A Reliable Architecture for Parallel Implementations of the Advanced Encryption Standard

Giorgio Di Natale, Marion Doulcier, Marie-Lise Flottes, Bruno Rouzeyre

A Reliable Architecture for Parallel Implementations of the Advanced Encryption Standard

G. Di Natale, M. Doulcier, M. L. Flottes, B. Rouzeyre

Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier
Université Montpellier II / CNRS UMR 5506
161 rue Ada, 34392 Montpellier Cedex 5, France
Tel: +33 467 41 85 01
Fax: +33 467 41 85 00
Email: dinatale@lirmm.fr

Abstract

This paper presents an on-line self-test architecture for hardware implementation of the Advanced Encryption Standard (AES). The solution exploits the inherent spatial replications of a parallel architecture for implementing functional redundancy at low cost. We show that the solution is very effective for on-line fault detection while keeping the area overhead very low. Moreover, the architectural modification for on-line test does not weaken the device with respect to side-channel attacks based on power analysis.

1. Introduction

Standard cryptographic functions such as the Advanced Encryption Standard (AES) are today implemented in a wide range of devices targeting various application domains with security requirements. In addition to the inherent property of these devices, allowing storage and transmission of sensitive information across insecure networks, many applications require high reliability for guarantying a proper digital security. Consequently, as other parts of the system, crypto-cores must be carefully designed in order to provide reliable

processing of sensible data. Design for on-line testability of such cores prevents structural failures to cause loss of service and compromise the security.

Fault detection and tolerance schemes for various implementations of cryptographic algorithms have been recently considered. Mainly, two approaches have been developed: based on information redundancy (e.g. the use of codes, [1] [2] [3]) or functional redundancy ([3] [4] [5]).

All the techniques based on codes add some bits to the original data word in order to check its validity. The main issue in these approaches is the prediction of the value of the code on an output, given the input value and the executed operation. For instance, the prediction of a parity bit is almost straightforward for the ShiftRows, MixColumns and AddRoundKey operations performed in the AES [7] because these transformations are either linear or they just perform some bit permutations (see Section 2 for a detailed description of the AES). Conversely, the prediction of the parity bit is not trivial for the SubBytes operation performed by the so-called S-Boxes. As a consequence, the parity prediction requires larger circuitry. Solutions based on parity codes ([1] [2]) lead to an overhead of about 20% and high single error detection. However they are not effective in case of multiple faults or single faults that lead to an even number of errors. Other solutions based on the use of more complex codes such as CRC [1] or systematic nonlinear robust codes [3] lead to higher fault coverage but at the expense of a significant area overhead (> 60%).

Alternatively, the techniques presented in [3], [4], and [5] are based on functional redundancy. They can be used whenever encryption and decryption modules are implemented on the same circuit. Each encoding phase is followed by a decoding and compare phase in order to check if the resulting decoded text matches with the initial plaintext. A similar procedure is employed when the circuit is used for decoding a cipher-text.

Conversely to most of the previously proposed approaches that focus on the S-Boxes only (dominant component, counting up to 75% of the circuit area), we propose a low cost self-test architecture for detecting single and multiple faults in most of the AES hardware. The form of testing is accomplished using duplication and comparison. The main idea is to implement the datapath in such a way that several identical blocks can be defined. With an additional block, online pair wise

comparisons of blocks are implemented to check the functionality of the AES hardware. Efficiency and low area overhead are achieved by exploiting the spatial duplication inherent to the parallel implementation of the algorithm.

Moreover, since any structural modification on the hardware implementation may jeopardize the digital security, the proposed architecture is also checked with respect to one of the most common attack based on power analysis [6].

The paper is organized as follows. Section 2 introduces the basic concepts and the characteristics of the Advanced Encryption Standard algorithm. Section 3 presents the proposed on-line self-test approach, while section 4 discusses the results in terms of area overhead and fault detection capability. Section 5 introduces the problem of side channel attacks based on power analysis, and presents experimental results showing the resistance of the proposed architecture to such an attack. Eventually, Section 6 concludes the paper.

2. Advanced Encryption Standard

AES [7] is a block cipher adopted as an encryption standard by the U.S. government. AES began immediately to replace the Data Encryption Standard (DES, used since 1976) for the reason that it outperforms in long-term security thanks to, among other things, larger key sizes (128, 192, or 256 key bits). For sake of simplicity, we focus on 128-bit key in the sequel of the paper

Another major advantage of AES is its efficient implementation on various platforms. It is suitable for small 8-bit microprocessor platforms, common 32-bit processors, and dedicated hardware implementations that can reach throughput rates in the gigabit range. Several hardware implementations are presented in [8].

The AES algorithm's internal operations are performed on a two dimensional array of bytes called State. The State consists of 4 rows of 4 bytes. Each byte is denoted by $S_{i,j}$ ($0 \leq i < 4$, $0 \leq j < 4$) . The four bytes in each column of the State array form a 32-bit word, with the row number as the index for the four bytes in each word. The initial plain text is a 128-bit block that can be expressed as 16 bytes: in_0, in_1, in_2… in_{15}. Encryption and decryption processes are performed on the State, at the end of which the final value is mapped to the output bytes array out_0, out_1, out_2, … out_{15}.

The AES is an iterative process composed of 10 rounds. The plain text to cipher is first copied to the State array. After the initial secret key addition (roundkey(0)), the first 9 rounds are identical, with small difference in the 10th round. As illustrated in Figure 1, each of the first 9 rounds consists of 4 transformations: SubBytes, ShiftRows, MixColumns and AddRoundKey. The final round excludes the MixColumns transformation. The encryption scheme in Figure 1 can be inverted to get a straightforward structure for decryption.

SubBytes Transformation

The SubBytes transformation is a non-linear byte substitution that operates independently on each byte of the State using a substitution table (S-Box). This S-Box is constructed by composing two transformations:

1. Take the multiplicative inverse in the finite field $GF(2^8)$; the element $(00000000)_2$ is mapped to itself;

2. Apply the following affine transformation (over $GF(2)$):

$$b_i^{'} = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i \tag{1}$$

for $0 \leq i < 8$, where b_i is the ith bit of the byte, and c_i is the ith bit of a byte c whose value is fixed and is equal to {01100011}.

This transformation can be pre-calculated for each possible input value since it works on a single byte (only 256 values). S-Boxes can be implemented either as a ROM or as combinational logic.
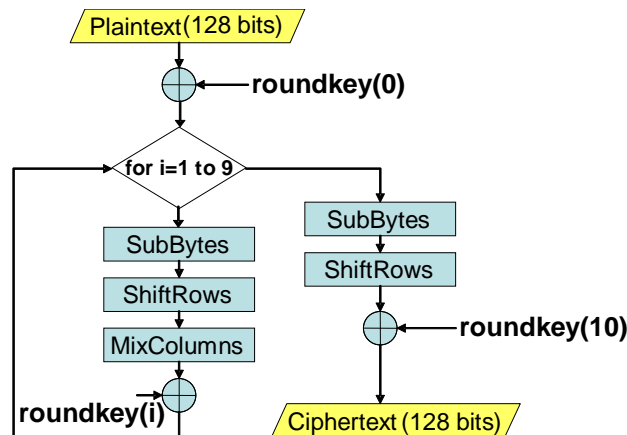

Figure 1: AES Algorithm (encryption)

ShiftRows Transformation

In this transformation, the bytes in the first row of the State do not change. The second, third, and fourth rows shift cyclically to the left one byte, two bytes, and three bytes, respectively.

MixColumns Transformation

The MixColumns transformation is performed on the State array column-by-column. Each column is considered as a four-term polynomial over $GF(2^8)$ and multiplied by a(x) modulo $x^4 + 1$, where:

$$a(x) = (00000011)_2 \, x^3 + (00000001)_2 \, x^2 + (00000001)_2 \, x + (00000010)_2 \quad (2)$$

AddRoundKey Transformation

In AddRoundKey transformation, a roundkey is added to the State array by bitwise XOR operation. Each roundkey consists of 16 bytes generated from the Key Expansion operation described below.

Key Expansion

The key expansion routine, as part of the overall AES algorithm, takes the input secret key of 128 bits and outputs an expanded key of 11*128 bits composed of the input secret key and 10 roundkeys, one for each round. Details of the algorithm for determining the value of each roundkey are given in [7].

3. Functional redundancy for on-line fault detection

The technique we propose in this paper is designed for all the AES cores (encryption and decryption) that use 16 S-Box repetitions. We do not consider low-area implementations, where there is only one S-Box at the cost of several clock cycles for completing one encryption/decryption round.
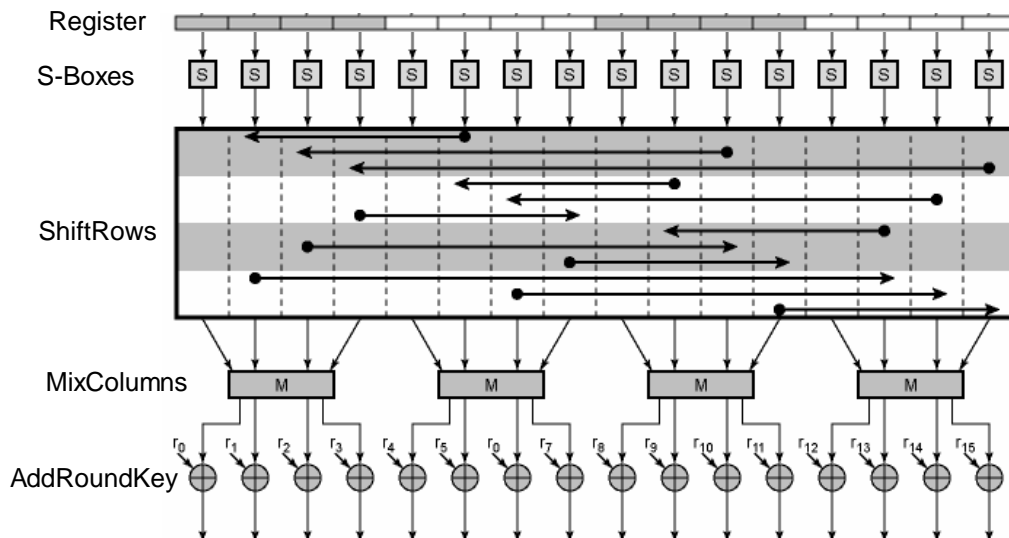

Figure 2: Typical AES Implementation

Typical hardware architecture of the AES with 16 S-Boxes is sketched in Figure 2. Here, sixteen 8-bit registers feed the 16 identical S-Boxes (S). Shiftrows involves only wires for shifting the bytes of the State, it operates on 128 bits. Four identical MixColumns blocks operate on 32 bits each.

Our goal is to identify a partitioning of the circuit that allows a repetition of identical sub-blocks. These sub-blocks will be compared two-by-two for on-line fault detection thanks to the implementation of an extra sub-block. In the classical architecture depicted in Figure 2, ShiftRows unfortunately prevents such a partitioning since it operates on all the 128 bits.

However by inspecting the AES algorithm, it can be seen that SubBytes and Shiftrows functions can be switched. We thus propose to perform ShiftRows before SubBytes, and even before loading the registers. Figure 3 depicts the proposed operation switching, where an RS block represents a Register/S-Boxes pair, whereas MA represents a MixColumns/AddRoundKey pair. In this figure we described the proposed method for a smaller part of the whole circuit. In particular, the figure depicts the reordering on 8 RS blocks and 2 MA blocks instead of the actual 16 and 4 ones, respectively.

The same procedure can be applied to the whole circuit and, as a consequence, the datapath can be divided in 4 identical slices that operate on 32 bits each, and that we call RSMA (32-bits **R**egister, 4 **S**-Boxes, 1 **M**ixcolumns and 32 xor for the **A**ddroundkey operation).
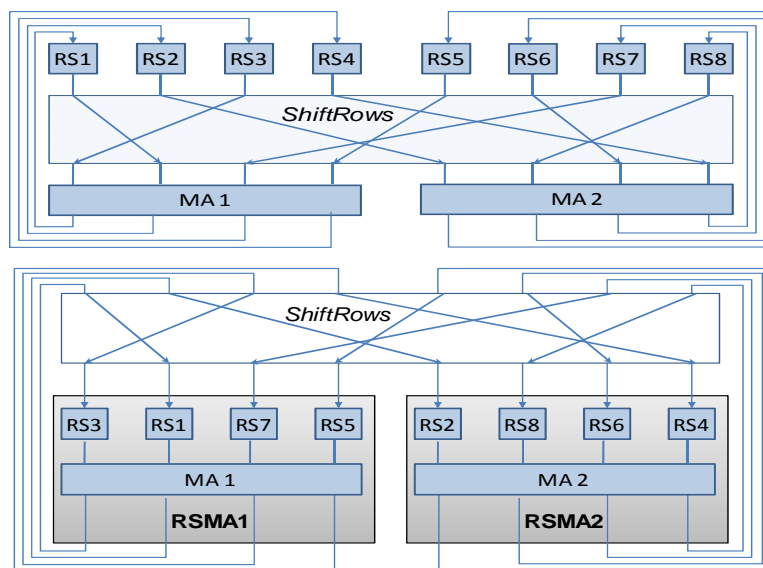

Figure 3: Switch between Shiftrows and SubBytes

The main idea of the proposed approach is to use one additional RSMA block, and to compare a pair of RSMA blocks at each clock cycle. In particular, at each clock cycle two blocks are fed by the same inputs and the related outputs are compared in order to detect possible faults. Figure 4 details the behavior of a part of the circuit where one extra RSMA block has been added. In this figure, LMux(2), LMux(3) and LMux(4) are multiplexers with an additional output that is asserted whenever the two inputs are equal (i.e., a multiplexer with a comparator).

Table 1 details the signals controlled and observed by the control unit. For instance, when RSMA4 and RSMA3 work together, the UMux(3) let the input I(4) go into the RSMA3. Among the five signals coming from the comparators, only one at a time is considered by the control unit. For example, in the above case, the check(4) signal is verified, i.e., the two related RSMA blocks are checked.
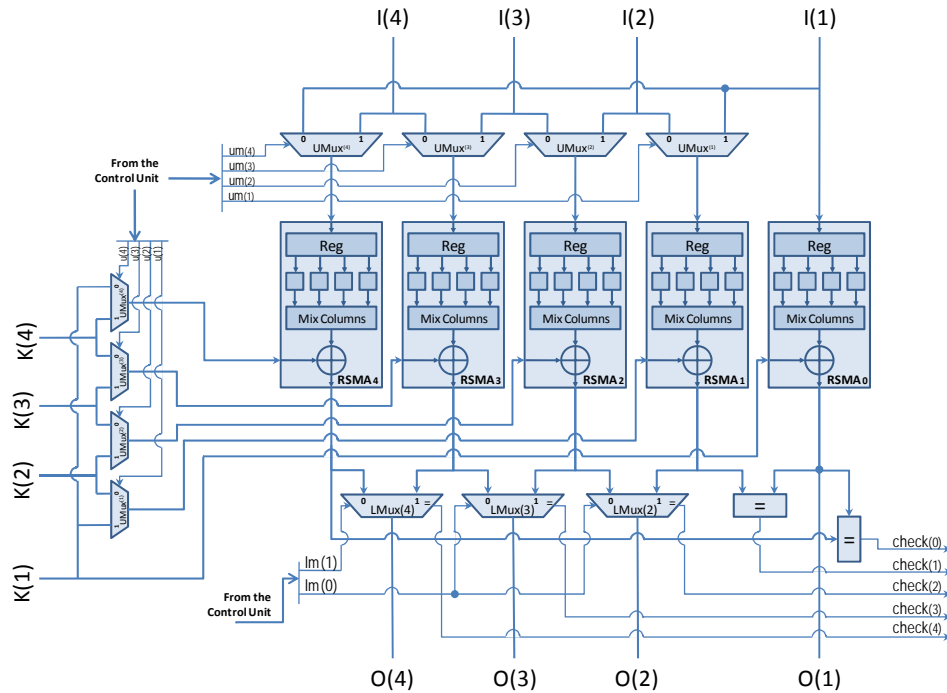

Figure 4: AES Architecture with RSMA duplication and comparison

The scheduling of the comparisons of a pair of RSMA blocks is a very important issue of the proposed method. One AES encryption lasts 10 clock cycles and there are 5 different configurations. Therefore it's possible to use each of the 5 configurations twice during one encryption. Through the 5 configurations, each

RSMA block is compared twice (once with the left block, once with the right block). Thus, if the 5 configurations are activated twice, each RSMA block is compared 4 times during one encryption. A counter is in charge of the test configuration scheduling.

Table 1: Signals controlled and observed by the Control Unit

Compared	Um	Lm	To check
RSMA4, RSMA3	1000	11	check(4)
RSMA3, RSMA2	1100	01	check(3)
RSMA2, RSMA1	1110	00	check(2)
RSMA1, RSMA0	1111	00	check(1)
RSMA0, RSMA4	0000	11	check(0)

4. Results and Fault Detection Analysis

This section provides results related to the area overhead and the fault detection analysis of the proposed approach. The proposed architecture has been described in VHDL and synthesized using Synopsys Design Compiler [9] using a 130nm CMOS library provided by STM [10]. We considered that all the keys used in the AddRoundKey step (see Section 2) are pre-computed and stored in the circuit.

The area of the original circuit is 52961 μm^2 (corresponding to 9660 logic cells) while the area of the proposed architecture is 71357 μm^2 (corresponding to 13084 logic cells and 34.7% of area overhead). The speed penalty is only 6% due to the insertion of the additional multiplexers.

Concerning the efficiency of the proposed architecture with respect to the fault detection, our functional redundancy strategy differs from the classical Double Modular Redundancy (DMR) scheme.

A classical DMR architecture allows detecting all the faults (single and multiple) that lead to an error (i.e., a difference at the output of one of the duplicated modules). Starting from the moment of appearance of the fault, the fault latency depends on the inputs applied to the circuit, only. In other words, the fault is detected as soon as the input vector can sensitize the fault and propagate it up to the output of the module (i.e., the input of the comparator between the two modules). Anyway, a system based on classical DMR scheme does not deliver faulty responses without noticing it (unless in case of equivalent faults in the two modules).

Our technique is able to detect any single or multiple fault leading to a wrong RSMA output value (as for the classical DMR) but only when the affected RSMA is compared with another one. Conversely to DMR, the dynamic reconfiguration of the modules leads to a comparison of each module twice every 5 clock cycles. Therefore it can happen that the system produces erroneous responses without noticing it even in presence of a single stuck-at.

We question here the probability to get an error on the AES output and to not detect it. This probability can be analyzed by computing the probability $P_{err}(f)$ of not detecting an error on the circuit's outputs while a given fault f affects the circuits. Here, only non-redundant faults are of interest, i.e. we focus on testable faults. For this analysis we focus on single stuck-at faults only, because the number of multiple faults is too high to be analyzed. However, unless the extremely very low probable case of multiple faults composed of 5 equivalent faults in the 5 RSMA modules, all multiple faults are covered by our technique.

With regard to the proposed architecture, $P_{err}(f)$ is the probability that the fault f is activated (i.e. sensitized and propagated in such a way that it leads to an error) during at least one of the 6 clock-cycles during which the faulty RSMA is not compared, and it is not activated during the 4 clock cycles when the RSMA is compared.

Let denote p_f the probability of activation of a fault f into an RSMA module, i.e., the probability that for a random input pattern the fault is sensitized and the error is propagated to its output. In the hypothesis to have several distinct functional inputs, we can consider that the device is fed by a random source. In addition, as demonstrated in [11], the inherent properties of the AES makes that the sequence of input values that are applied to consecutive rounds of the same encryption can be considered as random. Therefore, the probability p_f is equal to the ratio of input vectors that test f over the number of possible input vectors. The number of possible input vectors for the RSMA, is 2^{32}. Since fault simulation cannot be applied in exhaustive way, we split the problem in two parts. From one side, S-Boxes have 8 input bits only, consequently exhaustive analysis is possible and p_f can be obtained for each fault through simulation. Since MixColumns and AddRoundKey are invertible functions, all the errors appearing on the output of the S-Boxes propagate through the functions to the comparator. From the other

side, p_f of the MixColumns has been calculated thanks to its modular structure involving 4 identical 8-bit inputs sub-functions (2^8 combinations). Finally, the AddRoundKey operations involve only xor operations and are very easily tested. For both MixColumns and AddRoundKey, each fault is tested with p_f=50%. Since Sboxes are bijective functions, randomness properties are still kept at the input of MixColumns and AddRoundKey.

For the proposed architecture, the probability that f is not activated during the clock-cycles of comparison is equal to $(1-p_f)^4$ while the probability that f is activated during at least one of the clock-cycles without comparison is equal to $1-(1-p_f)^6$. Finally, it comes:

$$P_{err}(f)=(1-p_f)^4 \times (1-(1-p_f)^6)$$

(3)

Figure 5 represents $P_{err}(f)$ in function of p_f. It must be noticed that the hard-to-test faults $(p_f \cong 0)$ and the easy-to-test faults $(p_f \cong 1)$ are not those that most likely produce undetected errors. On the contrary, the maximum value (32.57%) corresponds to faults with p_f equal to 0.14.
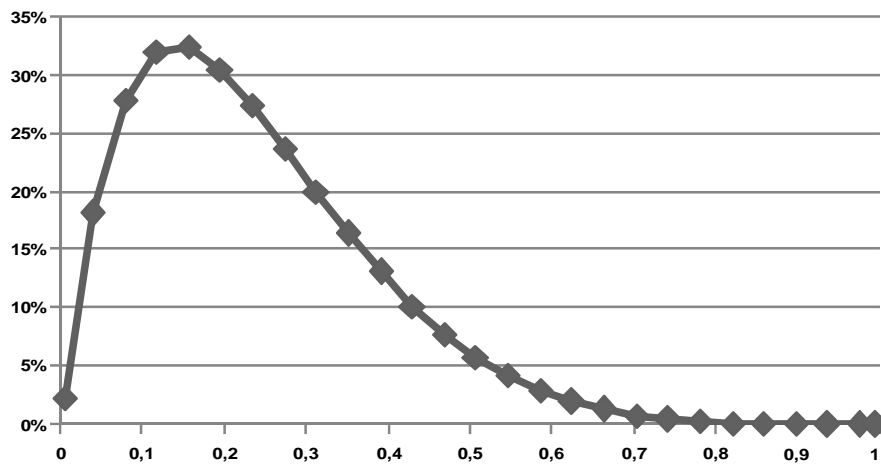

Figure 5: $P_{err}(f)$

In order to calculate the overall error probability, we simulated all the faults in the Sboxes to determine the distribution of probabilities of activation of the faults. An overall of 3860 stuck-at faults are present in our implementation. Basically, we calculated how many faults are activated by one test pattern (p_f = 1/256), how many faults are activated by 2 patterns (p_f = 2/256), and so on. Figure 6 summarizes, for each probability p_f, the number $FD(p_f)$ of faults with that activation probability.
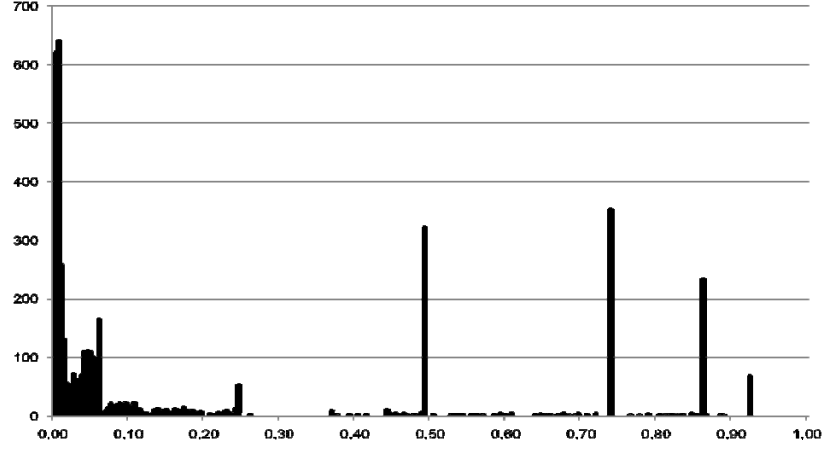
Figure 6: Fault Distribution FD(p$_f$)

Assuming that each fault has the same probability to appear in the circuit, the overall error probability $P_{ERR\text{-}Sbox}$ of each Sbox is calculated as the weighted average of the values P$_{err}$(f) according to the distribution FD(p):

$$P_{ERR-Sbox} = \frac{1}{\#Faults_{Sbox}} \sum_{i=1}^{256} \left\{ FD\left(\frac{i}{256}\right) \times \left[\left(1-\frac{i}{256}\right)^4 \times \left(1-\left(1-\frac{i}{256}\right)^6\right)\right]\right\} = 10.18\% \quad (4)$$

Concerning the MixColumn and AddRoundKey, there are #Faults$_{MCAK}$=750 faults, all of them with p$_f$=50%. Therefore the error probability $P_{ERR\text{-}Sbox}$ of MixColumns and AddRoundKey is calculated based on equation (3).

It comes that the overall probability P_{ERR} of the RSMA is:

$$P_{ERR} = \frac{\#Faults_{Sbox} \times 4 \times P_{ERR-Sbox} + \#Faults_{MCAK} \times P_{ERR-MCAK}}{\#Faults_{RSMA}} = 9.99\% \quad (5)$$

The architecture has thus a probability of 90.01% to detect any fault in the RSMA during a single encryption (10 clock cycles).

Let's now analyze the evolution of this probability based on the number of encryptions. When we perform E encryptions, an RSMA block is compared during 4·E clock cycles, while it is not compared during 6·E clock cycles.

The error probability can therefore be rewritten as follows:

$$P_{err}(E,f) = \left(1-p_f\right)^{4\cdot E} \times \left(1-\left(1-p_f\right)\right)^{6\cdot E} \quad (6)$$

Considering the fault distribution FD given in Figure 6 and the probability of error detection in the MixColumns and AddRoundKey, we can re-calculate the overall error probability P_{ERR} of the RSMA block in function of the number of encryptions (Figure 7). As it can be seen, the error probability slightly increases up to 14% for 5 encryptions, while for higher encryption numbers it tends to 0.

The error probability augmentation from 0 to 5 encryptions can be explained by the fact that at the beginning the probability to exercise the faulty module with random test patterns increases more quickly than the probability to compare the faulty module with a good one while exciting the fault. Since we focus on permanent fault, after a while (i.e. 5 encryptions) the probability to detect the fault (from comparison) is predominant. Namely, for 300 encryptions, the fault detection probability is equal to 99.9%
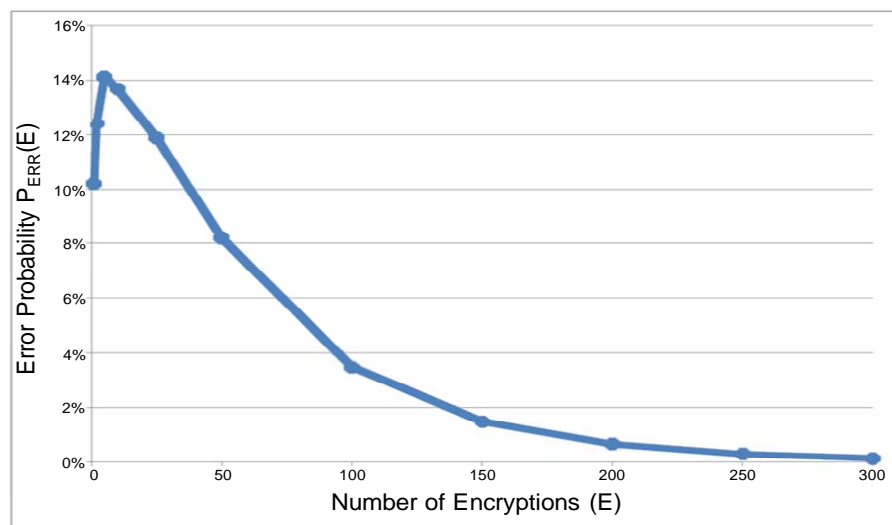

Figure 7: Error probability w.r.t. the number of encryptions

Equivalent curves would be obtained in case of 192-bit or a 256-bit key, since the datapath of the device does not change. The only difference in case of longer keys is a higher number of executed rounds. Therefore, we expect lower number of encryptions for achieving the same error probability.

5. Differential Power Analysis

An important issue when dealing with cryptographic cores is the sensitivity of the architecture implementation to side-channel attacks, in particular against Differential Power Analysis (DPA). We focused on this attack because among all the known possible attacks, this is one of the cheapest and easiest to perform. Basically, the DPA attack is a statistical technique relying on the correlation that exists between the current consumed by the device and the processed data.

We now introduce some theoretical issues that allow the reader to understand the principle underlying the DPA attack.

Let's consider the output of a gate whose state depends on both the plain text under ciphering (primary inputs) and the secret key. It is called the target node. Let's consider now a sequence of input patterns P_0, P_1, …, P_n that generate the transitions T_1 ($P_0 \rightarrow P_1$), T_2 ($P_1 \rightarrow P_2$), ..., $T_n(P_{n-1} \rightarrow P_n)$ on the circuit primary inputs. A logic simulation of the circuit while monitoring the target node allows classifying these input transitions in two sets, according to a guess on the key:

- PA, composed by the transitions that make the target node to commute from 0 to 1 and therefore that make the target gate to consume;
- PB, composed by the transitions that do not lead the target gate to participate to the power consumed by the circuit (i.e., transitions from 0 to 0, 1 to 1, and 1 to 0 on the target node).

Figure 8 represents the power consumption of the device when stimulated by numerous input vectors. We assume in this example that the guess on the secret key is correct. In other word, the simulation is performed with the key actually used in the circuit from which power consumptions are collected. Each rectangle represents the total power consumed by the circuit when a new vector is applied to the inputs. In this figure, and just for clarity of explanation, the power consumption is represented by a rectangle corresponding to the average of the consumption over the transition time. In the following, this issue will be re-defined in a more precise way. The set of transitions on the circuit inputs is split in the two sets: in the left part there are the PA transitions and the related consumptions while in the right part there are the PB transitions and their corresponding consumptions. A part of the power consumption related to the transitions belonging to PA is due to the power consumed by the target gate (shaded rectangles). Obviously, the commutation from 0 to 1 of non-target nodes also contribute to the power consumption of the circuit, however the input transitions that lead to such commutations are assumed to be evenly distributed to sets PA and PB. If a large number of transitions are considered, mean consumptions related to sets PA and PB are almost equal, except for the contribution of the target node.

In other words, since the two sets are classified in such a way that the set PA always leads to a component of power consumption that is not present in the set PB, the difference between the two mean powers computed from set PA and set PB must show a noticeable difference.
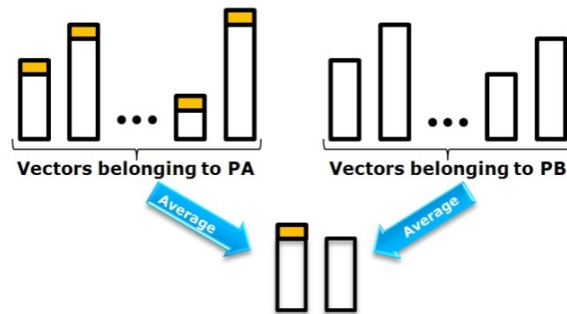

Figure 8: Power consumption after pattern partitioning

During a DPA attack, the target node is chosen in such a way that it depends on a small part of the key only, so that all the key suppositions can be considered. For example, when DPA is conducted against the AES, the target node is chosen as the output of an S-Box that depends on only 8 (of 128) secret key bits. Thus, only 256 key guesses are needed, instead of 2^{128}. The process is iterated on every S-Box.

For each key guess, the two sets PA and PB are created according to the results of the logic simulation and the key guess under evaluation. The power mean values are calculated for each set using the simulated power traces of the circuit under attack for each transition. Finally, the differences of the mean values of the two sets are calculated. When the key guess is correct (and only in this case), PA actually includes the input transitions that lead to a transition 0 to 1 on the target node while PB does not include any of these transitions. The difference between the mean power obtained from PA and PB can be observed in this case. On the contrary, when the curves are classed in PA or PB independently from the actual value of the secret key, the two average curves do not present any noticeable difference. The classification process is illustrated in Figure 9 where K_x is assumed to be the correct key, the one actually used during ciphering.
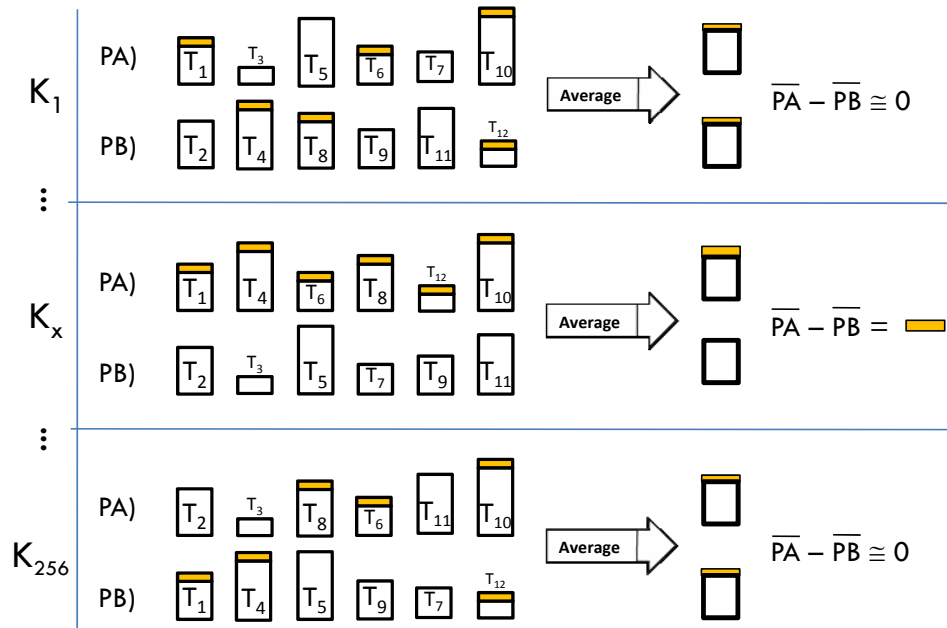
Figure 9: Pattern classification for several key guesses

It's important to note that the actual attack is performed by measuring and analyzing the instantaneous power consumptions over the whole transition period, and not using the time-averaged value as shown in Figures 8 and 9.

Figure 10 shows the appearance of the result of an attack over a period of 1ns and 8 key guesses. Each of the 8 curves represents the *DPA curve*, i.e. the difference between the mean powers issued from the transitions classified in sets PA and PB, in function of time. The curve that shows the higher peak (bold line in Figure 10) corresponds to the correct key guess.
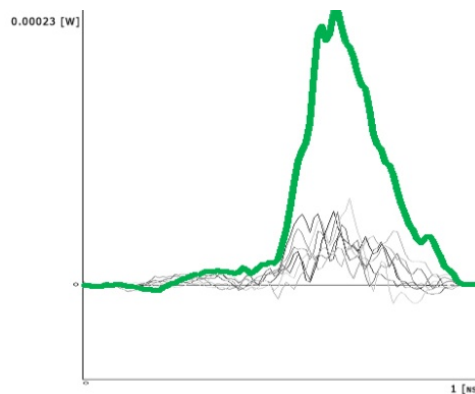

Figure 10: Real DPA curve

Concerning the proposed architecture, we performed DPA on the base AES architecture and on the proposed redundancy-based solution using an in-house DPA simulator [12]. We found that the DPA attack is slightly more difficult to

15

perform on the proposed architecture including one additional RSMA block and comparison circuitry. Figure 11 shows the DPA curves for the standard and the proposed implementations. The DPA attack succeeds when the curve with largest peak corresponds to the right secret key. After 256 encryptions the curve corresponding to the secret key clearly emerges among the others for the AES Standard implementation while, for our architecture, at least 512 encryptions are necessary to have the same confidence level. In fact, after only 256 encryptions the highest peak (red line) does not correspond to the correct key. This result is explained by the fact that, for the same input, this architecture has several power profiles based on the configuration of the pairs of RSMA blocks. In particular, for the same input, the circuit can be in 5 different states, i.e., 5 different power profiles.
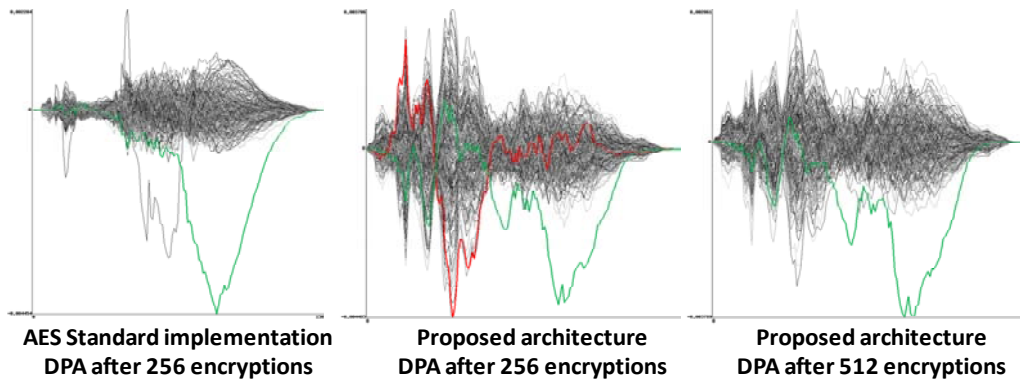

| **AES Standard implementation DPA after 256 encryptions** | **Proposed architecture DPA after 256 encryptions** | **Proposed architecture DPA after 512 encryptions** |

Figure 11: DPA Curves

6. Conclusions

In this paper we proposed a low cost architecture for detecting single and multiple faults in the hardware implementation of the Advanced Encryption Standard (AES) during its mission mode. The solution, based on spatial redundancy, reorders the AES algorithm subtasks. This modification does not influence the actual encryption function and it allows the implementation of 4 identical blocks working on 32-bits each. Thanks to this parallel and duplicated architecture, only one additional 32-bits block is added in the circuit leading to 4 tests per encryption cycle for every block. The solution is very effective in terms of fault latency and fault coverage while keeping the area overhead very low (about 34.7%). Finally, it is shown how the proposed implementation reduces the

correlation between the power consumption and the processed data, therefore it does not make easier attacks based on Differential Power Analysis.

References

[1] K. Wu, R. Karri, G. Kuznetsov, M. Goessel, "Low Cost Concurrent Error Detection for the Advances Encryption Standard", Proc. Int'l Test Conference, pp. 1242-1248, 2004

[2] G. Di Natale, M.-L. Flottes, B Rouzeyre, "An On-Line Fault Detection Scheme for SBoxes in Secure Circuits", Proc. IEEE Int. On-Line Testing Symposium, 2007, pp. 57-62

[3] R. Karri, K. Wu, P. Mishra, Y. Kim, "Concurrent Error Detection Schemes for Fault-Based Side-Channel Cryptanalysis of Symmetric Block Ciphers", IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 21, no. 12, Dec. 2002, pp. 1509-1517

[4] C. Yen, B. Wu, "Simple Error Detection Methods for Hardware Implementation of Advanced Encryption Standard", IEEE Trans Computers, vol. 55, no. 6, June 2006, pp. 720-731

[5] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, V. Piuri "Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard", IEEE Trans. Computers, vol. 52, no. 4, pp.492-505, Apr. 2003

[6] P. Kocher, J. Jaffe, B. Jun, "Differential Power Analysis", Proc. CRYPTO'99, 1999, pp 388-397

[7] "Advanced Encryption Standard (AES)", Federal Information Processing Standards Publication 197, November 26, 2001.

[8] X. Zhang, K. K. Parhi, "Implementation Approaches for the Advanced Encryption Standard Algorithm", IEEE Circuits and Systems Magazine, vol. 2, Issue 4, pp. 24-46, 2002

[9] http://www.synopsys.com

[10] http://www.st.com

[11] P. Hellekalek, S. Wegenkittl, "Empirical evidence concerning AES", ACM Trans. Model. Comput. Simul., Vol. 13, Issue 4 (Oct. 2003), pp 322-333.

[12] G. Di Natale, M.-L. Flottes, B. Rouzeyre, "An Integrated Validation Environment for Differential Power Analysis", IEEE International Symposium on Electronic Design, Test & Applications (DELTA 2008), Hong Kong, January 2008, pp. 527-532