Hardware Architecture Implemented on FPGA for Protecting Cryptographic Keys against Side-Channel Attacks

Rubén Lumbiarres-López, Mariano López-García, and Enrique Cantó-Navarro

Abstract—This paper presents a new hardware architecture designed for protecting the key of cryptographic algorithms against attacks by side-channel analysis (SCA). Unlike previous approaches already published, the fortress of the proposed architecture is based on revealing a false key. Such a false key is obtained when the leakage information, related to either the power consumption or the electromagnetic radiation (EM) emitted by the hardware device, is analysed by means of a classical statistical method. In fact, the trace of power consumption (or the EM) does not reveal any significant sign of protection in its behaviour or shape. Experimental results were obtained by using a Virtex 5 FPGA, on which a 128-bit version of the standard AES encryption algorithm was implemented. The architecture could easily be extrapolated to an ASIC device based on standard cell libraries. The system is capable of concealing the real key when various attacks are performed on the AES algorithm, using three statistical methods which are based on correlation, the Welch's t-test and the difference of means.

Index Terms—Security, side-channel attacks, power analysis attacks, software-hardware countermeasures

1 Introduction

THE addition of countermeasures for protecting the key in cryptographic algorithms has become an emerging field of research, since in the late 1990s several authors revealed the inherent weakness associated with physical devices used in their implementation [1]. When a cryptographic algorithm is implemented in a hardware device, it could be shown as both its power consumption and its electromagnetic radiation (EM) are heavily dependent on the data that are being processed. Since data rely on the cryptographic key, this dependence can be exploited to find out such a key by using a statistical method of analysis. Further, as the leakage information that is exploited is external to the hardware device, these methods are usually known as Side-Channel Analysis (SCA) attacks.

The most widely used statistical method is based on the calculation of the correlation between the captured power trace (or the EM) and a theoretical model of power consumption for a specific key. In order to obtain such a model, it is necessary to know both the data that are being processed and the behaviour of the basic CMOS cells that form the circuit. This model is usually approximated by the Hamming distance (HD) or the Hamming weight (HW) related to the binary value of the particular point to be attacked in the circuit [2]. This approximation is based on the assumption that the actual consumption is proportional to HW or HD. The former represents the number of ones included in a byte $v(t_k)$ at instant t_k, whereas the latter is based on the HW of the result obtained when operating with an exclusive-OR the value of byte v at instants t_{k-1} and t_k (i.e., $v(t_{k-1})$ and $v(t_k)$). Nevertheless, the knowledge of data is more complicated, since such data depend not only on the plain text to be encrypted but also on the value of the cryptographic key. Generally, it is accepted that the attacker knows the plain text (or the encrypted text) and he/she makes N hypotheses for the N possible keys. For the sake of feasibility, most publications focus the attack on a specific byte ($N = 256$), and usually it is considered impractical to use values of 32 or more bits due to the high number of hypotheses that should be performed. The correct key is determined by the highest correlation found among all guessed hypotheses. This correlation is calculated by capturing a set of current traces, whose number depends on factors such as the signal to noise ratio or the accuracy of the power consumption model [3].

Most researchers focus the design of countermeasures to avoid SCA attacks on breaking the existing correlation between the data processed by the hardware device and the cryptographic key. The success of such approaches is reflected in the value of the correlation factor, which should be identical and close to zero for all guessed keys. A simple way to achieve this objective is in designing systems in which the power consumption is constant for every clock cycle [4]. Such systems are usually designed in hardware at gate or cell level, and they require approximately double the area, compared with their non-protected counterpart versions. The design is performed on a dual-rail network based on two complementary wires, whose load capacitances must be perfectly balanced to guarantee the success of the countermeasure. Such a condition is difficult to achieve in practice, even when certain constrains are included as part of the placement and routing steps [5]. Another important group of approaches aim at eliminating the correlation by concealing all values v processed into the hardware device with a random mask m. Usually, the operation employed to conceal such values is an exclusive-OR, so that the masked value v_m could be represented by $v_m = v \oplus m$. Under certain conditions v_m is independent with respect to v, and therefore, the cryptographic key cannot be revealed by means of statistical methods. These approaches have been implemented at both cell and algorithm levels. The latter has the disadvantage that the execution time is about doubled when compared with a non-protected system. The former, related to hardware implementations, has proven to be vulnerable due to the early propagation effect [6], [7]. Nevertheless, as such weaknesses were known many of these proposals have been improved by including subsystems or modifications that minimise or eliminate such effects [8], [9], [10].

Another interesting approach implemented on FPGAs was presented by Kamoun et al. [11]. Their proposal is based on deteriorating the side-channel signal by adding a noise power generator. The main feature of such an approach, when compared with other implementations based on similar strategies, is that the noise is correlated with both the data manipulated by the system and a specific interfering key. However, this countermeasure is only effective when the attack is performed on the function block linked with the correlated noise power. Furthermore, the revealed key is not always the same and it depends on the number of traces captured and used to perform the attack.

Almost all previous passive approaches based their fortress on hiding the cryptographic key, in such a way that the failure of the SCA attack is produced because all possible keys are equally likely. Generally, this objective is difficult to achieve, since any small difference, signal delay or defect in the implementation is enough to generate a tiny correlation between the data and the cryptographic key. Such minimal correlation can be successfully exploited to obtain the key by simply processing a higher number of current traces.

The countermeasure proposed in this paper is completely different and is based on protecting the system by revealing a false key. This key is randomly chosen by the designer and can be changed at any time. In fact, the overall encryption process is always performed using such a false key, without introducing any constraint in the implementation or any additional mechanism of protection, so that an attack by SCA will never lead to finding out

- *R. Lumbiarres-López and M. López-García are with the Universitat Politècnica de Catalunya, Vilanova i la Geltrú 08800, Spain. E-mail: {ruben.lumbiarres, mariano.lopez}@upc.edu.*
- *E. Cantó-Navarro is with the Universitat Rovira i Virgili, Tarragona 43007, Spain. E-mail: enrique.canto@urv.cat.*

Manuscript received 16 Dec. 2015; revised 25 July 2016; accepted 12 Sept. 2016. Date of publication 0 . 0000; date of current version 0 . 0000.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TDSC.2016.2610966

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2016.2610966, IEEE Transactions on Dependable and Secure Computing

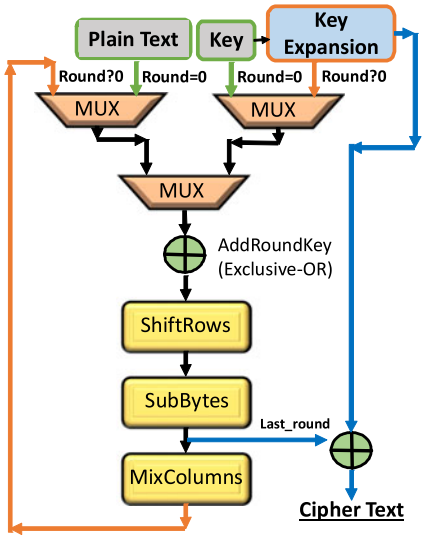

Fig. 1. Internal structure of the AES-128 algorithm.

the true key. This paper presents the design and implementation of this countermeasure on a Virtex 5 FPGA, although as no restrictions are included in the design, the proposal can be easily implemented on an ASIC by using standard cells.

The paper is organised as follows. Section 2 describes the fundamentals of the faking countermeasure. Section 3 shows the internal structure of the proposed hardware implementation. Section 4 shows the experimental results and finally Section 5 presents the conclusions.

2 Faking Countermeasure

Nowadays, the Advanced Encryption Standard (AES) is the most popular algorithm used by researchers when proposing countermeasures against attacks by SCA. The basis of this standard is well documented in [12], [13]. Briefly, it consists of four functions or steps, *AddRoundKey*, *SubBytes*, *ShiftRows* and *MixColumns*, which are subsequently applied on several rounds over a 4x4 matrix of 16 bytes. Such a matrix, termed the state, is initially obtained by combining the plain text and the encryption key K_{REAL} with an exclusive-OR gate. In the second and subsequent rounds, the state is operated with a new key obtained after processing the original key with an algorithm known as *key expansion*. Fig. 1 shows the internal structure of the AES algorithm including the aforementioned four steps.

For a specific instant of time t_k, an attack by SCA could be successfully performed on the input or the output of any of the four functions represented in Fig. 1. For that purpose the following information should be available: a theoretical model that represents the power consumption of the device to be attacked; a set of T captured current traces; and finally a known plain text which could be conveniently chosen according to the target function. The aim is to calculate the correlation factor between the theoretical model of power consumption and the actual consumed power (other methods based on a different statistical measure can be used). Note that, this calculation can only be performed if K_{REAL} is known, since its value, jointly with the plain text, is necessary to assess the model of power consumption. Usually, the attacker makes N hypotheses for K_{REAL} and calculates the correlation factor for each one. The true key could perfectly be identified, since such a key produces the highest correlation factor between all hypotheses. As aforementioned, in practice the attack is only feasible if the number of hypotheses made is in accordance with the processing capability of existing computers and the time needed for capturing the current traces. Thus, due to practical issues, the attack is usually focused on a specific byte of the state (256 possible encryption keys).

The faking countermeasure is based on the simple idea of processing the plain text with a false key K_{FALSE}. However, following this strategy the cipher text would be incorrectly encrypted with a key that is different to K_{REAL}. Thus, in some stage of the algorithm an additional processing should be introduced in order to recover the original text encrypted with the correct key. Let the relation between K_{FALSE} and K_{REAL} be the following:

$$K_{FALSE} = K_{REAL} \oplus K_{MASK}, \quad (1)$$

where K_{MASK} is a value that consists of 16 bytes, and $\oplus$ represents the exclusive-OR operator. Note that (1) is satisfied for each of the 16 bytes that form K_{MASK}, (i.e., $K_{FALSE}(i,j) = K_{REAL}(i,j) \oplus K_{MASK}(i,j)$, $i = 0..3$ and $j = 0..3$). The function *ShiftRows* is a permutation of the elements of each row of the state, so that it does not have any influence on the process of recovering the original text. In contrast, the non-linear function *SubBytes* has a significant effect. Such a function is applied over all bytes $a_F(i,j)$ ($i = 0..3$, $j = 0..3$) of the state, which are generated by combining the plain text $T(i,j)$ and $K_{FALSE}(i,j)$ (i.e., $a_F(i,j) = T(i,j) \oplus K_{FALSE}(i,j)$. Let $a_R(i,j)$ ($i = 0..3$, $j = 0..3$) be the value of the same byte if it was encrypted with K_{REAL} (i,j) (*i.e.*, $a_R(i,j) = T(i,j) \oplus K_{REAL}(i,j)$). Then, the output of *SubBytes*, which is represented by $SBox(a_F)$, could be related with $a_F(i,j)$ and $a_R(i,j)$ as follows:

$$SBox\,(a_F(i,j)) = SBox(a_R(i,j)) \oplus M(a_F(i,j)). \quad (2)$$

Taking into account (1) and (2), the value of function $M(a_F(i,j))$ can be expressed as:

$$M\,(a_F(i,j)) = SBox(a_X(i,j)) \oplus SBox(a_X(i,j) \oplus K_{MASK}(i,j)), \quad (3)$$

where $a_X(i,j)$ could be either $a_F(i,j)$ or $a_R(i,j)$. Note that, as byte $a_X(i,j)$ can only take 256 different values, $M(a_F(i.j))$ can be stored in a small memory and it can be pre-computed before executing the AES algorithm. Moreover, (3) is very useful, since if $M(a_F(i,j))$ is known then it is possible to recover, at the output of *SubBytes*, the state encrypted with K_{REAL} by using as input data the actual output of *SubBytes* (see (2)).

The state encrypted with K_{FALSE} is finally processed by the function *MixColumns*, which is based on linear operations over elements of different rows of the state. Let $b_F(i,j)$ and $b_R(i,j)$ ($i = 0..3$ and $j = 0..3$) be the output byte (i,j) of this function when the plain text is encrypted with K_{FALSE} and K_{REAL}, respectively. Thus, such output can be expressed as:

$$MixColumns(b_F(i,j)) = MixColumns(b_R(i,j)) \oplus N(i,j) \quad (4)$$

being

$$N\,(i,j) = MixColumns(M(a_F(i,j))) \quad (5)$$

and

$$MixColumns\,(b_R(i,j)) = MixColumns(Sbox(a_R(i,j))). \quad (6)$$

Therefore, at the end of any round processed with K_{FALSE}, the original text encrypted with K_{REAL} can be obtained by simply operating with an exclusive-OR the output of *MixColumns* (described in (4)) and $N(i,j)$ (described in (5)). This operation is known as remasking. Subsequently, the correct state is used as input of the following round, repeating the process until the algorithm is finished.

3 Implementation of the Faking Countermeasure

Fig. 2 represents the internal structure of the two proposals presented in this paper for implementing the faking countermeasure. In the first proposal (Fig. 2a), the system was segmented into two stages including their corresponding registers, so that each round

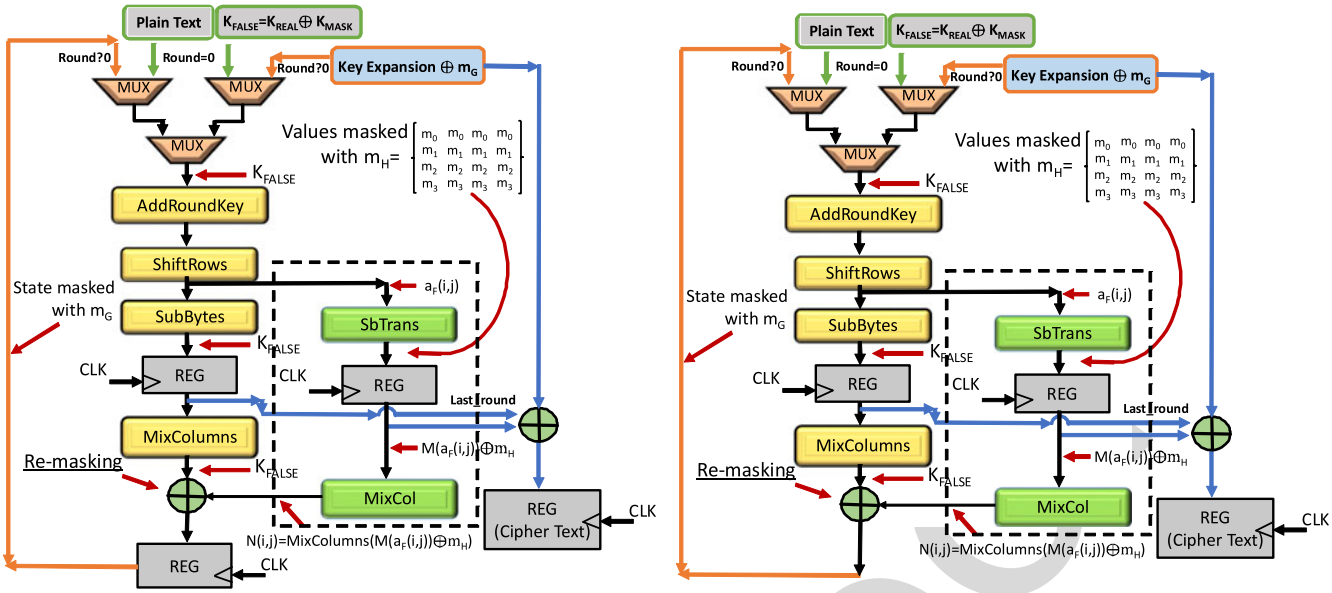

Fig. 2. Hardware implementation of the faking countermeasure: a) Two clocks per round, b) One clock per round.

can be solved in two clock cycles (T_{CLK}). In the first cycle, functions *AddRoundKey*, *ShiftRows* and *SubBytes* are evaluated, while in the second cycle the function *MixColumns* and the remasking are processed. In the second proposal (Fig. 2b), the register at the output of *MixColumns* is eliminated, so that each round is processed in only one cycle. As will be seen in the next section, when comparing both implementations the experimental results will be quite different, providing a trade-off between speed, number of traces required to undertake a successful attack and correlation value. Registers are the usual target chosen by attackers. They have the advantage that their output only switches once per clock cycle. In contrast, due to the delay of signals, logic gates may switch many times per clock cycle producing glitches that are difficult to predict. Such glitches have a remarkable influence on the consumed power, so that it is hard to generate a faithful model that matches the real power consumed by the device.

The implementation of function *AddRounKey* is very simple, since it corresponds to an exclusive-OR operation. Likewise, *ShiftRows* does not require any logical resource, since it can be implemented by connecting properly the output of *AddRoundKey* with *SubBytes*. Finally, *SubBytes* and *Mixcolumns* are implemented following an identical hardware architecture that is used to build *SbTrans* and *MixCol*, respectively.

The block *SbTrans* is in charge of implementing (3) for obtaining $M(a_F(i,j))$. As shown in Fig. 2, a random mask m_H is used for concealing the output of this function. This mask is necessary for several reasons:

- Note that the attacker is able to know K_{FALSE}, since this is the aim of the faking countermeasure. Then, based on (3), if a first-order SCA attack is performed on the output of *SbTrans*, the value of K_{MASK} can be easily revealed, and therefore applying (1) the actual value of K_{REAL} could also be determined. It is noteworthy that this situation of risk can be avoided by including a masking scheme that protects the output of *SbTrans*, so that (3) is modified as follows [14], [15]:

$$M(a_F(i,j)) = SBox(a_F(i,j)) \oplus SBox(a_F(i,j) \oplus K_{MASK}(i,j)) \oplus m_H. \quad (7)$$

- In addition, if no masking was used, by combining the values of registers located at the output of *SubBytes* and *SbTrans*, a second-order attack would be possible [15], since both values can be processed by an exclusive-OR leading to a new result which depends only on $a_R(i,j)$:

$$M(a_F(i,j)) \oplus Sbox(a_F(i,j)) = Sbox(a_R(i,j)). \quad (8)$$

- Besides, the mask m_H must be included to protect the state once the remasking is performed. Thus, at the output of the register located after *Mixcolumns* (Fig. 2a) the state will be masked with a new mask m_G:

$$m_G(i,j) = MixColumns(m_H(i,j)). \quad (9)$$

Additionally, note that before *AddRoundKey* the state is always protected since it is encrypted with the false key.

A masking scheme is effective if mask m_H changes its value randomly and independently on the data that is being processed. Thus, a True Number Random Generator (TNRG) is included as part of the design to create such a mask. The internal structure of this block is based on the design proposed in [19], which basically uses Configurable Logic Blocks (CLB) available in all FPGAs. However, the update of m_H is the main challenge of such design, since this change should be performed without affecting the execution time or the encrypted text. In order to facilitate this process, the pre-computed values of (7) are stored in a set of 16 memories denoted as $M_k\,(k = 0..15)$. The input of each memory M_k corresponds to one of the 16 bytes that form the state. Although only 256 bytes per memory would be necessary, (7) is implemented twice in two consecutive areas of memory, referred to as $M_{k,a}$ and $M_{k,b}$, (then, a total of 512 bytes are used) masked with two different masks m_H and included as part of M_k. Thus, when $M_{k,a}$ is being used to implement (7), the second block $M_{k,b}$ is being updated with a new mask m_H, without affecting the normal operation of the countermeasure. Afterwards, the second block $M_{k,b}$ is used for encrypting a new plain text, whereas the first block $M_{k,a}$ is updated in a similar way to how $M_{k,b}$ was previously updated. Details about the implementation are given in the next section.

The value of $N(i,j)$, described in (5), is calculated by means of the block termed as *MixCol* in Fig. 2. In fact, its implementation aims at reproducing the function *Mixcolumns* defined by the AES encryption algorithm. Its internal structure, only for byte 0, is represented in Fig. 3. The remaining set of bytes are calculated with an identical implementation. *Mixcolumns* is mainly based on additions

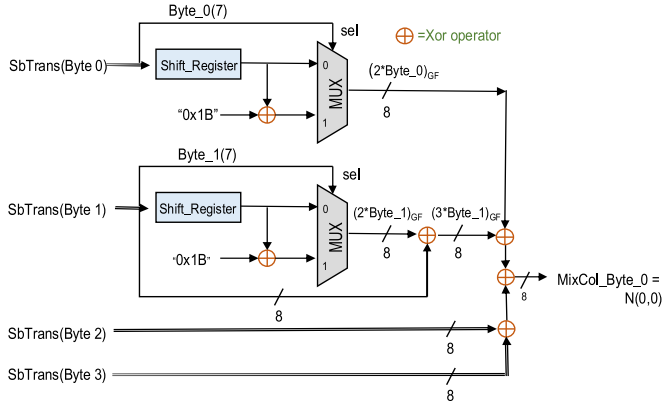

Fig. 3. Partial hardware implementation of block MixCol (calculation of byte 0). This block is repeated 16 times.

and multiplications for constants (only values 2 or 3) performed over the bytes of a column of the state. The easiest way of implementing a multiplication by 2 is using a shift-register, while a multiplication by 3 can be performed by means of a shift-register and an addition. Moreover, the sum of two bytes can be carried out by simply using an exclusive-OR operator. In this way, as demonstrated in Fig. 3, each byte at the output of *MixCol* can be implemented by including simple blocks such as shift-registers, multiplexors and exclusive-OR operators.

4 Experimental Results

Experiments were conducted to test the correctness of the proposed faking countermeasure. The whole system, following the hardware architecture shown in Figs. 2 and 3, was implemented on a Virtex-5 FPGA clocked at 24 MHz. Power traces were measured using a Tektronix CT-1 current probe with a bandwidth range 25 kHz – 1 GHz. The current probe was connected to an Agilent DSO1024A oscilloscope, which captures and stores current traces using a sample rate of 2 GS/s.

4.1 Area and Maximum Clock Frequency

The logic resources needed for implementing the overall system, and the maximum clock frequency fixed by the critical path, are represented in Table 1. These results were obtained using the ISE design Suite 13.1 and the synthesis tool XST provided by Xilinx. Only the area was the parameter chosen to be optimized a no additional constraints were included in the implementation process. The table shows the results obtained for an unprotected version of the AES 128-bit encryption algorithm (using one or two registers) and for the two proposals presented in Fig. 2 including the faking countermeasure (one or two clock cycles per round). Note that, the number of slices is increased by about 30 percent when such countermeasure is included, but it only represents a small part of the total amount of slices available in the FPGA. Besides, there is not a noteworthy difference between the logical resources needed by the two implementations based on one or two registers.

On the other hand, as the internal architecture presented in Fig. 2a is based on a couple of registers, each round could be solved in 2 clock cycles (the last round is also solved in 2 clock cycles, due to the output register in which the cipher text is placed). Thus, a complete encryption process is performed in $20{\cdot}T_{CLK}$. Moreover, both the area and the maximum clock frequency of the simplest implementation based on only one register (Fig. 2b) is almost identical to the first version. However, each round could be solved in $1{\cdot}T_{CLK}$, so that a plain text could be encrypted in $11{\cdot}T_{CLK}$, which represents an important advantage in terms of resolution time.

It is noteworthy that the implementation of the countermeasure requires the use of 16 blocks of BRAM, that are used for implementing the 16 $M_k\,(k = 0..15)$ memories previously described. The size of a BRAM memory block is 18 kb (16 kb are for data and 2 kb are for parity). In our particular case, each block of BRAM was configured as a memory capable of storing 2 k bytes of data. The upper area of such a memory is used for implementing $M_{k,a}$, whereas the lower area is employed for implementing $M_{k,b}$. As BRAM memory blocks are dual-port, they can be configured to read and write simultaneously at different addresses. This property allows managing each block of BRAM as two independent memories, which facilitates the updating of mask m_H following the procedure described in Section 3.

4.2 Results for Different Attacks

In order to perform the analysis, traces of current are compressed in such a way that all samples captured during a clock period are substituted by their average value. Results are given for both proposals presented in Fig. 2. Thus, Fig. 4 shows an attack performed on the register located at the output of function *SubBytes* when the faking countermeasure is activated. As can be seen, in both cases the system is completely protected by revealing the false key K_{FALSE} rather than the K_{REAL}. Specifically, in the first implementation the correlation obtained for the first byte is about 0.14, while in the second implementation the value is 0.05. Results for the rest of sub-bytes are represented in Table 2. Such a table also shows the ratio between the maximum values obtained for the correlation regarding K_{FALSE} and K_{REAL}. The best case is produced for the sub-byte 7, for which the correlation obtained for the true key is more than 6 times smaller than the correlation obtained for K_{FALSE}. The worst case is given in sub-byte 15, in which a ratio of 1.17 is obtained.

On the other hand, although the system based on two registers takes almost twice the time spent by the second proposal, the value of the correlation related to the false key is in most cases higher.

Fig. 5 shows the evolution of the correlation over an increasing number of current traces related to different plain texts. Note that, the minimum number of traces needed to differentiate K_{FALSE} from the rest of the possible keys is 2,000 and 5,000 traces for the proposals based on two and one registers, respectively. Again, the best result is obtained for the first proposal (i.e., two registers).

Fig. 6 shows an attack based on the difference-of-means method proposed by Kocher [1]. Unlike the original attack, which was performed on a single bit, this attack is targeted on a complete byte following a similar strategy that introduces some modifications

- The process is applied on each bit *j* included in the byte to be analysed.

TABLE 1
Area and Maximum Clock Frequency F_{max}

AES-128 encryption algorithm	LUTs (Lookup table)	FF (Flip-Flops)	Slices	BRAM	F_{max} (MHz)
Unprotected system excluding the faking (one register)	2,664 (9.2%)	2,610 (9.2%)	1,193 (15.6%)	–	167
Unprotected system excluding the faking (two registers)	2,609 (9.1%)	2,740 (9.5%)	1,096 (15.2%)	–	192
System protected by faking (one register, Fig. 2.*a)*	3,963 (13.8%)	2,750 (9.5%)	1,535 (21.3%)	16 (33%)	167
System protected by faking (two registers, Fig. 2.*b)*	3,806 (13.2%)	2,880 (10%)	1,332 (18.5%)	16 (33%)	192

Percentage (%) against the total number of resources in the FPGA.

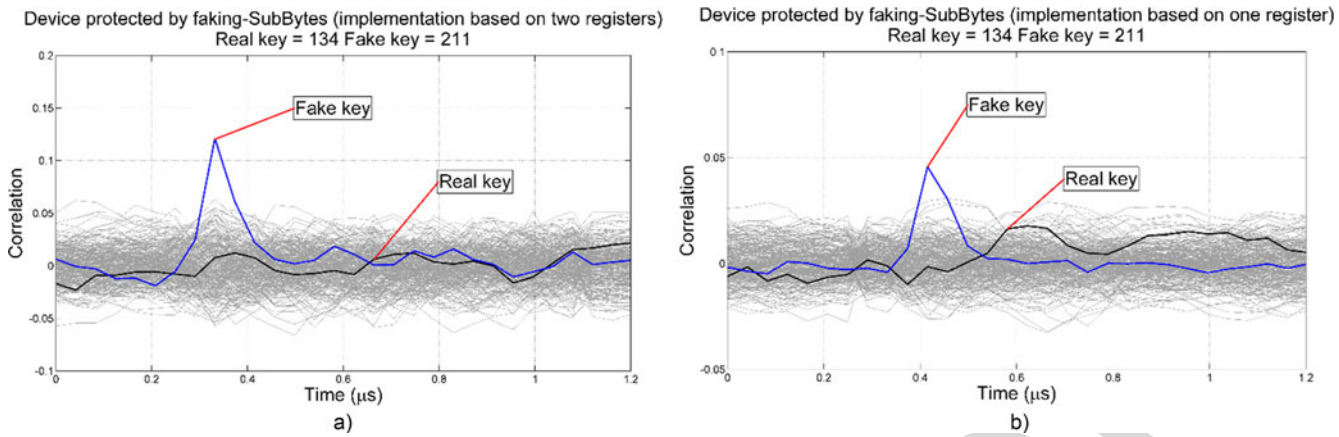

Fig. 4. Experimental attack on SubBytes based on the correlation method and including the faking countermeasure: a) Implementation based on two registers (as represented in Fig. 2a), b) Implementations based on one register (as represented in Fig. 2b). The K_{FALSE} is plotted in blue color and the K_{REAL} is plotted in bold.

TABLE 2
Maximum Correlation for K_{FALSE}, K_{REAL} and Maximum Value for the Result of the Difference-of-Means Method Using K_{FALSE}, K_{REAL}

Byte (2 register vs. 1 register)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Maximum correlation using the false key K_{FALSE_MAX}	0,14 vs. 0,05	0,08 vs. 0,03	0,10 vs. 0,05	0,13 vs. 0,04	0,12 vs. 0,07	0,10 vs. 0,03	0,14 vs. 0,07	0,13 vs. 0,05	0,11 vs. 0,08	0,09 vs. 0,04	0,09 vs. 0,05	0,11 vs. 0,05	0,11 vs. 0,07	0,09 vs. 0,06	0,07 vs. 0,07	0,12 vs. 0,05
Maximum correlation using the real key K_{REAL_MAX}	0,03 vs. 0,02	0,06 vs. 0,02	0,05 vs. 0,01	0,04 vs. 0,02	0,04 vs. 0,02	0,06 vs. 0,02	0,02 vs. 0,02	0,05 vs. 0,02	0,03 vs. 0,01	0,03 vs. 0,02	0,05 vs. 0,01	0,04 vs. 0,01	0,05 vs. 0,02	0,03 vs. 0,01	0,06 vs. 0,02	0,04 vs. 0,02
Ratio between max. correlations K_{FALSE_MAX} / K_{REAL_MAX}	5,51 vs. 2,61	1,36 vs. 1,3	1,99 vs. 3,69	2,96 vs. 2,05	2,92 vs. 3,41	1,71 vs. 1,62	6,41 vs. 4,35	2,37 vs. 2,88	3,54 vs. 5,73	2,85 vs. 2,70	2,01 vs. 4,78	2,97 vs. 4,97	2,23 vs. 3,48	2,80 vs. 5,71	1,17 vs. 3,48	3,22 vs. 3,48
Maximum value for difference-of-means D_{FALSE_MAX} *for* K_{FALSE}	0,43 vs. 0,29	0,41 vs. 0,23	0,37 vs. 0,31	0,39 vs. 0,30	0,47 vs. 0,41	0,41 vs. 0,26	0,41 vs. 0,42	0,39 vs. 0,35	0,34 vs. 0,49	0,27 vs. 0,28	0,32 vs. 0,29	0,34 vs. 0,32	0,47 vs. 0,45	0,32 vs. 0,39	0,33 vs. 0,44	0,46 vs. 0,34
Maximum value for difference-of-means D_{REAL_MAX} *for* K_{REAL}	0,07 vs. 0,15	0,26 vs. 0,14	0,20 vs. 0,08	0,19 vs. 0,17	0,20 vs. 0,15	0,13 vs. 0,14	0,09 vs. 0,10	0,14 vs. 0,12	0,11 vs. 0,10	0,13 vs. 0,07	0,16 vs. 0,02	0,19 vs. 0,04	0,21 vs. 0,09	0,11 vs. 0,07	0,25 vs. 0,15	0,11 vs. 0,07
Ratio between max. values of D_{FALSE_MAX} / D_{REAL_MAX}	6,05 vs. 1,95	1,55 vs. 1,69	1,87 vs. 3,94	2,08 vs. 1,80	2,32 vs. 2,85	3,15 vs. 1,90	4,37 vs. 4,32	2,86 vs. 2,95	2,98 vs. 4,79	2,06 vs. 4,11	1,98 vs. 13,86	1,77 vs. 8,46	2,22 vs. 5,24	2,96 vs. 5,31	1,34 vs. 2,97	4,22 vs. 4,49

Ratios for maximum values are also included.

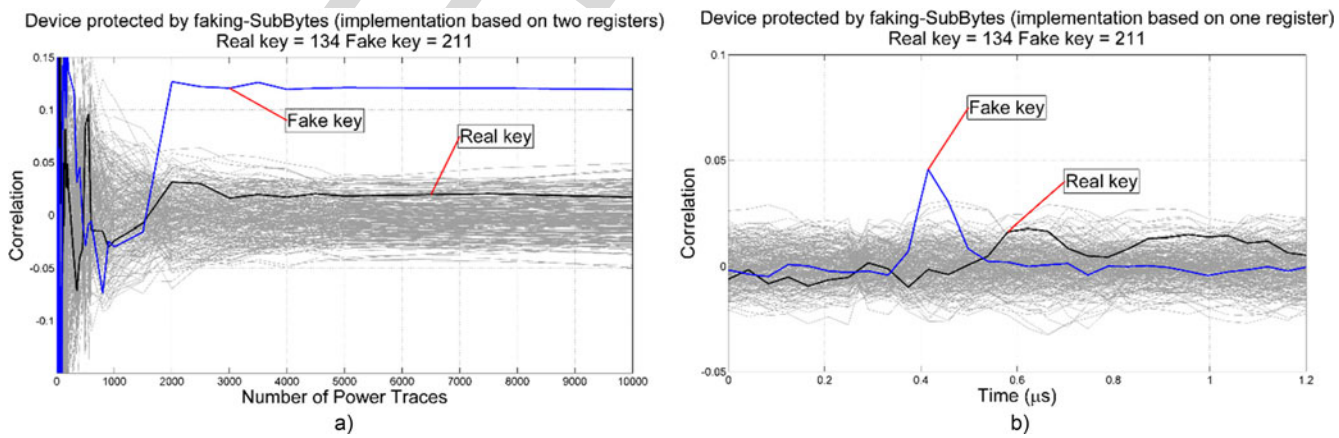

Fig. 5. Experimental attack on SubBytes based on the correlation method and including the faking countermeasure: a) Implementation based on two registers (as represented in Fig. 2a), b) Implementations based on one register (as represented in Fig. 2b). The K_{FALSE} is plotted in blue color and the K_{REAL} is plotted in bold.

- For the specific bit *j*, in which the attack is initially focused, the N current traces are separated into two groups, depending on the value that such a bit takes on the power consumption model for a particular plain text and a specific key K_n $(n = 0..255)$.
- For each key K_n, the average of each group is calculated and the difference between each average is assigned to the element $d(j, n)$ $(j = 0..7,\ n = 0..255)$ of a matrix D.
- The process is repeated for all bits and keys until matrix D is completed.

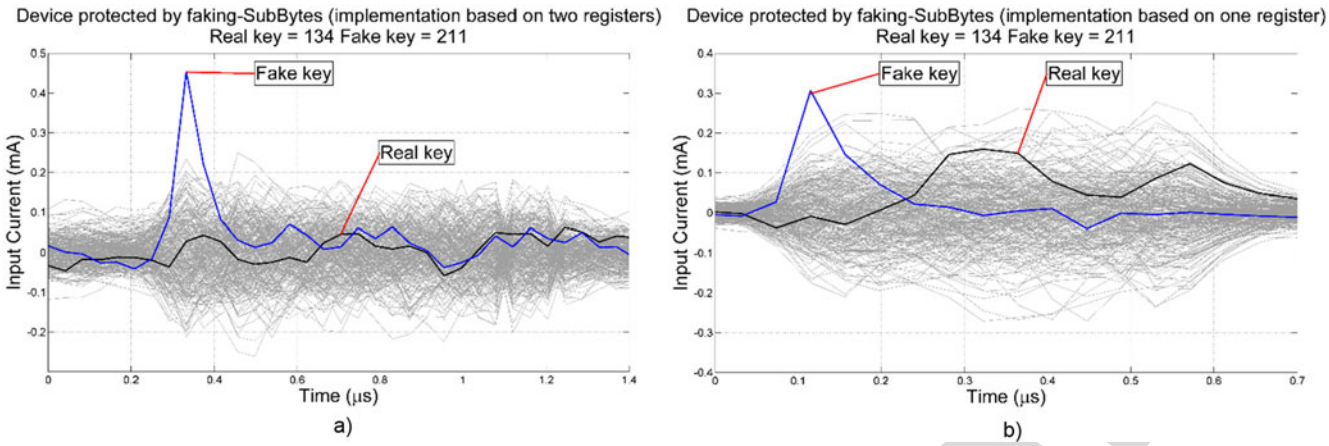

Fig. 6. Experimental attack on SubBytes based on the difference-of-means method and using the faking countermeasure. a) Implementation based on two registers (as represented in Fig. 2a) taking 3,000 traces, b) Implementations based on one register (as represented in Fig. 2b) taking 15,000 traces. The K_{FALSE} is plotted in blue color and the K_{REAL} is plotted in bold.

- For each column n of matrix D, its average value $D_n\,(n = 0..255)$ is calculated. The maximum value of D_n indicates the correct key K_n.

Fig. 6 represents the result of such an attack performed on the time interval in which the *SubBytes* operation is executed. Note that, the proposed countermeasure successfully conceals the real key. On the other hand, comparing the results of the two implementations it can be concluded again that the version based on two registers produces a higher difference of means, which corroborates the result presented in Fig. 4. Details about the numeric results for all sub-bytes are given in Table 2. Additionally, on such a version the number of current traces needed to obtain a successful result is 3,000, whereas when using only one register such a number is increased to 15,000 current traces.

Fig. 7 justifies the need of including a mask m_H to protect several vulnerable parts of the system. In this case, the figure shows an attack performed on the register located at the output of function *Mixcolumns*, but excluding the use of a mask m_H as part of the process carried out on function *SbTrans*. The attacker uses a particular plain text in which the 15 more significant bytes are identical. Only byte 0 is changing its value during each encrypting process. It is noteworthy that by using this approach, in the first round the value of the correlation at the output of *Mixcolumns* will be only affected by the first byte provided by the output of *SubBytes*. Such particularity makes an attack performed at function *Mixcolumns* feasible, since its value can be easily predicted. In practice, such an attack could be carried out by extending, by several additional clock cycles, the calculation of the correlation at the output of *SubBytes*. This conclusion is shown in Fig. 7. Note that, as no mask m_H is used, the system is only protected until the instant of time in which the remasking between the output of *Mixcol* and *Mixcolumns* is produced. The K_{FAKE} is revealed at time instant 325 ns, when the output of *SubBytes* is evaluated. However, in the following clock cycle, when the remasking function is calculated, the system is vulnerable and reveals the true key K_{REAL} (trace plotted in bold).

Finally, the protection offered by the faking countermeasure has also been evaluated following the Test Vector Leakage Assessment (TVLA) methodology proposed in [20]. Additional details about this methodology based on the Welch's t-test can be found in [21], [22]. Basically, such a test evaluates if whether two sets of data are significantly different from each other. The calculation of the t-test statistic is based on the mean, the variance and the number of samples that form each set of samples. For the sake of simplicity, if such t-statistic is higher than a threshold, usually $|t|>4.5$, then it is accepted that the device fails.

The two sets of data correspond to the overall set of captured traces that were obtained by a known encrypting plain text. If the categorization of the two sets is carried out without any knowledge of the encryption key, then the test is so-called non-specific t-test (or fixed versus random data datasets). In this case, a fixed text T_{fixed} is selected and the device is fed by T_{fixed} or by a random text T_{random} following a non-deterministic pattern. The categorization is performed taking into account if traces were obtained using T_{fixed} or T_{random}. Fig. 8 shows the results for the t-statistic in the first round when the operations *SubBytes* and *MixColumns* are being

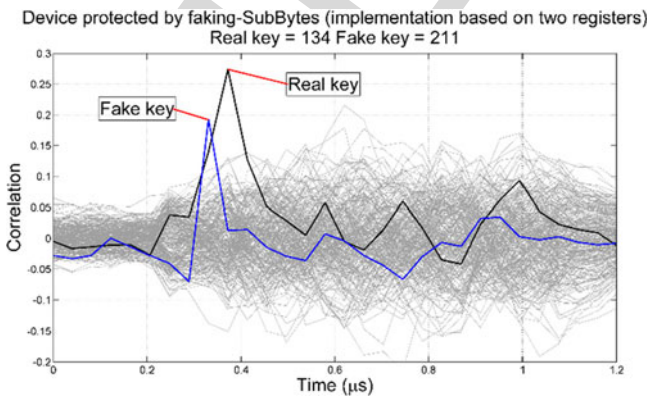

Fig. 7. Experimental attack on SubBytes using the correlation method. System protected by the faking countermeasure but without using the mask m_H. The K_{FALSE} is plotted in blue color and the K_{REAL} is plotted in bold.

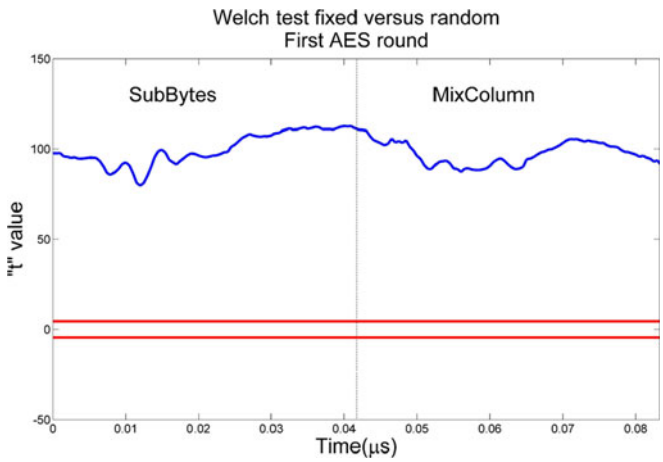

Fig. 8. Experimental t-test results for the fixed-versus-random test based on 15,000 AES operations. The t-statistic overcomes the failing thresholds -4.5 and +4.5 (represented in red lines).

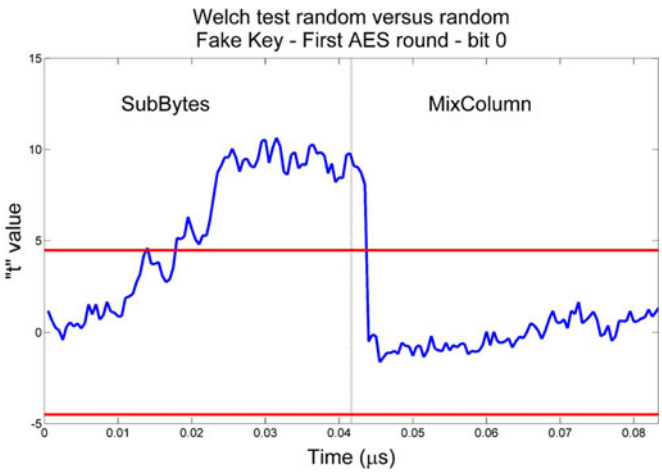

Fig. 9. Experimental t-test results for the random-versus-random test based on 15,000 AES operations using K_{FAKE}. The t-statistic overcomes the failing thresholds -4.5 and +4.5 (represented in red lines).

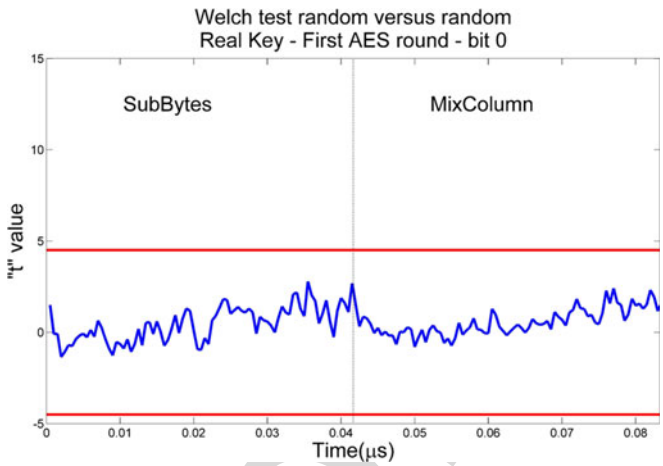

Fig. 10. Experimental t-test results for the random-versus-random test based on 15,000 AES operations using K_{REAL}. The t-statistic is lower than the failing thresholds -4.5 and +4.5 (represented in red lines).

evaluated. As is was expected, such a value is always higher than $|t|>4.5$ (fail test), since the system is designed to reveal the false key.

On the other hand, if the categorization of the two sets is performed by means of an intermediate value (for instance the value of a bit at an instant in time) then the test is so-called specific t-test (or random versus random data datasets). Note that in this case the encryption key should be known and then the Welch's test could be focused on K_{FALSE} or K_{REAL}. Fig. 9 shows the result of such a test over K_{FALSE}. As can be seen, during the operation *SubBytes* (the categorization is based on the output of this operation) the t-statistic is higher than 4.5, revealing a leakage of information. However, if the same test is performed or K_{REAL} the result is quite different. As Fig. 10 shows, is such a case the value of the t-statistic is always lower than $|t|<4.5$ (pass test), which shows as the K_{REAL} is effectively concealed by the faking countermeasure.

4.3 Comparison with Other Proposals Implemented at Cell Level

The implementation of countermeasures against SCA attacks, have usually been performed using structures based on DRP (Dual-Rail Precharge) logic styles. Such structures are based on either full-custom designs, such as the proposals performed in [4], [7], or designs based on standard cell libraries [5], [8]. Our proposal does not include any restriction so that it falls into the second group. Hence, as was seen in the experimental results, it can be implemented in FPGAs or in a different technology. On the other hand, the implementation of the faking countermeasure leads to an increase in the number of logical resources by about 30 percent (measured in terms of slices), when compared with the non-protected version (see Table 1). Additionally, the maximum clock frequency is identical, and it is not affected by the inclusion of the proposed countermeasure.

The performance of the proposals made in previous publications, in terms of logical resources and frequency, vary depending on the hardware structure in which the countermeasure is based. For instance, as it is shown in [16], an optimized design of a Wave Dynamic Differential Logic (WDDL) resistant style on an FPGA requires 1.95 times the slices of the single-ended design, and in a non-optimised version this number could be increased by up to 4 times. Other countermeasures such as BCDL [17], MDPL [18] or iMDPL [8] also increases the area needed by their implementation by a factor that is always higher than 2. On the other hand, DRP logic styles follow (independently, if a random mask is included as part of the basic cell) a sequence based on two states: precharge and evaluation. During the precharge phase the outputs are set to either 1 or 0, while in the evaluation phase only one of the outputs changes its value. Thus, for a constant clock frequency, the time needed by a DRP logic style for encrypting a plain text is twice that when compared with a structure based on simple Single Rail (SR) networks (assuming that all registers flip-flops are synchronised by positive or negative edge) and the same ratio is obtained if such a comparison is made against the faking countermeasure.

4.4 Comparison with an Implementation at Algorithm Level

The faking countermeasure can also be implemented at algorithm level, but this leads to lower performance than the approach carried out at cell level. Table 3 shows the results obtained when an AES 128-bit encryption algorithm is executed on MicroBlaze, the 32-bit microprocessor soft-core provided by Xilinx. In order to facilitate comparison with other microprocessors of similar features, the results of execution time are given in clock cycles (T_{CLK}). Additionally, two implementations have been performed. In the first (second column of Table 3) the countermeasure was included as part of the algorithm, whereas in the second one (third column of Table 3) the countermeasure was disabled. As can be seen, when the countermeasure is activated, the number of clock cycles is almost twice that of the non-protected version. Such a difference is

TABLE 3
Execution Time for One Round of AES 128-Bit Algorithm

Function	Execution time (including faking)	Execution time (without including faking)
AddRoundKey	52.76 μs (1266·T_{CLK})	39.16 μs (940·T_{CLK})
ShiftRows	4.04 μs (97·T_{CLK})	4.04 μs (97·T_{CLK})
SubBytes	26.58 μs (638·T_{CLK})	26.58 μs (638·T_{CLK})
MixColumns	54.03 μs (1297·T_{CLK})	43.17 μs (1036·T_{CLK})
SboxTrans	4.46 μs (107·T_{CLK})	–
MixCol	54.03 μs (1297·T_{CLK})	–
Remasking	6.04 μs (145·T_{CLK})	–
Execution time for one round	**201.94 μs** (4847·T_{CLK})	**112.95 μs** (2711·T_{CLK})

Results Given for a 32-Bit Microprocessor (MicroBlaze) at 24 MHz, Including and Excluding the Faking Countermeasure.

mainly due to the *MixCol* function, which requires about 25 percent of the total execution time. The results obtained in [2] for an AES 128-bit masked implementation at algorithm level are quite similar. The difference between the masked an unmasked implementations, measured in clock cycles, is also double. Thus, although the faking countermeasure can very well be included in a microprocessor, the best performance is obtained for the hardware implementation at cell level.

5 Conclusion

This paper presented a novel countermeasure against SCA attacks implemented in hardware. Unlike previous approaches aimed at concealing the statistical dependence between data and power consumption, the fortress of the countermeasure is based on revealing a false key. In order to verify the correctness of our proposal, two different implementations were performed on a Virtex 5 FPGA. Several attacks were carried out on function *SubBytes* of the AES 128-bit encryption algorithm. In all cases, the experimental results corroborated the efficiency of our proposal, demonstrating that the system is completely protected. Particularly, results show that the first implementation based on two registers provides the highest correlation factor for the false key using a lower number of captured traces. However, the second implementation is able to encrypt a plain text in half the time using about the same amount of logical resources. When compared with countermeasures based on dual-rail networks, the area needed for implementing our proposal is significantly lower. Additionally, as no restrictions are included in the hardware design, the system could also be implemented in an ASIC device using standard cell libraries.

Acknowledgments

This work was supported by the Ministerio de Economía y Competitividad in the framework of the Programa Nacional de Proyectos de Investigación Fundamental, project TEC2012-38329-C02-02.

References

[1] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. 19th Ann. Int. Cryptology Conf. Adv. Cryptology*, Aug. 15-19, 1999, pp. 388–397.
[2] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks—Revealing the Secrets of Smart Cards*. New York, NY, USA: Springer, 2007.
[3] S. Mangard, "Hardware countermeasures against DPA—A statistical analysis of their effectiveness," in *Topics in Cryptology*, T. Okamoto Ed. Berling, Germany: Springer, Feb. 23-27, 2004, pp. 222–235.
[4] S. K. Tiri, M. Akmal, and I. Verbauwhede, "A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards," in *Proc. 28th Eur. Solid-State Circuits Conf.*, 2002, pp. 403–406.
[5] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation," in *Proc. Des. Autom. Test Eur. Conf. Expo.*, Feb. 16-20, 2004, pp. 246–251.
[6] S. Mangard and K. Schramm, "Pinpointing the side-channel leakage of masked AES hardware implementation," in *Proc. Cryptographic Hardware Embedded Syst.*, 2006, pp. 76–90.
[7] D. Suzuki, M. Saeki, and T. Ichikawa, "Random switching logic: A countermeasure against DPA based on transition probability," *Cryptology ePrint Archive*, Report 2004/346, 2004. [Online]. Available: http: //eprint.iacr.org/.
[8] T. Popp, M. Kirschbaum, T. Zefferer, and S. Mangard, "Evaluation of the masked logic style MDPL on a prototype chip," in *Proc. Cryptographic Hardware Embedded Syst.*, Sep. 10-13, 2007, pp. 81–94.
[9] R. P. McEvoy, C. C. Murphy, W. P. Marnane, and M. Tunstall, "Isolated WDDL: A hiding countermeasure for differential power analysis on FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, no. 1, pp. 1–23, 2009.
[10] W. He, E. de la Torre, and T. Riesgo, "A precharge—Absorbed DPL logic for reducing early propagation effects on FPGA implementations," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, 2011, pp. 217–222.
[11] N. Kamoun, L. Boussuet, and A. Ghazel, "Correlated power noise generator as low cost DPA countermeasures to secure hardware AES chiper," in *Proc. 3rd. Int. Conf. Signals Circuits Syst.*, 2009, pp. 1–6.
[12] J. Daemen and V. Rijmen, *The Design of Rijndael: AES—The Advanced Encryption Standard*. Berlin, Germany: Springer, 2002.
[13] J. Daemen and V. Rijmen, "AES proposal: Rijndael," [Online]. Available: http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf, 1999.
[14] S. Mangard, E. Oswald, and F.-X. Standaert, "One for all—All for one: Unifying standard differential power analysis attacks," *IET Inf. Security*, vol. 5, no. 2, 2011.
[15] G. Piret and F. X. Standaert, "Security analysis of higher-order Boolean masking schemes for block ciphers (with conditions of perfect masking)," *IET Inf. Secur.*, vol. 2, no. 1, 2008.
[16] K. Tiri and I. Verbauwhede, "Synthesis of secure FPGA implementations," in *Proc. Int. Workshop Logic Synthesis*, 2004, pp. 224–231
[17] M. Nassar, S. Bhasin, J.-L. Danger, G. Due, S. Guilley, "BCDL: A high performance balanced DPL with global precharge and without early-evaluation," in *Proc. Des. Autom. Test Eur. IEEE Comput. Soc.*, Mar. 2010, pp. 849–854.
[18] T. Popp and S. Mangard, "Masked dual-rail pre-charge logic: DPA-resistance without routing constraints," in *Proc. 7th Int. Workshop Cryptographic Hardware Embedded Syst.*, 2005, pp. 172–186.
[19] P. Kohlbrenner and K. Gaj, "An embedded true random generator for FPGAs," in *Proc. ACM/SIGDA 12th Int. Symp. Field Programmable Gate Arrays*, 2004, pp. 71–78.
[20] J. Cooper, E. Demulder, G. Goodwill, J. Jaffe, G. Kenworthy, and P. Rohatgi, "Test vector leakage assessment (TVLA) methodology in practice," in *Proc. Int. Cryptographic Module Conf.*, 2013. [Online]. Available: http://icmc-2013.org/wp/wp-conent/uploads/2013/09/goodwillkenworthtestvector.pdf
[21] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side channel resistance validation," in *Proc. NIST Non-Invasive Attack Testing Workshop*, 2011. [Online]. Available: http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf
[22] T. Schneider and A. Moradi, "Leakage assessment methodology—A clear roadmap for side-channel evaluations," in *Proc. Cryptographic Hardware Embedded Syst.*, 2015, pp. 495–513.

Queries to the Author

Q1. Please provide complete bibliography details for references [7], [21], and [23].
Q2. Please provide year for reference [13].
Q3. Please provide page-range for references [20] and [22].