On-Line Self-Test of AES Hardware Implementations

G. Di Natale, 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
{dinatale,flottes,rouzeyre}@lirmm.fr

Abstract

In this paper we propose an on-line self-test architecture for hardware implementations of Advanced Encryption Standard (AES). The solution assumes a parallel architecture and exploits the inherent spatial replications of this implementation. Because Substitution boxes (S-Box) represent the largest hardware in this architecture, we focus on faults affecting these S-Boxes and propose a trade-off between hardware overhead and fault latency. We show that our solution is very effective while keeping the area overhead very low. Moreover, this architecture does not weak the device with respect to side-channel attacks based on power analysis. On the contrary, it makes more difficult this type of attack.

1. Introduction

Cryptography enables to store sensitive information or transmit it across insecure networks (like the Internet) so that it cannot be read by anyone except the intended recipient.

The classic cryptanalysis is purely theoretical. On the other hand, when the function is implemented in hardware, specific attacks are possible because the attacker has access to the physical cryptographic device and he can play around with it. These types of attacks are called "Implementation Attacks" which target the cryptographic device itself. These attacks can range from the physical opening of the cryptographic device to changing and observing the environmental conditions, e.g. attacks based on the observation of the inherent leakage of the cryptographic device.

Among all the attacks proposed in the literature, Side-Channel Attacks exploit the fact that the cryptographic device itself leaks physical information during the processing of a cryptographic algorithm. This physical leakage (e.g., power dissipation, timing information, ...) can be captured externally and can then be used to compromise secret keys of cryptographic algorithms by using standard statistical tools.

A good cryptographic device must therefore ensure high reliability and dependability and, in addition, it must implement some countermeasures to prevent the possibility of gathering the secret code by mean of a side-channel attack.

In this paper we propose a low cost concurrent on-line Self-Test technique able to detect single and multiple faults in the hardware implementation of the Advanced Encryption Standard (AES). The solution, based on the spatial replication inherent to the parallel implementation of the AES, exploits the native property to have 16 identical repetitions of the same block (the S-Box).

We present a trade-off between hardware overhead and fault detection latency where one additional S-Box is added every 4 S-Boxes in the circuit. The S-Boxes represent the biggest part of the AES circuit, counting up to 95% of the whole circuit. Therefore, it is possible to obtain very good results in terms of reliability by protecting the S-Boxes only. We will prove that our solution is very effective while keeping the area overhead very low. Moreover, although not specifically designed to protect against tampering, this architecture makes more difficult side-channel attacks based on power analysis.

The paper is organized as follows. Section 2 introduces the basic concepts and the characteristics of the Advanced Encryption Standard algorithm. Section 3 summarizes the state-of-the-art on this topic, while Section 4 presents the On-Line Self-Test approach. Section 5 discusses the results in terms of area overhead and fault detection capability. Eventually, Section 6 concludes the paper.

2. Advanced Encryption Standard

The Advanced Encryption Standard (AES) [4] is a block cipher adopted as an encryption standard by the U.S. government. AES began immediately to replace the Data Encryption Standard (DES), which has been in use since 1976. AES outperforms DES in improved long-term security because of, among other things, larger key sizes (128, 192, or 256 bits).

Another major advantage of AES is the possibility of efficient implementation on various platforms. AES is suitable for small 8-bit microprocessor platforms and common 32-bit processors, and it is appropriate for dedicated hardware implementations. Hardware implementations can reach throughput rates in the gigabit range. Several hardware implementations for AES circuit have been proposed [5]. No matter the type of implementation, the most expensive part of the circuit in terms of area is the so called S-Box.

The AES algorithm's internal operations are performed on a two dimensional array of bytes called State. The State consists of 4 rows of bytes and each row has Nb bytes. Each byte is denoted by Si, j ($0 \le i < 4$, $0 \le j < Nb$). Since the block length is 128 bits, each row of the State contains Nb = 4 bytes. For sake of simplicity we focus on key length equal to 128 bits. 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. At the beginning of encryption or decryption, the array of input bytes is mapped to the State array as illustrated in Figure 1. The 128-bit block 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}.

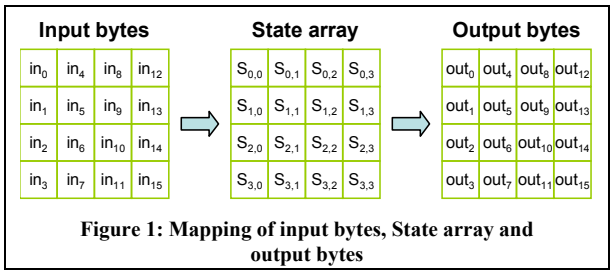

Figure 1: Mapping of input bytes, State array and output bytes

The AES algorithm is an iterative algorithm. Each iteration is called a round. The total number of rounds is 10. At the start of encryption, input is copied to the State array. After the initial roundkey addition, 10 rounds of encryption are performed. The first 9 rounds are identical, with small difference in the final round. As illustrated in Figure 2, 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 2 can be inverted to get a straightforward structure for decryption.

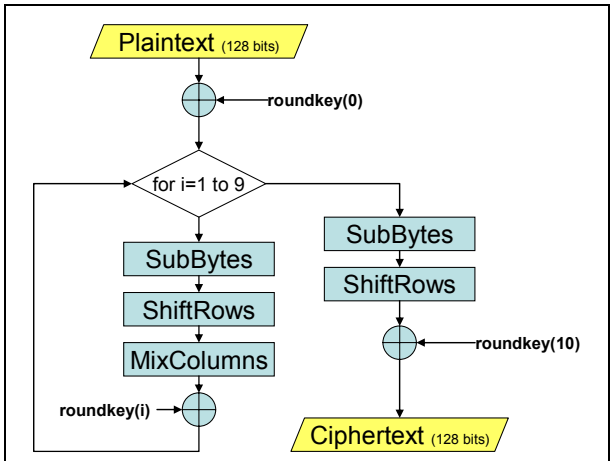

Figure 2: AES Algorithm (encryption)

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)):

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

for $0 \le i < 8$, where b_i is the i^{th} bit of the byte, and c_i is the i^{th} 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, therefore there are only 256 values. S-Boxes can be implemented either as a ROM or as combinational logic.

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:

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

AddRoundKey Transformation

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

Key Expansion

The key expansion routine, as part of the overall AES algorithm, takes the input key of 128 bits. The output is an expanded key of 11*128 bits, i.e., the expanded key is composed of the secret key and 10 roundkeys, one for each round. Details of the algorithm that allows determining the value of each roundkey are given in [4].

3. State-of-the-Art

Fault detection and tolerance schemes for various implementations of cryptographic algorithm have recently been considered. Several motivations led to increase the reliability of these circuits. From one side the circuit implementation of cryptographic algorithms can be quite area consuming, increasing the probability of device failures. Fault detection is therefore helpful in finding faults during the production tests. In addition, fault detection and tolerance schemes are very useful during mission time.

Since crypto chips are consumer products of mass production, cheap solutions for concurrent error detection and correction are of great importance. Mainly, two approaches have been developed: based on codes and based on functional redundancy.

All the techniques based on codes add some bits to the original data word. The main issue in this approach is the prediction of the value of the code, given the input value and the function executed by the circuit. For example, the prediction of a parity bit (when a parity bit is added to each byte) is almost straightforward for the ShiftRows, MixColumns and AddRoundKey steps because these transformations are either linear or they just perform some permutation of the position of the bits in the state array (see [6] for more details). On the contrary, the prediction of the parity bit is not trivial for the S-Box and a dedicated circuit must be added in order to calculate it. [6], [7], and [8] present a solution based on parity codes. In all cases the overhead is about 20% and the coverage of single faults is very high. Nevertheless, in case of multiple faults or in case of single faults that lead to an even number of errors, these solutions are not effective. Other solutions are based on the use of codes more complex and therefore more expensive, such as CRC [9] or systematic nonlinear robust codes [10], that allows reaching higher values of coverage in case of multiple faults. In this case the area overhead significantly increases (> 60%).

In [11] the authors propose a technique based on functional redundancy that can be used whenever the encryption and decryption modules are implemented on the same circuit. In this case after each encoding (or decoding) the plaintext (ciphertext) is decoded (encoded) again to check its correctness.

None of the previous works considered the characteristics of the proposed approach when the circuit is attacked by mean of power analysis. Only in [12] there is a comparative analysis for several codes of the correlation between the hamming distance of the processed data and the power consumption.

The technique that we propose guarantees high fault coverage of single and multiple faults and it makes the use of power analysis very difficult.

4. Architecture Description

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. The proposed solution is anyway applicable to AES implementations where the computation is performed in a semi-parallel way: for instance, AES with 4 or 8 S-Boxes can be modified in order to increase their dependability with the technique we propose in this paper. Typical hardware architecture of the AES where 16 S-Boxes are used at the same time is sketched in Figure 3.

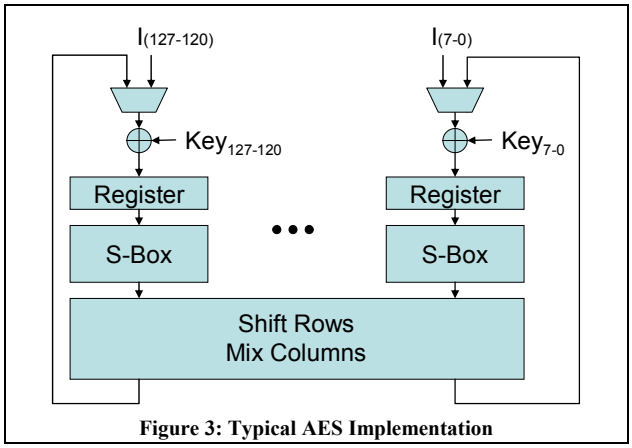

Figure 3: Typical AES Implementation

The most critical part of the circuit in terms of silicon area is composed by the registers and the S-Boxes, counting up to 85% of the whole circuit. In this paper we focus on the registers and the S-Boxes.
The main idea of the approach is to use one additional SubBytes block (8 bits register and S-Box) every 4 blocks, and to on-line test a pair of SubBytes blocks each clock cycle (see figure 4). 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.

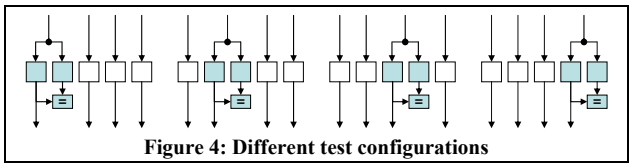

Figure 4: Different test configurations

Figure 5 details the behavior of a part of the circuit where one SubBytes block has been added to 4 blocks. In this figure, `LMux(2)` and `LMux(3)` are multiplexers with an additional output that is asserted whenever the two inputs are equal (i.e., a multiplexer with a comparator).

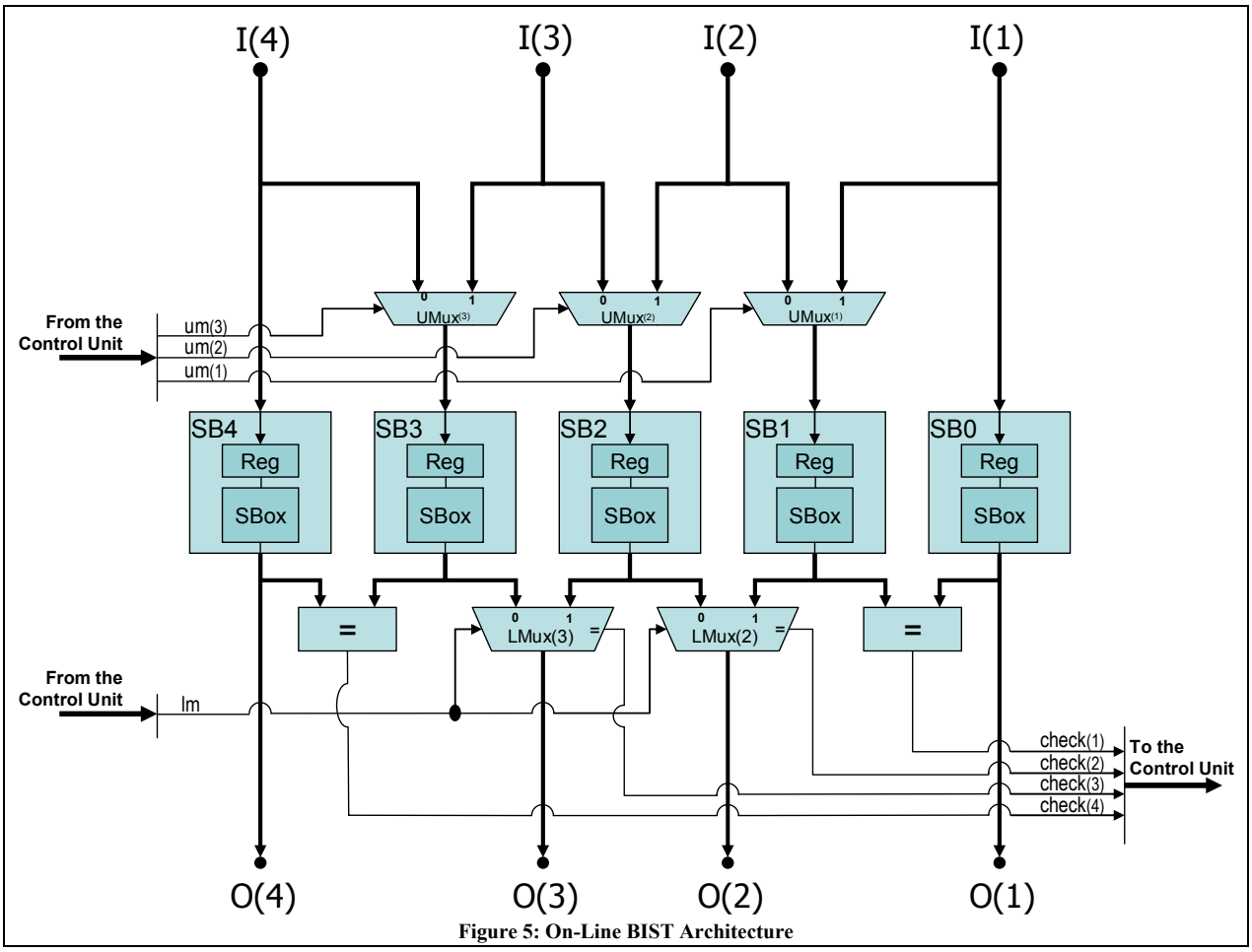

Figure 5: On-Line BIST Architecture

Table 1 details the signals controlled and observed by the control unit.

For instance, when `SB4` and `SB3` work together, the `UMux(3)` let the input `I(4)` go into the `SB3`. Among the four 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 related SubBytes block is checked.

Table 1: Signals controlled by the Control Unit

Working together	Um	Lm	To check
SB4, SB3	000	1	check(4)
SB3, SB2	100	1	check(3)
SB2, SB1	110	0	check(2)
SB1, SB0	111	0	check(1)

The scheduling of the pairs of SubBytes blocks that have to work together is a very important issue of the proposed method. The control unit must guarantee that each SubBytes block is tested at least once during the 10 rounds of the AES algorithm. A first solution is based on the use of a counter. In this way each SubBytes is tested at least twice (4 different combinations, repeat in 10 clock cycles, i.e., one for each AES round).

These considerations also lead to the fact that this technique is applicable for levels of redundancy starting from one spare SubBytes block every 10 blocks. In this way, there is the time to test each block before the end of the 10 rounds. We decided to add one spare block every 4 blocks because this solution has a low overhead (as shown in Section 5) while keeping the fault detection latency at an acceptable level (each block tested every 4 clock cycles).

In order to make more difficult the attacks based on fault injection or power analysis, the scheduling of the pairs of SubBytes blocks can be randomly performed. Instead of using a counter for the selection of which pairs must be on-line tested, it's possible to use an LFSR or a True Random Number Generator. In this case, the control unit has to guarantee that anyway all the combinations of pairs are tested before the end of the 10 rounds. Some considerations will be analyzed in Section 5.

5. Experimental Results

In this section we provide some results related to the area overhead and the fault coverage of the proposed approach.

The proposed architecture has been described in VHDL and synthesized using Cadence RTL Compiler [13]. We used the 0.35µm CMOS library provided by Austria Micro Systems [14].

All the S-Boxes have been synthesized as combinational logic. However, the proposed solution can be implemented using a ROM for the S-Box. Moreover, we implemented two different versions of the circuit: in the first version we considered that all the keys used in the AddRoundKey step (see Section 2) were pre-computed and stored in the circuit; in the second version we also implemented the module able to generate all the round keys. In this case, since the Key Generator uses 4 S-Boxes, we implemented the same architecture of Figure 5 for this module.

Table 2 summarizes the area of the circuit described in Figure 5.

Table 2: Area

	Base		Redundant	
	# Cells	Area [μm^2]	# Cells	Area [μm^2]
Registers S-Boxes	9590	642563	2397	160640
MixColumns ShiftRows AddKey	552	82918	0	0
Control Unit	55	4332	275	32067
Total (w/o Key Generator)	**10197**	**729813**	**2672**	**192707**
Key Generator	2805	228210	746	46937
Total	**13002**	**958023**	**3418**	**239644**

The area overhead of the first version is 26,40% (corresponding to a gate overhead of 26,20%) while the overhead of the second version is 26,28% (corresponding to a gate overhead of 25,01%).

This technique is able to detect any fault (single or multiple) that leads to a different output value of the S-Box while it is tested.

Regarding the protection against attacks based on power analysis, it's important to notice that this architecture has several power profiles based on the configuration of the pairs of SubBytes blocks. In particular, for the same input, the circuit can be in 4 different states (see Figure 4) for each of the 4 groups of 5 SubBytes blocks. This leads to a 256 different combinations of pairs, i.e., 256 different power profiles. This characteristic makes the power analysis extremely difficult, particularly when the scheduling of the pairs of SubBytes blocks is randomly selected because in this case the 256 power profiles are not cyclically repeated.

6. Conclusions

Cryptosystems are inherently computationally complex, and in order to satisfy the high throughput requirements of many applications, they are often implemented by means of VLSI devices.

In this paper we proposed a low cost concurrent on-line self-test technique able to detect faults in the registers and in the S-Boxes of the AES.

The solution, based on spatial redundancy, exploits the native AES property to have 16 identical repetitions of the same block (the S-Box). We presented a trade-off between hardware overhead and fault latency where one additional S-Box is added every 4 S-Boxes in the circuit leading to 2 tests of every S-Box per encryption cycle (10 rounds).

The solution is very effective while keeping the area overhead very low (about 26%). Moreover, although not specifically designed to protect against

tampering, this architecture makes more difficult side-channel attacks based on power analysis.

This solution can be easily enhanced with the use of other detection and/or correction techniques like those based on codes.

We consider as future work the development of an advanced architecture with 2 additional S-Boxes for each block of 4 S-Boxes, able to detect and repair faults in the circuit.

7. References

[1] E. Biham, A. Shamir, "Differential fault analysis of secret key cryptosystems," In Advances in Cryptology – CRYPTO'97, LNCS 1294, pp. 513–525, Springer-Verlag, 1997

[2] P. Dusart, G. Letourneux and O. Vivolo, "Differential Fault Analysis on A.E.S.," Cryptology Archive of IACR, No. 010, 2003, available at http://eprint.iacr.org/2003/010

[3] C. Giraud, "DFA on AES," Cryptology Archive of IACR, No. 008, 2003, available at http://eprint.iacr.org/2003/008

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

[5] 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

[6] 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

[7] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, V. Piuri, "A parity Code Based Fault Detection for an Implementation of the Advanced Encryption Standard", Proc. IEEE Int. Symposium on Defect and Fault Tolerance in VLSI, pp. 51-59, Nov. 2002

[8] V. Ocheretnij, G. Kouznetsov, R. Karri, M. Gossel, "On-Line Error Detection and BIST for the AES Encryption Algorithm with Different S-Box Implementations", Proc. IEEE Int. On-Line Testing Symposium, 2005, pp. 141-146

[9] 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

[10] M Karpovsky, K. J. Kulikowski, A. Taubin, "Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard", Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04), pp. 93-101

[11] 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

[12] V. Maingot, R. Leveugle, "On the Use of Error Correcting Codes in Secured Circuits", Proc. IEEE Latin-American Test Workshop (LATW07), 2007

[13] http://www.cadence.com

[14] http://asic.austriamicrosystems.com/databooks/index.html

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