A Novel Parity Bit Scheme for SBOX in AES Circuits

Giorgio Di Natale, Marie-Lise Flottes, Bruno Rouzeyre

A Novel Parity Bit Scheme for SBox in AES Circuits

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 – This paper addresses an efficient concurrent fault detection scheme for the SBox hardware implementation of the AES algorithm. Concurrent fault detection is important not only to protect the encryption/decryption process from random and production faults. It will also protect the system against side-channel attacks, in particular fault-based attacks, i.e. the injection of faults in order to retrieve the secret key. We will prove that our solution is very effective while keeping the area overhead very low.

I. INTRODUCTION

Cryptographic algorithms play a crucial role in the information society. When we use teller machines, home banking services or credit cards, call someone on a mobile phone, get access to health care services, or buy something on the web, cryptographic algorithms are used to offer protection. These algorithms guarantee that nobody can steal our money, place a call at our expense, eavesdrop on our phone calls, or get unauthorized access to sensitive health data. Information technology keeps changing and will become increasingly pervasive, while disappearing from the eye of the user. However, this evolution keeps presenting new security challenges, and there is no doubt that cryptographic algorithms and protocols will form an important part of the solution.

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 complex, increasing the probability of device failures. Fault detection is therefore helpful in finding faults during the production tests. In addition, fault tolerance schemes are very useful to on-line tolerate faults during mission time. From the other side, intentional intrusions and attacks based on the malicious injection of transient faults into the device are very efficient in order to extract the secret key [1] [2].

The Advanced Encryption Standard (AES) [3] 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 had been in use since 1976. AES outperforms DES in improved long-term security because of larger key sizes (128, 192, and 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 [4]. No matter the type of implementation, the most expensive part of the circuit in terms of area is the so called SBox. Section 2 will describe in detail the algorithm of the AES and in particular the definition and the characteristics of the SBox. In this paper we focus on a novel parity bit scheme to protect the SBox core.

Conversely to the other computational blocks of the AES algorithm, the SBox performs an operation that is not linear and is not invariant with respect to the parity of the processed data, i.e., the parity bit is not preserved after the transformation. This is the reason why it is necessary to insert an additional circuit able to predict the value of the output parity bit starting from the input value.

In this paper, we present novel low cost concurrent error detection (CED) S-Box architecture for the AES. Compared to previous works, our solution has higher fault coverage and lower area overhead.

The paper is organized as follows. Section 2 describes the characteristics of the Advanced Encryption Standard algorithm. Section 3 summarizes the state-of-the-art on this topic. Section 4 presents the parity-based concurrent error detection approach, whereas Section 5 discusses the results in terms of fault detection capability and area overhead, and compares these results with those published in the literature. Eventually, Section 6 concludes the paper.

II. ADVANCED ENCRYPTION STANDARD

The Rijndael algorithm used for the AES standard implements a symmetric-key cryptographic function in which both the sender and receiver use a single key to encrypt and decrypt the information.

Although in [5], the block length of Rijndael can be 128, 192, or 256 bits, the AES algorithm [3] only adopted the block length of 128 bits. Meanwhile, the key length can be 128, 192, or 256 bits. 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 Fig. 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}.

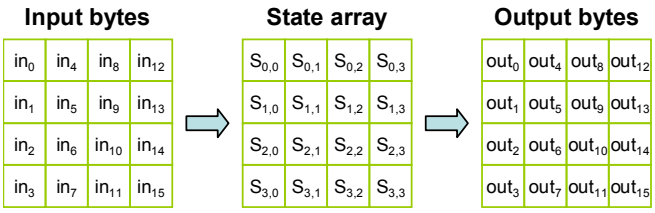

Fig. 1: Mapping of input bytes, State array and output bytes

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 the same, with small difference in the final round. As illustrated in Fig. 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 structure in Fig. 2 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 (SBox). This SBox 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. Implementations of the SBox are discussed in Section 3.

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.

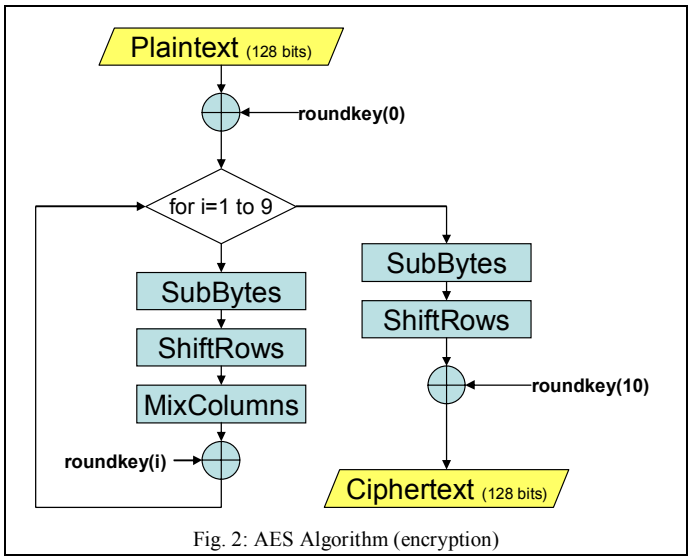

Fig. 2: AES Algorithm (encryption)

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 is described are given in [3].

III. STATE-OF-THE-ART

Since crypto chips are consumer products of mass production, cheap solutions for concurrent error detection and correction are of great importance [6] [7] [8] [9]. A natural choice for concurrent error detection is the application of parity codes. Concurrent checking for the AES by parity prediction was first introduced in [11] and [12]. One of the main problems targeted in the literature is the prediction of the output parity given the input state and the input parity bit.

The prediction of the parity bit (when a parity bit is added to each byte) is almost straightforward for the ShiftRows, MixColumns and AddRoundKey steps [11]. On the contrary, the prediction of the parity bit is not trivial for the Sbox. In this section we summarize the solutions based on the parity bit for the SBox.

The SBox is usually implemented either as a 256x8 bits memory consisting of a data storage section and an address decoding circuit, or a combinational circuit. The incoming data bytes will normally have properly generated even parity bits. A solution to generate the outgoing parity bits is proposed in [12] and sketched in Fig. 3.a: an even parity bit is either stored with each data byte in the SBox (memory implementation), or on-line generated with an ad-hoc combinational circuit (in the case of combinational logic implementation for the SBox). This solution is not very expensive and it guarantees acceptable fault coverage.

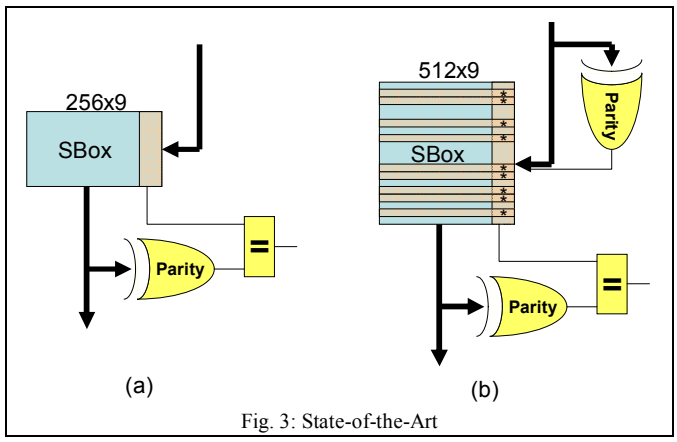

Fig. 3: State-of-the-Art

To increase the dependability and to detect additional input parity errors and some internal memory errors (data or decode), [11] proposes replacing the original 8-bit decoder with a 9-bit one, yielding a 512x9 bits memory (Fig. 3.b). If a 9-bit address with an even parity is decoded, the corresponding output byte with its associated even parity bit is produced. Otherwise, a constant word of 9 bits with a deliberately odd parity is output, e.g., "00000000 1". Thus, half of the entries in the SBox memory will be deliberately wrong (in the figure, all the rows marked with a '*'). In case of a single error in the input value, a wrong cell will be addresses. That cell will contain an erroneous parity bit that will be detected during the parity bit check. This solution guarantees higher fault coverage but it's very expensive in terms of used area.

Section 5 gives some comparative results.

IV. ARCHITECTURE DESCRIPTION

In this paper we focus on the use of the parity code for a single Sbox. We propose a solution that is suitable for all the schemes where there is a parity check for each byte element of the matrix S. The main problem in implementing the parity bit for the SBox is related to the fact that the SBox transformation is not invariant with respect to the parity bit. Hence, it is necessary to implement a method to predict the output parity, given the input value.

In order to meet higher fault detection capability a code-based fault detection approach has been adopted, consisting of information redundancy applied to both data and address of the memory storing the SBox values. With this solution we are able to target Address Faults as well. An Address Faults typically causes that during a read operation an unexpected cell is accessed by a given address. The use of detection codes based on both the address and the data allow the detection of Address Faults [13].

One characteristic of the Sbox is that it implements an invertible function. This feature allows calculating the input value starting from the output response. Therefore it is possible to predict the parity bit of the input word starting from the output response of the SBox (without implementing the inverse function, see below for details).

The main idea is that we do not add any parity bit in the memory that stores the SBox values (or into the combinational logic that implements it). On the contrary, we calculate the parity of the input value and we compare it with the parity bit predicted starting from the output value of the Sbox. In addition, we calculate the parity bit of the output of the SBox and we compare it with the prediction of this bit starting from the input value. All works presented in the past were based on the predictor of the output parity.

We calculated the Output Parity Predictor and the Input Parity Predictor using the truth tables of the SBox and of the parity bits, calculated for both the input value and the output value. Fig. 4 shows the first elements of the truth tables.

This scheme allows double protection of the SBox circuit and it should allow covering more faults than the architectures proposed in the literature. Section 5 will prove that actually this scheme is more effective.

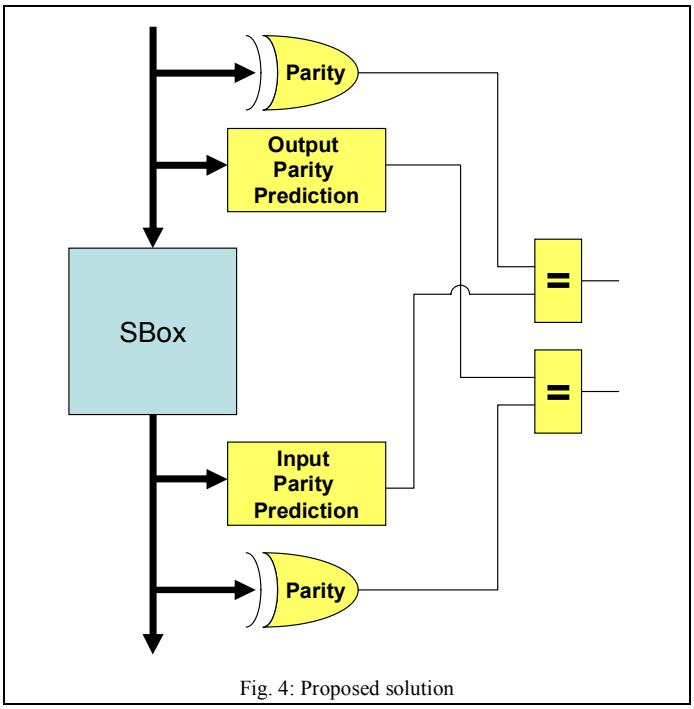

Fig. 4: Proposed solution

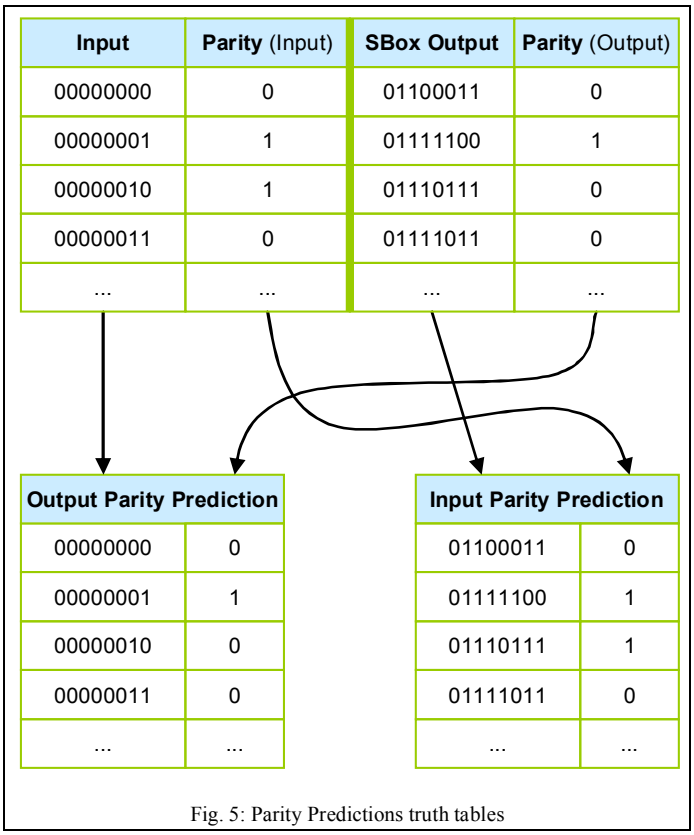

Fig. 5: Parity Predictions truth tables

V. EXPERIMENTAL RESULTS

In this section we provide some results related to the area overhead and the fault coverage of the proposed approach. We also compare these results with the architecture proposed in [11] and [12].

The architecture proposed in Fig. 4 has been described in VHDL and synthesized using Synopsys Design Compiler. Both the SBox and the prediction circuits have been synthesized as combinational logic. However, the proposed solution can be implemented using a ROM for the SBox.

We used the 0.35µm CMOS library provided by Austria Micro Systems [14].

The architecture has been implemented in two different ways in order to better assess the detection capabilities of the proposed approach. One synthesis has been performed posing as a constraint the minimization of the area. The second synthesis has been optimizing for the speed of the circuit.

Table 1 summarizes the area of the circuit described in Fig. 4, with both results (area optimization and speed optimization).

The area overhead is 37,35% with area optimization and 37,92% with speed optimization.

In order to measure the detection capability of the proposed architecture we used the fault simulator provided by Synopsys (TetraMax). The circuit has been modified in such a way that the only output signals visible by the fault simulator are the comparator signals. In this way, the obtained fault coverage gives a measure of the detection capability when a single error affects the circuit. In this experiment we focused on single stuck-at faults. The obtained fault coverage is equal to 99,20% for the circuit synthesised with area optimization and 99,25% for the speed optimization.

TABLE 1
AREA

	Area Optimization		**Speed Optimization**	
Module	**# Cells**	**Area** [µm²]	**# Cells**	**Area** [µm²]
SBox	555	34780	566	35672
InPrediction	90	5569	99	6061
OutPrediction	93	5879	94	5923
Parity	8	1420	8	1420
Comparators	2	124	2	124

Table 2 summarizes some comparison between our solution and the architectures proposed in [12] and [11], sketched respectively in Fig. 3.a and Fig. 3.b. Those architectures have been synthesized using the same technological library. In both cases the SBox has been implemented as combinational logic.

TABLE 2
COMPARISON

Architecture	**Area Overhead**	**Fault Coverage**
Our approach	37,35%	99,20%
[12] (Fig. 3.a)	18,17%	91,95%
[11] (Fig. 3.b)	47,28%	93,43%

The solution proposed in [12] allows covering 91,95% of the faults only, guaranteeing anyway a lower area overhead.

The solution proposed in [11] guarantees higher fault coverage than the solution proposed in [12], but it has a very high area overhead (47,28%). In addition, the area overhead is even higher when the ROM is used to implement the SBox. In this case the overhead is about 125%.

In any case, our solution guarantees higher fault coverage and, thanks to the double prediction based on both address and data, it would allow covering a percentage of address faults when the SBox is implemented as a ROM.

VI. CONCLUSIONS

Crypto-systems 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.

The high complexity of such implementations raises concerns regarding their reliability. Research is therefore needed to develop methodologies and techniques for designing robust cryptographic systems, and to protect them against both accidental faults and intentional intrusions and attacks, in particular those based on the malicious injection of faults into the device for the purpose of extracting the secret information.

The introduction of the parity bit prediction, both in input and output, increased significantly the fault coverage of the circuit, without resorting to expensive solutions requiring large extra memory area.

We consider as future work the development of a scheme for concurrent error detection with double prediction of the parity bits using a pipelined architecture in order to lower even more the hardware overhead.

REFERENCES

[1] D. Boneh, R. DeMillo, R. Lipton, "On the Importance of Eliminating Errors in Cryptographic Computations", Journal of Cryptology, vol. 14, pp. 101-119, 2001

[2] M. Akkar, C. Giraud, "An Implementation of DES and AES, Secure against some Attacks", Proc. Of CHES'01, pp. 315-325, 2001

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

[4] 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

[5] J. Daemen, R. Rijmen,, "AES Proposal: Rijndael", version 2, 1999, Available at http://www.esat.kuleuven.ac.be/~rijmen/rijndael

[6] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, V. Piuri, "Detecting and Locating Faults in VLSI Implementations of the Advanced Encryption Standard", Proc. 18th IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems, pp. 105-113, Nov. 2003

[7] 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

[8] 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

[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] 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

[11] 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

[12] 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

[13] A. Benso, S. Chiusano, G. Di Natale, M. Lobetti-Bodoni, P. Prinetto, "On-line & Off-line BIST in IP-Core Design", IEEE Design and Test of Computers, September/October 2001, Vol. 18, N. 5, pp. 92 99

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