cryptography to avoid the security problems and the use of solutions whose cost has the lowest possible impact on implementations. Finally, [4] discusses the risks that exist if attacks on embedded applications used in this field are not taken into account. Because of this, the

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

development of new cryptographic algorithms and solutions to protect information and comply with the strong restrictions imposed by the applications is constant. However, trying to compromise the security of such algorithms through their physical implementations has become a serious problem. To this end, attacks on the physical implementations of these devices are increasing in number. These attacks include Side Channel Analysis (SCAs) and Active Faults Analysis Attacks (FA).

In the first case, the attacker tries to obtain information from the cryptographic algorithm during encryption passively (measuring its timing [5], power consumption [6] or electromagnetic emissions [7] among others). The most known SCA are the Power Analysis attacks [6], [8], being powerful attacks due to their effectiveness and low cost. In a Power Analysis (PA) attack, the power consumption of the cryptographic device is measured during the normal encryption process. Then, the measured power consumption and the data (plaintext, ciphertext, or available data in the specific application) are mathematically processed to statistically obtain the secret key and thus the secret information.

In the second case, the attacker tries to manipulate the circuit with the possibility of using different degrees of invasion, being more or less aggressive with the device [9]–[21]. The most popular type of attack, because it is cheap and allows the device to be manipulated without leaving a trace and without causing permanent damage, is the active non-invasive one. In this case, the attacker is able to generate transient operating errors (faults) and thus obtain the secret information contained in the device. These types of faults combined with the mathematical formulation used to retrieve secret information are known as Differential Fault Analysis (DFA) attacks.

DFA has become one of the main methods for compromising the security of block ciphers widely used in a multitude of environments and applications. Due to their great effectiveness and the real threat they pose to the security of cryptographic devices and therefore to the integrity of user data, the scientific community has turned to proposing different mechanisms for detecting this type of attack. Each of these mechanisms has a significant cost or does not adequately protect cipher implementations against numerous DFA models. The so-called countermeasures or detection schemes try to minimize the vulnerabilities of cryptocircuits against different attack techniques. There are therefore numerous proposals for detection schemes reported in the literature, such as hardware redundancy [26], temporal redundancy [27], information redundancy [28] or the combination between them [29]. Within the different schemes, in this paper, we will focus on information redundancy scheme group.

However, despite protecting systems against DFA attacks, there are combined attacks where both vulnerabilities against fault injection and power leakages are exploited [22], [23]. This also makes the evaluation of the interaction of DFA countermeasures against PA attacks indispensable, since the extra operations for DFA countermeasures can potentially increase the leaked information exploitable by PA attacks [25]. In this sense, it is mandatory to evaluate the impact of the added DFA countermeasures in the robustness against power analysis attacks.

Towards this, this paper proposes four methodologies for designing fault detection while accessing the side-channel attack resistance impact, targeting main standard encryption algorithm used to ensure the privacy of user data, the Advanced Encryption Standard (AES) [30]. The main contribution of this work are the four methodologies for designing fault detection solutions using Hamming codes and parity bits as signature generators.

These solutions allow detecting fault injections at both bit and byte level, both odd and even faults. Four protection schemes have been designed, which allow to protect the cipher partially or completely. Partial protection is carried out on the T-boxes, while full protection is carried out on the operations, intermediate state matrix and KeyAddition. With the use of Hamming codes and parity bits as signature generators, it is possible to protect the data being processed in a more comprehensive way at a lower cost. The proposed approaches are particularly targeted at AES block ciphers based on T-box implementation, taking advantage of memory blocks for data transformation. Nevertheless, the proposed solutions are extensible to other block ciphers with memory-based implementations, which while vulnerable to timing attacks on software are secure and particularly efficient in hardware-based implementations. The resulting proposals are then evaluated in terms of side-channel attack resistance, more concretely against PA attacks, evaluating the potential leaked information through Test Vector Leakage Assessment (TVLA) [31]. The obtained results, suggest fault coverages between 98.5 to 100%, with no particular degradation in terms of exploitable information leakage by PA attacks.

The rest of the paper is organized as follows. Section II introduces the T-box based AES block cipher implementations, presents a description of DFA attacks reported in the literature, and also describes the type of faults against which implementations should be protected. Section III describes the proposed detection approaches and their application to the AES algorithm. Section IV introduces the setup validation schemes used. Section V presents the experimental results obtained, discussing the fault coverage both by simulation and experimentally. In Section VI presents the performance evaluation and comparison with other schemes considering area and performance cost. Section VII presents the exploration of power consumption leakages exploitable in PA attacks. Finally, in Section VIII the conclusions of this work are depicted.

## II. STATE OF THE ART

The AES [30] cipher is the National Institute of Standards and Technology (NIST) standard, selected to replace Data Encryption Standard (DES) cipher, using the Rijndael algorithm. Depending on the key size (128, 192, or 256 bits)

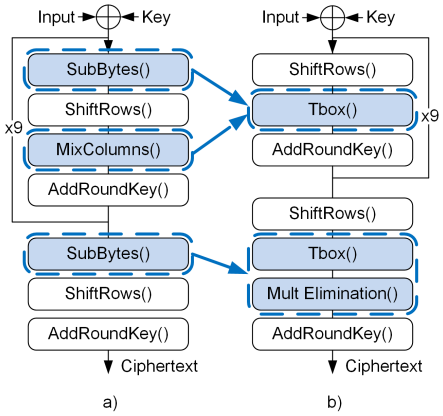


**FIGURE 1.** Schematic representation of AES: a) Standard, b) T-box based.

AES performs the input transformation over multiple rounds (10, 12, or 14, respectively). The round process consists of processing the 128 bit input block (16 byte), called state S, through the operations SubBytes(), ShiftRows(), MixColums(), and AddRoundKey(). Figure 1a) depicts the data transformation when a 128-bit AES key is used.

AddRoundKey()- Performs the Exclusive-OR (XOR) operation between the state matrix and the expanded key of each round. SubBytes()- In this function the transformation of the state matrix with the Sbox8 of the AES byte by byte is applied. ShiftRows()- This function rotates each row of the state matrix. MixColumns()- Multiplies the internal state by a fixed data matrix defined by the algorithm.

In the case of T-box based AES implementations, depicted in Figure 1b), it can be seen that the SubBytes() and MixColumns() operations are implemented by the T-box() operation and the ShiftRows() operation is performed at the beginning of the round. In the last round, since there is no MixColumn() operation, the T-box() operation and an operation called Mult Elimination() must be performed, which consists of eliminating the part corresponding to the MixColumn() operation from the T-box(). These types of implementations are oriented to the use of Field Programable Gate Array (FPGA) memories where the T-box() operation are mapped.

### A. DFA ON AES

Within the field of cryptanalysis, DFA is one of the most popular techniques for compromising the security of encryption algorithms. An example of this is its continued application to most proposed encryption algorithms [32]–[35]. Its principle of operation is based on combining fault injection with mathematical analysis of the effect produced by these faults on the operation of the ciphers. In a DFA, the objective is to capture the encrypted data when the cipher works correctly, correct ciphertext, and then, using the same secret key, to inject faults and capture the encrypted data when the cipher works incorrectly, called faulty ciphertext. This process of obtaining a faulty ciphertexts is repeated several times (depending on the type of DFA), injecting the fault in the same clock cycle for the same encryption/decryption process (identical key and plaintext), which allows to obtain multiple faulty cipher outputs. Following this, by mathematically comparing the correct and faulty samples, it is possible to extract enough information to be able to calculate the internal secret of the cryptographic device. To inject the faults exploitable by DFA, all analyses establish a set of assumptions by which attacks must be carried out, for example, the need to inject a fault in a given bit, in a given operation and at a specific time. Due to the specific assumptions needed for different DFA attacks, the mathematical formulations required to establish the relationship between the faulty ciphertexts and the correct data are diverse. In order to minimize this vulnerability, numerous solutions have been proposed to protect the circuits, as summarized below.

Since the active fault attack presented by Bonet et al [36], targeting the Rivest-Shamir-Adleman (RSA) cryptosystem, several DFAs have been widely used in many cryptographic algorithms. One typical target is the AES algorithm, due to the fact that it is a NIST standard and is widely used. It has been subjected to numerous differential analyses and attacks with the aim of compromising its security [9]–[21]. These works have shown the possibilities of compromising the security of the AES cipher using fault attacks. This cipher vulnerabilities can be classified according to the point of attack, namely the state matrix, the S-box() and the KeySchedule(). Note that, when the point of attack refers to the state matrix, it means that the fault is introduced in the matrix when it is modified between each of the processes carried out by the cipher, namely SubBytes(), ShiftRow(), MixColumns(), and AddRoundKey(). Another important concept is the moment in the computation where the fault is introduced, i.e. the round in which the cipher operates. Table 1 lists the different types of attacks reported in the literature where the DFA on the AES cipher is carried out. This table classifies the DFAs reported in the literature according to the operation on which the attack is carried out, the type of attack and the round where the attack is performed.

When getting the state matrix as a point of attack ([10]–[13], [16], [20], [21]), single-bit, multi-bit, single-byte, and multi-byte faults can be considered. In this case, the attacks must be carried out after the seventh round, and in the case of [13] the authors state that it can be carried out in any of them. According to, if a single-bit or single-byte fault is inserted in the state matrix, before the SubBytes() operation of the ninth round, only 50 faulty ciphertext are needed to recover the key. This vulnerability was experimentally shown by inserting glitches into the cipher clock signal [9]. In this case, the attacker must be able to repeat the fault in three different positions within the same byte using the same plaintext. It is also possible to attack the MixColumn() operation in the seventh, eighth and ninth rounds [11]. For the attack made on the MixColumn() operation in the eighth round, 20 pairs of correct/faulty ciphertext are necessary. For the attack on the

**TABLE 1. DFA on AES classification.**

| Reference | Attack on State | Attack on SubByte() | Attack on KeySchedule() | Type of Attack | Attack in Round |
|---|---|---|---|---|---|
| [10] | ✔ | ✘ | ✔ | S.Bit \| S.Byte | 8 and 9 |
| [11] | ✔ | ∼ | ✘ | M.Bit (Same Byte) | 7, 8 and 9 |
| [12] | ✔ | ✔ | ✘ | S.Byte | 8 |
| [13] | ✔ | ∼ | ✘ | S.Byte | 9 |
| [14] | ✘ | ✘ | ✔ | S.Byte | 8 and 9 |
| [15] | ✘ | ✘ | ✔ | S.Byte | 8 |
| [16] | ✔ | ✘ | ✔ | S.Byte | 7, 8 and 9 |
| [17] | ✘ | ✘ | ✔ | M.Byte | 9 |
| [18] | ✘ | ✘ | ✔ | M.Byte | 9 |
| [19] | ✘ | ✘ | ✔ | M.Byte | 9 |
| [20] | ✔ | ✘ | ✘ | S.Byte \| M.Byte | 8 |
| [21] | ✔ | ✘ | ✘ | S.Byte \| M.Byte | 8 |

∼ = Maybe applicable
S.Bit and M.Bit = Single Bit and Multi Bit respectively.
S.Byte and M.Byte = Single Byte and Multi Byte respectively.

MixColumn() operation in the ninth round, 40 to 50 different pairs of correct/faulty ciphertext are needed.

When the SubByte() operation is targeted, the DFA presented in [12] could compromise the security, while the DFAs presented in [11], [13] the authors do not test the analysis but claim that it would also be applicable. In these cases, the faults needed by the DFAs are multi-bit or single-byte. As in the case of the state matrix, the attack should be performed from the seventh round onward.

If the attack is performed over KeySchedule(), it is possible to consider the DFAs presented in [10], [14]–[19]. In these cases, the faults types needed by the DFAs are single-bit, multi-bit, single-byte, or multi-byte from the seventh round onwards.

These vulnerabilities can also be extended to the T-box implementations, since the T-boxes implement the MixColumns() and SubBytes() operations in lookup memories. As such, it also need to be considered and protected.

### B. FAULT DETECTION SOLUTIONS IN THE LITERATURE

In order to minimize the vulnerability introduced by DFA, different fault detection solutions have been proposed in the literature, presenting different protection levels. These countermeasures can be classified in three main groups, namely hardware redundancy, temporal redundancy, information redundancy, or a combination between them. To evaluate the effectiveness of each specific countermeasure, fault coverage is used as a metric to determine its level of protection. Fault coverage is the number of faults detected based on the type and number of attacks performed. It should be noted that attacks can be of different types, depending on the type of fault to be achieved, e.g., single bit, multiple bits, single byte, etc.

Hardware redundancy consist on duplicating the whole hardware or part of it in order to implement a double process of encryption or decryption for error detection. This approach typical offers the highest fault coverage, being able to detect all type of faults (unless the same fault is introduced at the same point of the computation in both implementations). However, these solutions tend to have very high area and energy consumption costs, potential reaching a 100% increase. For example, the solution proposed in [26] is able to detect all types of faults but imposes an area overhead of 86%.

Temporal redundancy consists of repeating the whole or part of the encryption or decryption process to check if any fault was injected. This approach presents the lowest area degradation, but at the cost of more processing time, resulting in high-throughput degradation. In [27] a more efficient solution is presented with an additional 7% resource usage and a throughput degradation of only 17%, but the fault coverage only considers single bit faults.

Information redundancy works by providing additional information to the data during the encryption or decryption process. This extra information, such as parity bits, allows to identify if faults were injected during the process. These approaches tend to provide the lowest overhead on the area and the lowest performance degradation. However, this lower impact often results in a lower level of fault coverage. In [28] the proposed solution implies an area overhead of only 8% but it is only able to detect odd faulty bits. The solutions proposed by [37] only protects against odd faulty bits. On the other hand, the solutions presented in [28], [38], [43], only protect against odd faulty bits, while the countermeasure proposed in [39], [40] are able to detect all type of faults (even/odd-bit and single/multi-byte faults, but their fault coverage percentages do not reach 100%, being 90% in the first case and 98% in the second. Furthermore, both solutions do not protect against possible faults in the KeyAddition operation, leaving the system exposed to DFA attacks that exploit this.

## III. PROPOSED FAULT DETECTION SOLUTIONS

The following describes the three proposed fault detection solutions. As described above, multiple types of faults can be used to target the security of the AES cipher, from the bit level to the byte level. Therefore, the proposed solutions aim to detect as many faults as possible, based on the idea that the more information is known about the processed data, the

greater the possibility of detecting whether a fault has been injected or not. Note that, the solutions herein proposed focus on the encryption process, in particular, on round transformations. However, these proposed solutions can be equally applied to the decryption process and KeySchedule().

### A. HAMMING CODE AS A SIGNATURE GENERATOR

This approach is based on the use of Hamming code as a signature generator for the data being ciphered. The Hamming codes are well known as a family of linear error-correcting codes invented by Richard W. Hamming in 1950 [41]. 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. 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 characteristics of the Hamming code.

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

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.

The use of Hamming codes to protect the AES cipher is not new. In [42], it is used to correct errors in the state matrix when a parity bit detects an error.

Herein, we propose the Hamming codes to generate a signature of the data being processed throughout the entire round, by applying a Hamming code as a signature generator for the fault detection in the T-box based AES.

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.

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

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

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

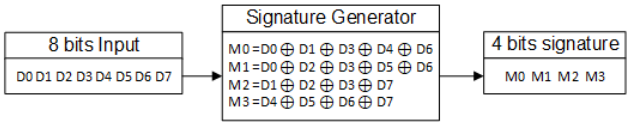


**FIGURE 2.** Representation of hamming code signature generator.

Figure 2 shows a schematic representation of the signature generator based on Hamming code, where the 4-bit signature is obtained from 8 input bits. With this approach, one only needs to store and test a single 4 bit value of final signature, which relates the information of the input and output of the encryption process for each byte to be protected. This is the only additional value that must be stored to check if the value was corrupted by a fault injection during the cryptographic operation. Note that this approach is only able to detect a fault, not correct them.

### B. PARITY BITS AS A SIGNATURE GENERATOR

This approach is based on the use of parity bits as a signature generator for the data used during the cipher. The parity bit methodology has been widely used in a multitude of applications for error detection [46], [48], [49]. Its use as fault injection countermeasures applied to the AES cipher can be found in the works [28], [47]. The parity bit adds extra bits to the data in order to determine how many *ones* are present in the data. Depending on the number of extra bits that are added, it is possible to implement one type of parity or another. For example, if only one parity bit is added, odd type errors can be detected. If a larger number of parity bits are added, it is possible to detect even type faults. The limitation of the parity bit schemes is that if only one parity bit is used and, for example, two bits change at the same time, the parity bit generated by the corrupted data will be the same as the one produced by the uncorrupted data, and therefore the fault is filtered.

Herein, the use of parity bits is proposed in order to generate a signature of the input and the output of the processed data by the T-box(). Note that, rather than implementing a traditional parity bit, a specific method to generate a signature is used. Moreover, by using memories to implement the T-boxes, the addition of the signature is much less expensive. In the proposed solution, the generated signatures are compressed into four bits and embedded in the memory. As in the Hamming-code approach, the aim is to know all possible information from the processed data and store bits. Using four extra bits, two for encoding the input data and two for encoding the output data, it is possible to generate a signature that allows to know if the data has been modified during the encryption processes. If the eight bits of the input data of the T-box() are denoted by $D_{0-7}$ and the eight bits of the output data are denoted by $S_{0-7}$, it is possible to generate a signature using (7) denoted by $M_{0-3}$. The four bits of the signature are added to the T-box memory. Figure 3 shows a schematic representation of the signature generator based on parity bits,

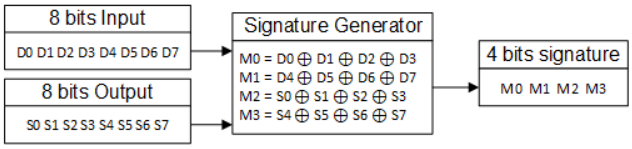


**FIGURE 3.** Representation of parity bit signature generator.

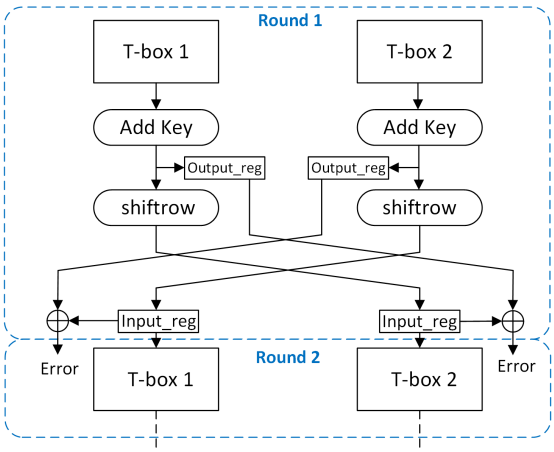


**FIGURE 4.** Schematic representation of the intermediate states protection. ShiftRow application.

where the 4-bit signature is obtained from 8 input bits and 8 output bits of the memory.

$$\begin{aligned} M_0 &= D_0 \oplus D_1 \oplus D_2 \oplus D_3 \\ M_1 &= D_4 \oplus D_5 \oplus D_6 \oplus D_7 \\ M_2 &= S_0 \oplus S_1 \oplus S_2 \oplus S_3 \\ M_3 &= S_4 \oplus S_5 \oplus S_6 \oplus S_7 \end{aligned} \tag{7}$$

Unlike the Hamming code approach, this solution does not require the generation of a final signature from the input data signature and the output data signature, since the four bits of the final signature are obtained directly from $M$.

### C. INTERMEDIATE STATES AND KEY ADDITION PROTECTION

Given that the two previous approaches only protect the T-box() operation, additional work is needed to protect the complete round computation, by protecting the intermediate states and the key addition. To do this, a protection scheme has been designed to detect faults in the remaining stages of the cipher process. This is achieved by combining the use of intermediate registers and adding parity bits.

To protect the intermediate matrix states, intermediate registers have been used that allow, using a comparator, to know if there has been a fault injection. If the value of the state matrix is changed during the round operations, it will be detected. As an example, in Figure 4 it is possible to see how this approach is applied to the state matrix when the ShiftRow() operation is performed.

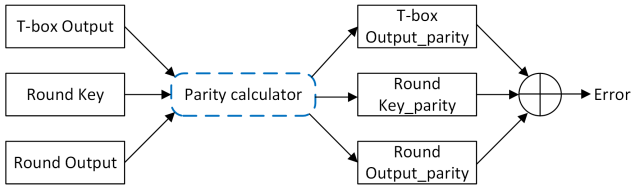


**FIGURE 5.** Schematic representation of the key addition protection.

On the other hand, to know if the key addition operation is attacked, parity bit checkers have been added to know if the key or the state matrix has been modified. With this, it is possible to determine if faults have been inserted, since the parity of the matrix and the parity of the round key must be the same as the parity of the round output. The proposed solution is depicted in Figure 5.

### D. APPLICATION ON A T-BOX BASED AES

The solutions proposed above are well suited to T-box based implementations, in particular when targeting FPGA devices with embedded memories. These embedded memories blocks (BRAMs) output from 18 up to 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 Sbox() operation, as depicted in Figure 1b. In the case of 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. However, the following solution can be applied directly for decryption. In the following discussion, the output of the T-box(), depicting the constant multiplication of the output of the S-box(), is represented by {1S, 1S, 2S, 3S}.

In the case of the Hamming approach, instead of generating a Hamming code to protect 32 bits, which would require a high number of protection bits, only an 8-bit value needs to be considered, namely $1S$. As described in [50], the 4 bytes of each AES T-box() output can be combined to derive the S-box() output (both for encryption and decryption), has:

$$\begin{aligned} 9S \oplus eS \oplus bS \oplus dS &= 1S \\ 1S \oplus 1S \oplus 2S \oplus 3S &= 1S \end{aligned} \tag{8}$$

Knowing the value of the input of the T-box(), it is possible to generate the signature of the input and Xored with the $1S$ signature, thus obtaining the final signature using (6). This final signature is added to each data value stored in the BRAM.

In the case of the parity signature scheme, instead of calculating the parity for the entire 32 bits, only the 8 bits of the T-box() input and the 1S output combination need to be considered. Using (7), it is possible to generate the signatures composed of four bits that are added to the data contained by the BRAM.

In both cases, this means that only 4 extra bits of information are needed for each byte of the State. Given that the available BRAMs already outputs 18 or 36 bits, rather than the needed 32 (or $2 \times 16$ bits), this additional redundant information can be stored for free. The only additional area cost comes from the signature or parity generation and comparison logic.

With these approaches, it is possible to cover the entire computation of the T-box(), i.e. any fault produced during the memory access/read (denoted by $e$) and in the input/output of the T-box() process.

## IV. EXPERIMENTAL SETUPS

The fault detection solutions have been applied to an T-box based AES design [50]. The resulting implementations have been tested against fault injection attacks both in simulation and experimentally. Furthermore, to evaluate whether the inclusion of the proposed fault attack countermeasures result in additional information leakage, power sice-channel analysis attacks were also performed.

In order to carry out each test, different experimental measurement setups were used. Each of the setups allows to independently evaluate the fault coverage (by simulation and experimentally), to analyse the performance of each of the implementations, and finally to evaluate the negative impact with regard to power side-channel attacks. To carry out each test, different experimental setups were used, as described below. Each setup allows to independently evaluate the fault coverage (by simulation and experimentally), to analyse the performance of each of the implementations, and finally to evaluate the negative impact regarding power side-channel attacks. Notice that two different FPGAs are used in setups (2) and (3,4), since the evaluation platforms use different boards, optimized for each type of analysis. This evaluation, described in more detail in the following sections, was performed using:

1) The fault coverage by simulation has been performed using fault injection simulations with Xilinx ISE 14.7 ISim and Matlab.
2) The experimental fault coverage evaluation setup, depicted in Figure 6, is composed of a NewAE CW1173 ChipWhisperer-Lite, the target board NewAE CW305 with a Xilinx Artix 7 XC7A100T FPGA, a PicoScope 3205D oscilloscope, and a personal computer with a Core-i5 processor with 8 GB of RAM and Matlab software.
3) The frequency and area costs evaluation have been obtained using a Xilinx Spartan 6 XC6SLX75 FPGA under Xilinx ISE 14.7.
4) The evaluation of the power side-channel analysis was performed using the Sakura-G board with a Xilinx Spartan 6 XC6SLX75 FPGA, a PicoScope 3205D oscilloscope, and a personal computer with a Core-i5 processor with 8 GB of RAM, depicted in Figure 7.

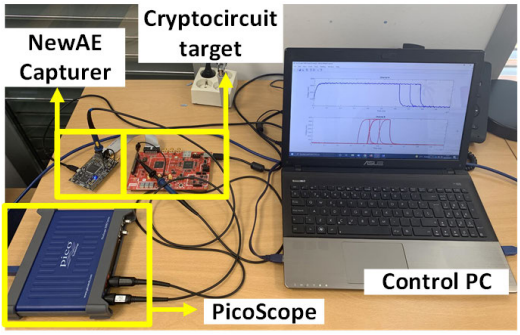


**FIGURE 6.** Attack setup with the capturer, cryptotarget, PicoScope and control PC.

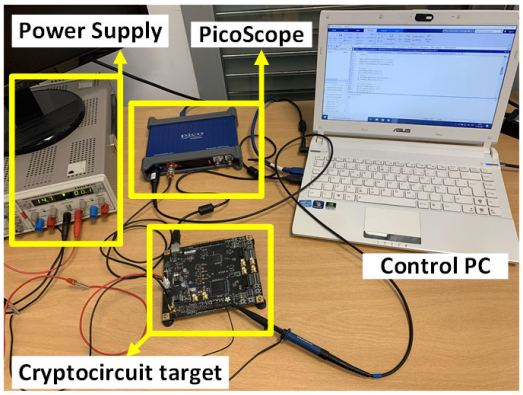


**FIGURE 7.** Experimental setup for TVLA analysis: SAKURA-G board, PicoScope and control PC.

The proposed solutions were used to derive four different protected AES implementations. The first protection structure consists of the protection offered by the Hamming code signature generator covering only the T-box. The second protection structure consists of the parity bit signature generator that also targets only the T-box. The third structure is composed of the combination of the Hamming code signature generator and the protection of the intermediate states along with the key addition (complete AES). Finally, the fourth structure consists of the parity bit signature generator together with the intermediate states protection and the key addition (complete AES). For simplicity sake, these structures are denoted as: Hamming, Par_sig, Hamming_Total and Par_sig_Total, respectively.

All the tests carried out have been performed using random keys and plain texts, to ensure that the obtained results do not depend on these inputs.

## V. FAULT COVERAGE

### A. SIMULATION-BASED ANALYSIS

In order to test the fault coverage of the proposed solutions, different fault injection simulations have been performed, testing different types of fault, from single-bit, single-byte, multi-bit, multi-byte, and random. On the one hand, fault

detection was performed using Xilinx ISE 14.7 ISim software with functional and post-routed simulations.

The simulation of fault injections was performed using Matlab, considering the four protection structures described above, where the faults are injected during cipher operations. These tests take into account the fault injections on the intermediate state matrix, cipher round operations, and KeyAddition() process. Since the space of possible faults is $2^{256}$ (128 bits of the state matrix and 128 bits of the keyAddition()) and their analysis would be very time consuming, we have reduced the possible faults to useful and exploitable faults for DFA. These are all possible faults of the single/multiple bit type in the same byte and in different bytes, both even and odd, and single/multiple byte.

In addition, random type faults were also considered, although it should be noted that this type of fault does not fall within those exploitable by DFA, since there must be some control in the fault injection process in order to take advantage of the fault. For example, a random fault can range from a single bit to the change of the whole state matrix or a large part of it; the latest case is not useful since they cannot be exploited by DFA. Algorithm 1 represents the code used to inject the faults over the T-box(). The code requires two inputs, the attack position, in the example the $T - box$ and the type of the attack, single-bit, single-byte, multi-bit, multi-byte, or random.

**Algorithm 1** Attack to the AES T-Box Based

**Require:** Attack_Position = $\{T - box\}$
**Require:** Attack_Type
  **return** Cipher_Text = {Final_state}
  $M_{0-127}$ = State, $B_{0-127}$ = Input_Text, $K_{0-127}$ = Key
  AddRoundKey (B,K)
  **for** $i = \{1\}$ *to* $\{9\}$ **do**
    ShiftRow(M)
    **if** Attack_Position != 0 **then**
      T-box(M) $\bigoplus$ Attack_Type
    **else**
      T-box(M)
    **end if**
    AddRoundKey(M)
  **end for**
  ShiftRow(M)
  T-Box(M)
  MultElimination(M)
  AddRoundKey(M)
  Final_State = M

1.5 million tests have been carried out for each countermeasure, in a total of 6 million tests, using Matlab. For the Hamming_Total, Par_sig, and Par_sig_Total countermeasures all faults were detected, both odd and even, bit and byte type and random faults. Among all the random faults, we have been able to see that the faults where, in their majority, bits in the state matrix that were changed but not detected. Among all random faults, it should be noted that the possible space for a random fault ranges from a single bit to all 128 bits of the matrix. Therefore, there are cases where faults occur where the vast majority or all bits of the state matrix change. It should be noted that these types of fault are not exploitable by DFA and are not useful for cryptanalysis). While these cases are not detected by the proposed countermeasures, they are not useful in a real case and can therefore be discarded. Nevertheless, out of 1.5 million tests, these cases occur only a very few times. Regarding the Hamming countermeasure, out of the 1.5 million tests, only one fault was not detected. This fault, of odd type (specifically the fault 14680064 - in decimal), injected during the T-box() operation, generates the same signature at the output of the T-box() as the correct data, and thus cannot be detected. It should be noted that the Hamming_Total countermeasure, which extends the Hamming countermeasure, is able to detect this fault.

From these data, it can be estimated that the fault coverage of the Hamming_Total, Par_sig and Par_sig_Total approaches is 100% while the Hamming approach is 99.99%, given the particular case mentioned above.

### B. EXPERIMENTAL ANALYSIS

To better evaluate the effectiveness of fault coverage of the proposed approaches and to further validate them, an experimental attack setup was used to obtain experimental data for each solution. This set-up consists of a capture unit and a target FPGA board from NewAE. In this case, the fault injection system has been designed using Python. The considered fault injection attack is based on clock signal modification. This technique has been widely used since it was introduced in [51] and consists of injecting a small pulse into the main clock signal, and therefore it is possible to violate the setup and the hold times of the cipher components allowing to flip the internal values of the cipher or to obtain different cipher wrong operations. The setup used for the fault injection is depicted in Figure 6, where the capture unit generates the short pulse, selects the round of the encryption process where the pulse is injected, and captures the correct and faulty cipher text at the output. The PicoScope allows to sample the error signal and the cipher busy signal with enough accuracy. It should be noted that such attacks must be performed with precision, since a too short period in the small pulse can be filtered out; conversely, a period that is too big will not produce any error injection. In the tests performed, a clock period of 76.92 ns (13 MHz) was used as the main clock, with a short pulse of 3.33% of the main clock period. Since this setup implements a realistic attacks, the possibilities to control the type of faults are limited. Therefore, we cannot be sure what kind of faults are in fact injected into the cipher, since we only have access to the clock signals, control signals and data output signals.

Note that, numerous tests were performed to ensure that the fault is not a data sampling error. The test validation consist of performing the cipher operations numerous times with the same key and plaintext, capturing the result, and comparing them with each other. In these tests, the small

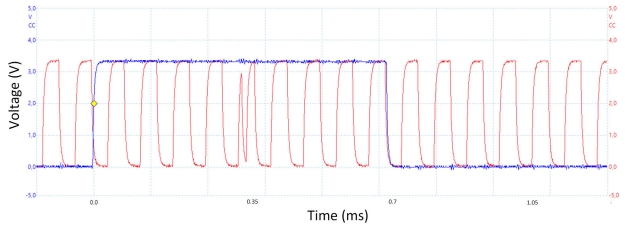


**FIGURE 8.** Experimental capture form PicoScope. Blue, busy cipher signal, red, clock cipher signal with short pulse injection.

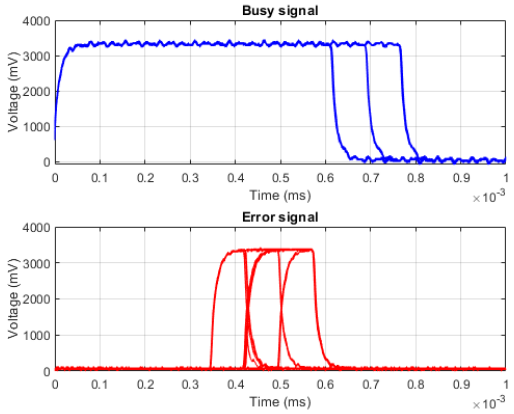


**FIGURE 9.** Experimental capture of the cipher busy signal (top) and scheme error signal (bottom).

**TABLE 2.** Fault coverage by experimental attacks.

| Scheme | Number of Attacks | Number of fault detected | Fault coverage (%) |
|---|---|---|---|
| Unprotected | 10000 | 0 | 0 |
| Hamming | 10000 | 10000 | 100 |
| Hamming_Total | 10000 | 10000 | 100 |
| Par_sig | 10000 | 9850 | 98.50 |
| Par_sig_Total | 10000 | 9999 | 99.99 |

pulse (using larger and smaller pulse weight) is injected into an intermediate round and the output data is sampled at the end of the encryption operation using a sufficiently slow frequency to avoid sampling errors.

In Figure 8 depicts the short pulse injection into the clock signal as well as the cipher busy signal, while Figure 9 depicts the cipher busy signal and the error signal triggering when the fault is detected. As it can be seen in the Figure 9, there are three different falling edges in the busy signal. These three edges are due to the fact that three small pulses have been injected into the main clock signal to cause a fault in the cipher computation. Since it is necessary to inject very small pulses to cause faults in the cipher, it is possible that such a small pulse is filtered out and does not produce any timing violations, and therefore no faults are injected. Therefore, three small pulses are injected consecutively (from 0.3 ms to 0.6 ms) to increase the probability that faults are injected into the cipher.

To evaluate fault detection, each solution was subjected to 10000 fault injection attacks. The fault detection obtained is summarized in Table 2. Unlike the simulated results, only the Hamming and Hamming_Total countermeasures have demonstrated a 100% fault detection. The Par_sig_total countermeasure detected all but one fault out of 10000, resulting in a fault coverage of 99.99%. The Par_sig countermeasure resulted in the lowest fault coverage, since out of 10000 attacks did not detect 150, resulting in an average fault coverage of 98.50%. In the last two cases, it is unclear which type of faults were not detected. Overall, these results suggest that the proposed solutions provide highly effective approaches to faults detention when implemented and in real attack scenarios.

## VI. PERFORMANCE EVALUATION AND COMPARISON WITH OTHER APPROACHES

In order to evaluate the cost of the proposed solutions and compare them with the existing state of the art, the resulting area and performance overhead results have been obtained with respect to the unprotected design on an Xilinx Spartan 6 XC6SLX75 FPGA. The obtained results suggest an area cost of:

1) **Unprotected**: 966 Slices and 1225 LUTs.
2) **Hamming**: 1229 Slices and 1454 LUTs.
3) **Hamming_Total**: 1453 Slices and 1750 LUTs.
4) **Par_sig**: 997 Slices and 1343 LUTs.
5) **Par_sig_Total**: 1425 Slices and 1673 LUTs.

From this it can be concluded that a total resource usage increase of 27%, 50%, 3%, and 47% Slice registers and 18%, 42%, 9% and 36% Look-up-Tables (LUTs), and no additional BRAM usage for the Hamming, Hamming_Total, Par_sig, and Par_sig_Total, respectively.

While the amount of registers increases more than the LUTs, LUTs are the ones imposing the overall device occupation (since more LUTs than registers are required). As such, the LUT usage increase is the one used for the area cost metric, depicted at the bottom of Table 3.

In terms of performance, different post-route runs were performed and the results show that the proposed structures are able to work up to the maximum frequency of the unprotected one. Therefore, it can be concluded that the proposed solutions do not have a performance impact. These results suggest that it is possible to detect all types of faults exploited by DFAs, without impacting the performance of the cryptographic operations.

To better compare the proposed solution with the existing state of the art, Table 3 summarizes the area overhead, frequency degradation, type of faults detected, fault coverage, and type of protection for the AES implementations of these solutions. The frequency degradation is herein used to determine whether the increased security makes the data encryption too slow to operate under similar conditions to the unprotected cipher. The values presented in Table 3 are those provided by the authors of each referenced paper and

**TABLE 3. Comparison with different detection schemes.**

| Scheme | Type of redundancy | Area Overhead | Frequency Degradation | Type of fault | | | | Fault Coverage (%) | Type of protection | Technology |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Odd Bit | Even Bit | Single byte | Multi Byte | | | |
| Unprotected | none | 1 | 1 | ✘ | ✘ | ✘ | ✘ | 0 | NIA | Spartan 6 |
| [28] | Information | 1.08 | 0.70 | ✔ | ✘ | ✘ | ✘ | 75.6 | Partial | Virtex 1000 |
| [37] | Information | 1.44 | NIA | ✔ | ✘ | ✘ | ✘ | 99.12 | Partial | NIA |
| [38] | Information | 1.40 | NIA | ✔ | ✘ | ✘ | ✘ | 97 | Partial | NIA |
| [39] | Information | 1.77 | 0.86 | ✔ | ✔ | ✔ | ✔ | 90 | Partial | Virtex E |
| [40] | Information | 1.25 | 0.88 | ✔ | ✔ | ✔ | ✔ | 98 | Partial | Virtex 5 |
| [43] | Information | 1.73 | 0.64 | ✔ | ✘ | ✘ | ✘ | 88 | Partial | NIA |
| [44] | Information | 1.32 | 0.97 | ✔ | ✔ | ✘ | ✘ | 97 | Partial | Virtex II |
| [26] | Hardware | 1.87 | 1 | ✔ | ✔ | ✔ | ✔ | 100 | Partial | Virtex 5 |
| [27] | Temporal | 1.07 | 0.83 | ✔ | ✘ | ✘ | ✘ | 100 | Partial | Virtex 4 |
| [29] | Combination | 1.38 | 0.78 | ✔ | ✔ | ✔ | ✔ | 100 | Partial | Virtex 6 |
| [45] | Combination | 1.58 | 0.83 | ✔ | ✔ | ✘ | ✘ | 93.75 | Partial | Spartan 3 |
| **Hamming** | **Information** | **1.18** | **1** | ✔ | ✔ | ✔ | ✔ | **100** | **Partial** | **Spartan 6** |
| **Hamming_Total** | **Information** | **1.42** | **1** | ✔ | ✔ | ✔ | ✔ | **100** | **Complete** | **Spartan 6** |
| **Par_sig** | **Information** | **1.09** | **1** | ✔ | ✔ | ✔ | ✔ | **98.50** | **Partial** | **Spartan 6** |
| **Par_sig_Total** | **Information** | **1.36** | **1** | ✔ | ✔ | ✔ | ✔ | **99.99** | **Complete** | **Spartan 6** |

NIA=No Information Available.
$\sim$ = only odd type of faults

are taken directly from them. Note that, the values are for different devices since each author uses their own setups. Given this, the values depict the area overhead and frequency degradation, rather than absolute values. The results obtained for the herein proposed structures have been obtained with no area or frequency optimizations.

This comparison mainly considers information redundancy solutions. For completeness, hardware [26], temporal [27], and combined redundancy solutions [29], [45] are also presented. The type of faults detected by each solution is divided into Odd/Even Bit and Single/Multi Byte. The ability to detect each type of fault is represented by a check or cross mark, respectively. The solutions with four check marks are those that offer the greater fault coverage. The percentages of fault coverage was obtained from the papers in which the solutions are presented and related to the types of faults that can be detected by the solution. Regarding the type of protection, two types can be distinguished; partial and complete. Complete protection is considered when it is capable of detecting faults injected into any process during the encryption operation, considering intermediate operations and KeyAddition. A partial protection is when complete protection is not provided.

Compared with the proposed solutions, the most relevant state of the art solutions are [26]–[29], [37], [39], [40].

The solution proposed in [28] provides one of the detection approaches with the lowest area overhead. However, it reduced the performance to 70% in regard to the unprotected version and is only able to detect odd faulty bits. Moreover, its fault coverage is only 75.6% and does not protect the KeyAddition.

On the other hand, the solution in [39] is able to detect even/odd-bit and single/multi-byte faults and its frequency degradation is 14% but its area overhead reaches 77%, which is very high, its failure coverage does not exceed 90%, and only provides partial protection, not protecting the keyAddition as well as [28].

The solution proposed in [40] is also able to detect even/odd-bit and single/multi-byte faults and its frequency degradation is 12%, but with an area overhead higher than the herein proposed Hamming and Par_sig approaches. It should also be noted that its type of protection is partial (only protects the S-box()).

When considering the solutions with a higher percentage of fault coverage such as [26], [27], [29], [37], it can be seen that all of them only present a partial protection. In the case of [37], the solution protects the S-box and the KeyAddition, but it does not take into account the intermediate states of the state matrix. Moreover, its area cost is higher than all the herein proposed solutions. It does not present frequency penalty data. The solution in [26] has no frequency penalty and can protect against all types of faults. However, its area cost is significantly higher, its protection type is only partial, only covering the state matrix. The solution in [27] provides a fault coverage of 100% and its area overhead is only 7%, however, it cannot protect the KeyAddition, its frequency penalty is of 17%, and is only capable of detecting odd bit type faults. Finally, the solution in [29] is able to detect all the faults considered and has a lower area penalty than the Hamming_Total solution (although Hamming_total offers a complete protection type). However, the frequency penalty of [29] is 22% higher than the solutions herein presented. If we compare it with the Hamming and Par_sig solutions (which offer a partial coverage type), we can see that the cost in area is 20% higher than the presented solutions.

## VII. POWER ANALYSIS EVALUATION

One of the most important aspects to take into account when designing countermeasures is to consider not only the security levels achieved by the countermeasure and the overall performance degradation, but also the effect it has on the security levels against other types of attacks.

Countermeasures against fault attacks typically use hardware redundancy, information redundancy or temporal

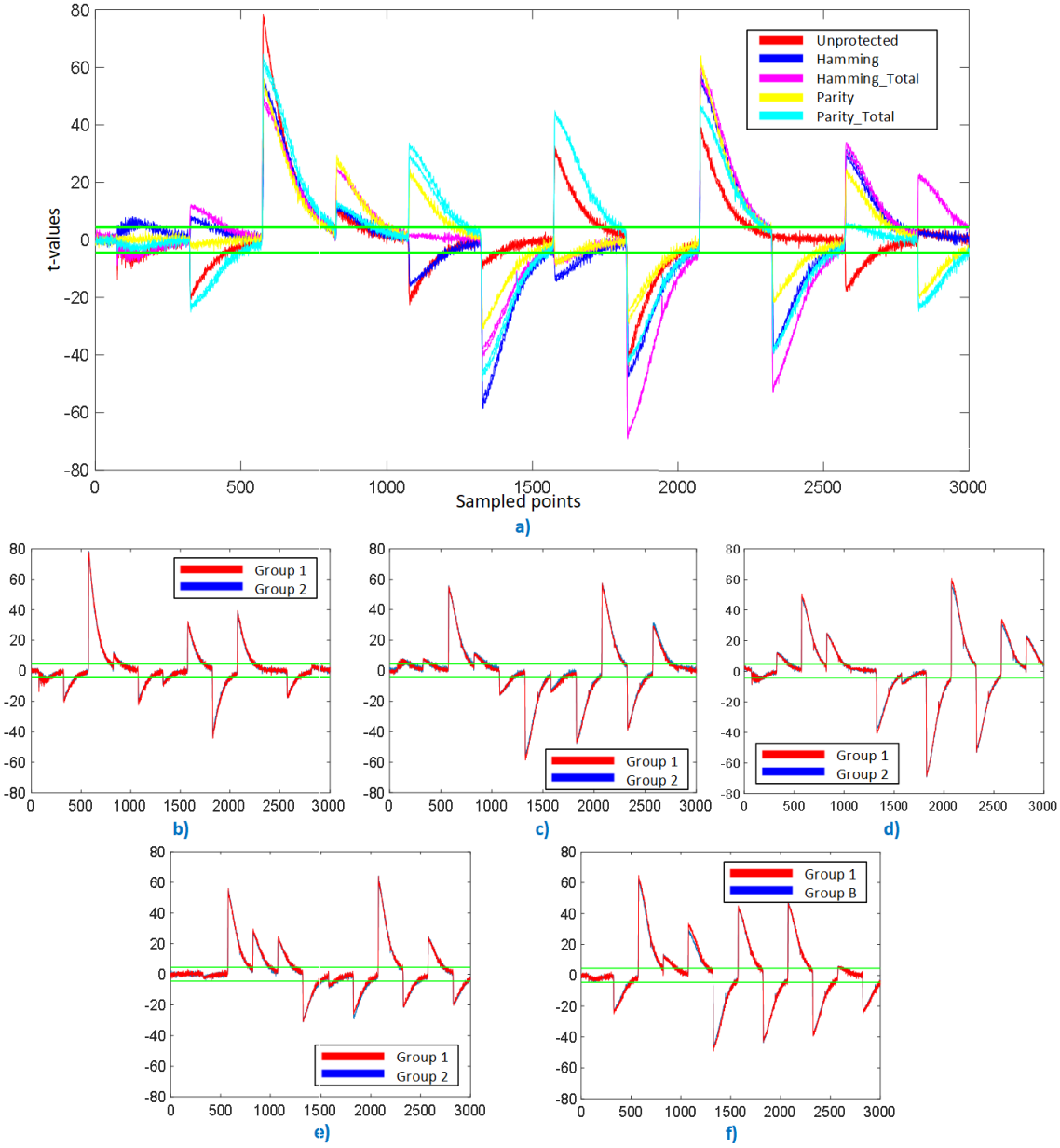


**FIGURE 10.** T-test results: a) all implementations, b) unprotected, c) Hamming, d) Hamming_Total, e) Par_sig and f) Par_sig_Total.

redundancy. This involves adding extra operations with sensitive data within the cryptographic algorithm. These additional operations can result in an increased information leakage (extra power consumption, electromagnetic radiation, execution time among others) that can be exploited by other attacks, known as Side-Channel Attacks [5]–[7]. One of the most usual and effective SCA are those exploiting the key correlated power consumption, in particular DPA attack [6], [8]. The main advantages of DPAs are that the attacker does not need to have a significant information about the implementation of the algorithm. Moreover, since it is a passive non-invasive attack, the normal operation of the device is not altered and the circuit is not physically modified, no evidence of the attack is left behind, so it is not possible to detect whether the cryptographic device has been attacked.

This section focus on assessing if the proposed DFA countermeasures cause any additional leakage of relevant information that may be exploited by SCA, given the extra operations with sensitive data. Towards this, a SCA analysis regarding power consumption has been carried out. To this end, among the several vehicles to evaluate the vulnerability of cryptographic hardware implementations against DPA attacks (such as Mutual Information Analysis (MIA) [52], Measurements To Disclose the Key (MTD) through practical attacks [53],

among others), we have selected the Test Vector Leakage Assessment (TVLA) [31]. TVLA methodology is a reliable, quick and easy test allowing to detect potential side-channel problems with device and in which orders.

To carry out the TVLA analysis, the setup shown in Figure 7 was used. Both the control of the experiment and the data processing was performed using a PC runing Matlab. The power traces were measured with the oscilloscope PicoScope 3205D. The used evaluation board is the SAKURA-G with a Spartan6 Xilinx FPGA where the cryptographic modules are deployed. In the TVLA analysis, the fixed *vs* random test has been carried out, showing in Figure 10 the +/− 4.5 t-value threshold with a green line, which implies a 99.99% confidence to reject the null hypothesis. This means that all the implementations with t-values above 4.5 have a potential leakage that may be exploitable against DPA attacks.

The TVLA results obtained for each implementation are shown in Figure 10, where the sampled points *vs* t-values are shown for: a) all implementations, b) unprotected, c) Hamming, d) Hamming_Total, e) Par_sig and f) Par_sig_Total. From this analysis (see Figure 10-a), it can be seen that all implementations have potential vulnerabilities against power analysis attacks, as all implementations have t-values over the +/−4.5 value. However, the information leakages are of the same order and there is no significant variation in the security levels achieved.

It is important to notice that not all the values above the +/− 4.5 are potentially exploitable in a DPA attack. In the case of the AES algorithm, the attacks are normally focused in the first and the last round [8]. The available known data, used by the power model to estimate the hypothetical power consumption values, is the plaintext, provided in the first round, or the ciphertext, available in the last round. In this sense, a designer can focus specially in these points in the t-test results to see if there are high leakages in this critical rounds. In Figure 10, the points related to the first round are located between points 500-1000 approximately, and the last round between points 2500-3000.

However, it is important to keep in mind that to be sure that the implementation is not vulnerable, it has to have all values below the +/− 4.5 range. This is the case, for example, for combined DPA and DFA attacks [22]–[24], were both side channel leakages and faulty responses are analysed at the same time. Notice that some DFA attacks exploit not only the information of the first and the last round, but also the 7, 8 and $9^{th}$ rounds [11], [16]. The additional operations to increase the security against DFA, are applied in all rounds. For this reason, they can increase the leakage in all encryption rounds. Given this, side-channel leakage needs to be analysed in the whole encryption process. In some points the added logic may even be contributing with added noise, leading for a higher SCA resistance. This is why, as pointed in [25], the SCA resistance must be evaluated in all the particular implementations of cryptographic circuits, as they can affect both positively or negatively to the security. To the best of the authors knowledge, the related DFA state of the art do not evaluate the impact of there solutions to the SCA data leakage, and thus cannot herein properly compared.

In case of detecting an increment in power leakage information, combined DFA and DPA countermeasures can be implemented. In this sense, in the literature a wide range of DPA countermeasures have been proposed that could be compatible with the DFA countermeasures proposed in this paper, as for example the ones focused at gate-level [54].

## VIII. CONCLUSION

In this paper an extensive analysis of the vulnerabilities of AES block ciphers against differential fault analysis and the main blocks to protect it have been presented. The design of four fault injection countermeasures based on signature generation using Hamming and Par_sig schemes has been presented. These schemes have been tested by simulation using ISE and Matlab software, showing that the protection schemes are able to detect exploitable faults by DFAs.

An experimental attack setup has been designed to evaluate the fault coverage of each scheme and it has been shown that the Hamming and Hamming_total schemes offer 100% protection, while the Par_sig and Par_sig_total schemes offer 98.5% and 99.99% respectively.

Regarding the trade-off between resource consumption and frequency degradation, the results show that all the countermeasures have a resource overhead of less than 42%, reaching 9% in the case of the Parity scheme. In addition, it has been experimentally proven that none of the countermeasures has a frequency penalty with respect to the unprotected cipher. Finally, it has been shown that the proposed schemes are capable of detecting all the faults exploitable by the DFA, presenting a balance between area and fault coverage that significantly improves the schemes reported in the literature.

On the other hand, the information leakage exploitable by DPA attacks of each scheme has been evaluated. Results show that none of the countermeasures degrades the security levels against DPA attacks. Furthermore, some of the countermeasures such as Par_sig, slightly improve the security levels against DPA showing smaller t-values.

## ACKNOWLEDGMENT

Authors want to acknowledge to projects SCAROT 1380823-US/JUNTA/FEDER, UE, to Erasmus+, to COST Action CRYPTACUS, to FCT (Fundação para a Ciência e a Tecnologia, Portugal), to the ERDF (European Regional Development Fund, EU) through the projects FCT: UIDB/50021/2020 and LISBOA-01-0145-FEDER-031901 (PTDC/CCICOM/31901/2017, HiPErBio) and to Disruptive Sdn seCuRE communicaTIons for eurOpean defeNse, EDIDP-CSAMN-SDN-202-74-DISCRETION.

## REFERENCES

[1] I. K. Dutta, B. Ghosh, and M. Bayoumi, "Lightweight cryptography for Internet of Insecure Things: A survey," in *Proc. IEEE 9th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Las Vegas, NV, USA, Jan. 2019, pp. 475–481.

[2] N. A. Gunathilake, A. Al-Dubai, and W. J. Buchana, "Recent advances and trends in lightweight cryptography for IoT security," in *Proc. 16th Int. Conf. Netw. Service Manage. (CNSM)*, Izmir, Turkey, Nov. 2020, pp. 1–5.
[3] S. S. Dhanda, B. Singh, and P. Jindal, "Lightweight cryptography: A solution to secure IoT," *Wireless Pers. Commun.*, vol. 112, no. 3, pp. 1947–1980, Jun. 2020.
[4] Z. Kazemi, M. Fazeli, D. Hely, and V. Beroulle, "Hardware security vulnerability assessment to identify the potential risks in a critical embedded application," in *Proc. IEEE 26th Int. Symp. Line Test. Robust Syst. Design (IOLTS)*, Napoli, Italy, Jul. 2020, pp. 1–6.
[5] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, other systems," in *Proc. Annu. Int. Cryptol. Conf. (CRYPTO)*, 1996, pp. 104–113.
[6] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. Annu. Int. Cryptol. Conf. (CRYPTO)*, 1999, pp. 388–397.
[7] Y. Hayashi, N. Homma, T. Mizuki, T. Aoki, H. Sone, L. Sauvage, and J.-L. Danger, "Analysis of electromagnetic information leakage from cryptographic devices with different physical structures," *IEEE Trans. Electromagn. Compat.*, vol. 55, no. 3, pp. 571–580, Jun. 2013.
[8] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. New York, NY, USA: Springer, 2007.
[9] M. Agoyan, J. M. Dutertre, D. Naccache, B. Robisson, and A. Tria, "When clocks fail: On critical paths and clock faults," in *Proc. Int. Conf. Smart Card Res. Adv. Appl. (CARDIS)*, in Lecture Notes in Computer Science, 2010, pp. 182–193.
[10] C. Giraud, "DFA on AES," in *Proc. Int. Conf. Adv. Encryption Standard*, in Lecture Notes in Computer Science, vol. 3373, 2005, pp. 27–41.
[11] P. Dussart, G. Letourneux, and O. Vivolo, "Differential fault analysis on A.E.S," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.*, in Lecture Notes in Computer Science, vol. 2846, 2003, pp. 293–306.
[12] G. Piret and J. J. Quisquater, "A differential fault attack technique against SPN structures, with application to the AES and KHAZAD," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, 2003, pp. 77–88.
[13] A. Moradi, M. T. Manzuri, and M. Salmasizadeh, "A generalized method of differential fault attack against AES cryptosystem," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, 2006, pp. 91–100.
[14] C. N. Cheng and S. M. Yen, "Differential fault analysis on AES key schedule and some countermeasures," in *Information Security and Privacy*, vol. 2727. Berlin, Germany: Springer, 2003, pp. 118–129.
[15] S. S. Ali and D. Mukhopadhyay, "A differential fault analysis on AES key schedule using single fault," in *Proc. Workshop Fault Diagnosis Tolerance Cryptogr.*, Sep. 2011, pp. 35–42.
[16] S. S. Ali, D. Mukhopadhyay, and M. Tunstall, "Differential fault analysis of AES: Towards reaching its limits," *J. Cryptograph. Eng.*, vol. 3, no. 2, pp. 73–97, Jun. 2013.
[17] J. Takahashi, T. Fukunaga, and K. Yamakoshi, "DFA mechanism on the AES key schedule," in *Proc. Workshop Fault Diagnosis Tolerance Cryptogr. (FDTC)*, Sep. 2007, pp. 62–72.
[18] C. H. Kim and J. J. Quisquater, "New differential fault analysis on AES key schedule: Two faults are enough," in *Smart Card Research and Advanced Applications*, in Lecture Notes in Computer Science, vol. 5189, 2008, pp. 48–60.
[19] S. S. Ali and D. Mukhopadhyay, "Differential fault analysis of AES-128 key schedule using a single multi-byte fault," in *Proc. Int. Conf. Smart Card Res. Adv. Appl.*, in Lecture Notes in Computer Science, vol. 7079, 2011, pp. 50–64.
[20] D. Saha, D. Mukhopadhyay, and D. Roychowdhury, "A diagonal fault attack on the advanced encryption standard," *IACR Cryptol. ePrint Arch.*, vol. 581, pp. 1–17, Jan. 2009.
[21] D. Mukhopadhyay, "An improved fault based attack of the advanced encryption standard," in *Proc. Int. Conf. Cryptol. Afr. (AFRICACRYPT)*, 2009, pp. 421–434.
[22] S. Patranabis, J. Breier, D. Mukhopadhyay, and S. Bhasin, "One plus one is more than two: A practical combination of power and fault analysis attacks on PRESENT and PRESENT-like block ciphers," in *Proc. Workshop Fault Diagnosis Tolerance Cryptogr. (FDTC)*, Sep. 2017, pp. 25–32.
[23] A. Chakraborty, B. Mazumdar, and D. Mukhopadhyay, "A combined power and fault analysis attack on protected grain family of stream ciphers," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 12, pp. 1968–1977, Dec. 2017.
[24] S. Patranabis, N. Datta, D. Jap, J. Breier, S. Bhasin, and D. Mukhopadhyay, "SCADFA: Combined SCA+DFA attacks on block ciphers with practical validations," *IEEE Trans. Comput.*, vol. 68, no. 10, pp. 1498–1510, Oct. 2019.
[25] F. Regazzoni, L. Breveglieri, P. Ienne, and I. Koren, "Interaction between fault attack countermeasures and the resistance against power analysis attacks," in *Fault Analysis in Cryptography*. Berlin, Germany: Springer, 2012, pp. 257–272.
[26] M. Joye, P. Manet, and J. B. Rigaud, "Strengthening hardware AES implementations against fault attacks," *IET Inf. Secur.*, vol. 1, no. 3, pp. 106–110, Sep. 2007.
[27] J. Rajendran, H. Borad, S. Mantravadi, and R. Karri, "SLICED: Slide-based concurrent error detection technique for symmetric block ciphers," in *Proc. IEEE Int. Symp. Hardw.-Oriented Secur. Trust (HOST)*, Jun. 2010, pp. 70–75.
[28] K. Wu, R. Karri, G. Kuznetsov, and M. Goessel, "Low cost concurrent error detection for the advanced encryption standard," in *Proc. Int. Conf. Test*, 2004, pp. 1242–1248.
[29] R. Karri, K. Wu, P. Mishra, and Y. Kim, "Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 21, no. 12, pp. 1509–1517, Dec. 2002.
[30] *Advanced Encryption Standard (AES)*, Federal Information Processing Standards publication, NIST, FIPS Standard 197, vol. 441 2001, p. 0311.
[31] J. Cooper, E. Demulder, G. Goodwill, J. Jaffe, and G. Kenworthy, "Test vector leakage assessment (TVLA) methodology in practice," in *Proc. Int. Cryptograph. Module Conf.*, vol. 20, 2014, pp. 1–11.
[32] H. Xiao and L. Wang, "The differential fault analysis on block cipher KLEIN-96," *J. Inf. Secur. Appl.*, vol. 67, Jun. 2022, Art. no. 103205.
[33] D.-P. Le, S. L. Yeo, and K. Khoo, "Algebraic differential fault analysis on Simon block cipher," *IEEE Trans. Comput.*, vol. 68, no. 11, pp. 1561–1572, Nov. 2019.
[34] B. Karmakar and D. Saha, "DESIV: Differential fault analysis of SIV-Rijndael256 with a single fault," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Dec. 2020, pp. 241–251.
[35] H. Luo, Y. Wu, and W. Chen, "Differential fault attack on TWINE block cipher with nibble," in *Proc. IEEE 20th Int. Conf. Commun. Technol. (ICCT)*, Oct. 2020, pp. 1151–1155.
[36] D. Boneh, R. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn. (EUROCRYPT)*, 1997, pp. 37–51.
[37] L. Breveglieri, I. Koren, and P. Maistri, "Incorporating error detection and online reconfiguration into a regular architecture for the advanced encryption standard," in *Proc. 20th IEEE Int. Symp. Defect Fault Tolerance VLSI Syst. (DFT)*, 2005, pp. 72–80.
[38] M. Kermani and A. Reyhani-Masoleh, "Parity-based fault detection architecture of S-box for advanced encryption standard," in *Proc. 21st IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, Oct. 2006, pp. 572–580.
[39] M. Karpovsky, K. J. Kulikowski, and A. Taubin, "Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard," in *Proc. Int. Conf. Dependable Syst. Netw. (DSN)*, 2004, pp. 93–101.
[40] H. Mestiri, N. Benhadjyoussef, M. Machhout, and R. Tourki, "High performance and reliable fault detection scheme for the advanced encryption standard," *Int. Rev. Comput. Softw.*, vol. 8, no. 3, pp. 730–746, 2013.
[41] R. W. Hamming, "Error detecting and error correcting codes," *Bell Syst. Tech. J.*, vol. 29, no. 2, pp. 147–160, Apr. 1950.
[42] C. Moratelli, F. Ghellar, E. Cota, and M. Lubaszewski, "A fault-tolerant, DFA-resistant AES core," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2008, pp. 244–247.
[43] C.-H. Yen and B.-F. Wu, "Simple error detection methods for hardware implementation of advanced encryption standard," *IEEE Trans. Comput.*, vol. 55, no. 6, pp. 720–731, Jun. 2006.
[44] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A lightweight high-performance fault detection scheme for the advanced encryption standard using composite fields," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 1, pp. 85–91, Jan. 2011.
[45] J. Chu and M. Benaissa, "Error detecting AES using polynomial residue number systems," *Microprocessors Microsyst.*, vol. 37, no. 2, pp. 228–234, Mar. 2013.
[46] B. Khodadad-Mostashiry, "Parity prediction in combinational circuits," in *Proc. Annu. Int. Symp. Fault-Tolerant Comput. (FTCS)*, Madison, WI, USA, vol. 9, Jun. 1979, pp. 185–188.
[47] H. Mestiri, F. Kahri, B. Bouallegue, and M. Machhout, "A hardware FPGA implementation of fault attack countermeasure," in *Proc. 15th Int. Conf. Sci. Techn. Autom. Control Comput. Eng. (STA)*, Dec. 2014, pp. 178–183.
[48] R. Karri, G. Kuznetsov, and M. Goessel, "Parity-based concurrent error detection of substitution-permutation network block ciphers," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, 2003, pp. 113–124.

[49] M. K. Stojcev and M. D. Krstic, "Parity error detection in embedded computer system," in *Proc. 5th Int. Conf. Telecommun. Modern Satell., Cable Broadcast. Service (TELSIKS)*, 2001, pp. 445–450.
[50] R. Chaves, G. Kuzmanov, S. Vassiliadis, and L. Sousa, "Reconfigurable memory based AES co-processor," in *Proc. 20th IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, Apr. 2006, pp. 1–8.
[51] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Proc. IEEE*, vol. 94, no. 2, pp. 370–382, Feb. 2006.
[52] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, "Mutual information analysis," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, 2008, pp. 426–442.
[53] K. Tiri, D. Hwang, A. Hodjat, B. C. Lai, S. Yang, P. Schaumont, and I. Verbauwhede, "Prototype IC with WDDL and differential routing—DPA resistance assessment," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, 2005, pp. 354–365.
[54] E. Tena-Sánchez, F. E. Potestad-Ordóñez, C. J. Jiménez-Fernández, A. J. Acosta, and R. Chaves, "Gate-level hardware countermeasure comparison against power analysis attacks," *Appl. Sci.*, vol. 12, no. 5, p. 2390, Feb. 2022.

**F. E. POTESTAD-ORDÓÑEZ** received the B.Sc. degree in electronic engineering, the M.Sc. degree (Hons.) in electronic products design, and the Ph.D. degree from the University of Seville, Spain, in 2013, 2015, and 2019, respectively. Since 2015, he has been with the Instituto de Microelectrónica de Sevilla (IMSE-CNMCSIC)/University of Seville. His current research interests include fault injection attacks of cryptographic circuits and countermeasure study and design for secure systems.

**E. TENA-SÁNCHEZ** received the B.Sc. degree in telecommunications engineering from the University of Cantabria, Spain, in 2010, and the B.Sc. degree (Hons.) in electronics engineering, the M.Sc. degree in microelectronics, and the Ph.D. degree from the University of Seville, Spain, in 2012, 2013, and 2019, respectively. Since 2011, she has been with the Instituto de Microelectrónica de Sevilla (IMSE-CNM-CSIC)/University of Seville. Her current research interest includes CMOS digital design of secure cryptographic circuits.

**A. J. ACOSTA-JIMÉNEZ** received the B.Sc. (five-year) and Ph.D. degrees in physics from the University of Seville, Seville, Spain, in 1989 and 1995, respectively. Since 1990, he has been with IMSE. Since 2002, he has also been an Assistant Professor with the University of Seville. He is currently a Full Professor at the University of Seville and a Senior Researcher at the Instituto de Microelectrónica de Sevilla (IMSE-CNM-CSIC), Seville. He has coauthored more than 100 international scientific publications. He has led a number of different national and European research and development projects. His current research interests include low-power and low-noise CMOS digital and mixed-signal high-performance VLSI Design, timing in VLSI digital systems, and cryptographic circuits. He has served as a member of editorial boards for international journals and on program committees for several prestigious conferences. He was the General Chair of the 2002 PATMOS International Workshop.

**C. J. JIMÉNEZ-FERNÁNDEZ** graduated in physics from the University of Seville, Spain, in 1989. He received the Ph.D. degree in physics from University of Seville, in 2000. Since 1990, he has been with IMSE. Since 2002, he has also been an Assistant Professor with the University of Seville. He has participated in seven projects financed by the EC and 12 projects financed in national calls. He has coauthored more than 40 contributions to workshops, conferences, and journals. He has supervised five Ph.D. thesis. His current research interests include experimental vulnerability analysis of cryptocircuits against fault attack and design of secure cryptocircuits.

**RICARDO CHAVES** (Senior Member, IEEE) received the Ph.D. degree in computer engineering from the Technical University of Delft, The Netherlands, and the University of Lisbon, Portugal, in 2007. He is currently an Associate Professor at the Department of Computer Science, University of Lisbon/IST, and a Researcher at INESC-ID. He has published more than 90 papers in international journals and conferences in the research areas. His research interests include cryptography systems, reconfigurable hardware architectures, and embedded systems.