Concurrent Structure-Independent Fault Detection Schemes for the Advanced Encryption Standard

Mehran Mozaffari-Kermani, *Student Member*, *IEEE*, and Arash Reyhani-Masoleh, *Member*, *IEEE*

Abstract—The Advanced Encryption Standard (AES) has been lately accepted as the symmetric cryptography standard for confidential data transmission. However, the natural and malicious injected faults reduce its reliability and may cause confidential information leakage. In this paper, we study concurrent fault detection schemes for reaching a reliable AES architecture. Specifically, we propose low-cost structure-independent fault detection schemes for the AES encryption and decryption. We have obtained new formulations for the fault detection of SubBytes and inverse SubBytes using the relation between the input and the output of the S-box and the inverse S-box. The proposed schemes are independent of the way the S-box and the inverse S-box are constructed. Therefore, they can be used for both the S-boxes and the inverse S-boxes using lookup tables and those utilizing logic gates based on composite fields. Our simulation results show the error coverage of greater than 99 percent for the proposed schemes. Moreover, the proposed and the previously reported fault detection schemes have been implemented on the most recent Xilinx Virtex FPGAs. Their area and delay overheads have been compared and it is shown that the proposed schemes outperform the previously reported ones.

Index Terms—Advanced encryption standard, concurrent error detection (CED), reliability, signature-based fault detection.

1 Introduction

The National Institute of Standards and Technology initiated a process to select a symmetric key encryption/decryption algorithm in 1997. Finally, Rijndael algorithm was accepted among other finalists as the Advanced Encryption Standard (AES) in 2001 [1]. The fast hardware and software implementations and the high level of security of the AES have led to its widespread usage in different critical applications needing reliable systems and architectures [1], [2]. As an example, the AES has been lately utilized for the bitstream security mechanisms in the FPGAs for increasing the reliability of the FPGA-based designs [3], used in the recent Xilinx Virtex FPGA families [4].

The objective in using the AES is to transfer the data so that only the desired receiver with a specific key would be able to retrieve the original data. However, with the existence of malicious injected faults in nonsecure environments, the hardware implementation of the AES does not guarantee that the data are transferred reliably. In fact, several fault attacks on the AES are reported in the literature, see, for example, [5], [6], and [7]. These types of attacks are based on injecting faults to the structure of the AES to obtain the confidential information or simply to cause malfunctioning of the AES algorithm. This results in the incorrect output for the AES designs. To make a robust implementation against these attacks, several fault detection schemes have been proposed to date, see, for example, [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], and [21].

There exist a number of fault detection schemes based on the error detecting codes, see [8] and [9] for the basic parity-based schemes. Using a parity bit for each byte in the AES encryption transformations has been presented in [8]. In [9], a general method of concurrent checking for Substitution Permutation Networks (SPN) has been proposed. These were followed by other fault detection schemes for the entire AES, see, for example, [10], [11], [12], and [13]. In these schemes, the output parity bits of each transformation in every round are predicted from the inputs of the corresponding transformation. Then, the comparisons between the predicted parities and the actual parities can be scheduled so that the desired error coverage is obtained.

In [14] and [15], the redundant unit fault detection scheme is used, where algorithm-level, round-level, or operation-level fault detections are considered. In this scheme, a transformation or a round or the entire encryption/decryption is followed by its inverse and the result is compared with the original input to obtain the error indication flag. Although almost all the faults in this fault detection scheme are detected, it suffers from the area and delay overheads of at least 100 percent. The scheme in [16] proposes using the transformations in an AES round twice for the same data to detect the transient errors. However, this scheme is unable to detect the permanent internal faults or the malicious injected faults lasting for a long period. In [17], a fault detection scheme based on the merged S-boxes and inverse S-boxes is proposed.

A multiplication-based scheme is presented in [19]. In this scheme, the result of the multiplication of the input and the output of the multiplicative inversion is compared with the predicted result of unity. However, this scheme is not

• *The authors are with the Department of Electrical and Computer Engineering, Faculty of Engineering, The University of Western Ontario, 1151 Richmond Street North, Thompson Engineering Building, London, ON N6A 5B9, Canada. E-mail: mmozaff@uwo.ca, areyhani@eng.uwo.ca.*

Manuscript received 31 Oct. 2008; revised 26 Feb. 2009; accepted 21 May 2009; published online 29 Jan. 2010.
Recommended for acceptance by C. Bolchini and D. Sciuto.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-2008-10-0540.
Digital Object Identifier no. 10.1109/TC.2010.33.

suitable for the S-boxes and inverse S-boxes implemented using lookup tables (LUTs). This is because the output (the input) of the multiplicative inversion in the S-box (the inverse S-box) may not be accessible in the LUT-based implementations. Therefore, the fault detection scheme presented in [19] is not applicable for these implementations.

In this paper, we present structure-independent fault detection schemes for obtaining a reliable AES implementation. We summarize our contributions as:

- We have presented a systematic method for obtaining the fault detection signatures for the multiplicative inversion of the S-boxes (inverse S-boxes).
- We have proposed new formulations resulting in novel fault detection schemes for checking SubBytes, inverse SubBytes, and the other transformations in the encryption and the decryption of the AES. The proposed schemes are independent of the method the S-box (respectively, the inverse S-box) is implemented. Thus, they can be applied to both the LUT and composite fields implementations.
- We have simulated the proposed fault detection structures for the AES encryption and decryption. Through our simulations after injecting up to 700,000 random stuck-at errors, we have shown that the proposed low-cost schemes reach the error coverage of greater than 99 percent.
- Finally, our proposed fault detection schemes and almost all of the previously reported ones have been implemented on the recent Xilinx Virtex FPGAs, and their area and delay overheads have been derived and compared. The FPGA implementation results show the low area and delay overheads for the proposed fault detection schemes.

The organization of this paper is as follows: In Section 2, preliminaries regarding the AES algorithm are explained. The proposed structure-independent schemes for the fault detection of the S-boxes and the inverse S-boxes are presented in Section 3. Then, the fault detection schemes for the entire AES encryption and decryption are considered in Section 4. In Section 5, the results of the simulations of the proposed schemes are presented and their error coverages are obtained. In Section 6, the presented fault detection schemes and the previously reported ones are implemented on FPGAs and they are compared in terms of time and space complexities. Finally, conclusions are made in Section 7.

2 Preliminaries

In this section, we briefly explain the four transformations of each round of the encryption and the decryption in the AES. In the implementations of the AES-128 (128-bit key) transformations, the irreducible polynomial of $P(x) = x^8 + x^4 + x^3 + x + 1$ is used to construct the binary field $GF(2^8)$. Each transformation in every round acts on its 128-bit input denoted as the *state*. The states are considered as 4×4 matrices whose entries are 8 bits. For example, the input state S with its 8-bit entries, i.e., $s_{r,c}$, $0 \leq r, c \leq 3$, is represented as follows:

$$S = [s_{r,c}]_{r,c=0}^{3}. \tag{1}$$

2.1 AES Encryption

Considering (1) as the input state of an encryption round, the transformations in each round of encryption (except for the last round) are as follows [1]:

- **SubBytes**: The first transformation in each round is the bytes substitution (*SubBytes*) implemented by 16 S-boxes. Let $s_{r,c} \in GF(2^8)$ and $s'_{r,c} \in GF(2^8)$ be the 8-bit input and output of each S-box, respectively. Then, the S-box consists of a multiplicative inversion, i.e., $s_{r,c}^{-1} \in GF(2^8)$, followed by an affine transformation consisting of the matrix Γ and the vector γ to generate the output as

$$s'_{r,c} = \Gamma s_{r,c}^{-1} + \gamma$$
$$= \begin{pmatrix} 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&0&0&1&1&1&1&1 \end{pmatrix} s_{r,c}^{-1} + \begin{pmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{pmatrix}. \tag{2}$$

The 8-bit outputs of 16 S-boxes are used to obtain the output state of the SubBytes transformation as

$$S' = [s'_{r,c}]_{r,c=0}^{3}. \tag{3}$$

- **ShiftRows**: In the second transformation, *ShiftRows*, 4 bytes of the rows of the input state are cyclically shifted to the left and the first row is left unchanged to obtain the output state, i.e., $SR(S')$, as

$$SR(S') = \begin{pmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3}\\ s'_{1,1} & s'_{1,2} & s'_{1,3} & s'_{1,0}\\ s'_{2,2} & s'_{2,3} & s'_{2,0} & s'_{2,1}\\ s'_{3,3} & s'_{3,0} & s'_{3,1} & s'_{3,2} \end{pmatrix} = [s'_{r,(r+c)\ mod\ 4}]_{r,c=0}^{3}. \tag{4}$$

- **MixColumns**: In the third transformation, *MixColumns*, the output state is obtained by multiplying a constant matrix with the output state of ShiftRows, $SR(S')$ in (4), to obtain the output state of MixColumns, i.e., the matrix S'', as

$$S'' = [s''_{r,c}]_{r,c=0}^{3} = \begin{pmatrix} \{2\}_h & \{3\}_h & \{1\}_h & \{1\}_h\\ \{1\}_h & \{2\}_h & \{3\}_h & \{1\}_h\\ \{1\}_h & \{1\}_h & \{2\}_h & \{3\}_h\\ \{3\}_h & \{1\}_h & \{1\}_h & \{2\}_h \end{pmatrix} SR(S'). \tag{5}$$

- **AddRoundKey**: The final transformation is *AddRoundKey* in which the input state is added (modulo-2) with the key of the round. Considering the roundkey input state as the matrix $K = [k_{r,c}]_{r,c=0}^{3}$, with entries $k_{r,c}$, $0 \leq r, c \leq 3$, the output state of the AddRoundKey transformation, i.e., $\boldsymbol{O}$, is obtained as

$$\boldsymbol{O} = [o_{r,c}]_{r,c=0}^{3} = \boldsymbol{S}'' + \boldsymbol{K}. \tag{6}$$

2.2 AES Decryption

In the AES decryption rounds, four transformations, i.e., *InvShiftRows*, *InvSubBytes*, *AddRoundKey*, and *InvMixColumns*, are utilized. Considering $\boldsymbol{S}'$ as the input state of a decryption round, in the first transformation, *InvShiftRows*, similar to *ShiftRows* in encryption, the first row of the input state remains unchanged. However, the other rows entries are cyclically shifted to the right as follows:

$$ISR(\boldsymbol{S}') = \begin{pmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,3} & s'_{1,0} & s'_{1,1} & s'_{1,2} \\ s'_{2,2} & s'_{2,3} & s'_{2,0} & s'_{2,1} \\ s'_{3,1} & s'_{3,2} & s'_{3,3} & s'_{3,0} \end{pmatrix}. \tag{7}$$

The next transformation in each round is *InvSubBytes* implemented by 16 inverse S-boxes. In the inverse S-box, the inverse affine transformation precedes the multiplicative inversion in $GF(2^8)$ to generate $s_{r,c}^{-1} = \boldsymbol{\Gamma}^{-1} s'_{r,c} + \boldsymbol{\Gamma}^{-1}\gamma$, where Γ and γ are presented in (2). The 8-bit outputs of 16 inverse S-boxes are used to obtain the output state of the InvSubBytes transformation as $\boldsymbol{S} = [s_{r,c}]_{r,c=0}^{3}$.

The next transformation is *AddRoundKey* in which the input state is added with the key of the round. Then, the output state of AddRoundKey is obtained as $\boldsymbol{S}'' = [s''_{r,c}]_{r,c=0}^{3} = \boldsymbol{S} + \boldsymbol{K}$. Finally, the last transformation, *InvMixColumns*, is equivalent to multiplying the input state with a constant matrix with hexadecimal entries to obtain the output state of the round as

$$\boldsymbol{O} = [o_{r,c}]_{r,c=0}^{3} = \begin{pmatrix} \{0e\}_h & \{0b\}_h & \{0d\}_h & \{09\}_h \\ \{09\}_h & \{0e\}_h & \{0b\}_h & \{0d\}_h \\ \{0d\}_h & \{09\}_h & \{0e\}_h & \{0b\}_h \\ \{0b\}_h & \{0d\}_h & \{09\}_h & \{0e\}_h \end{pmatrix} \boldsymbol{S}''. \tag{8}$$

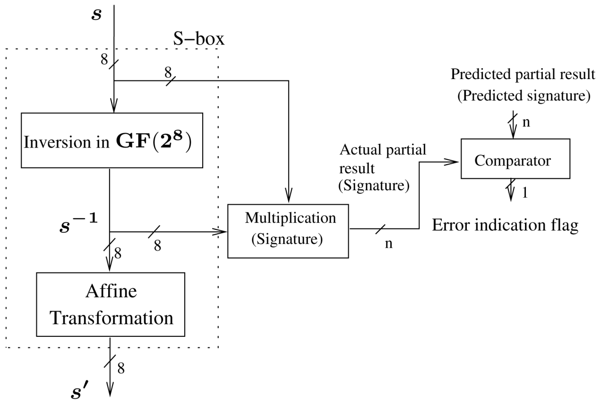
Among the four transformations in the encryption and the decryption of the AES, only the S-boxes and the inverse S-boxes are nonlinear operations. Furthermore, not only are the S-boxes used in the AES transformations, but they are also used in the key expander unit generating the keys used in the AES rounds. Therefore, the fault detection schemes of these operations affect the fault detection implementations of the entire AES.

3 A New Fault Detection Scheme for the S-Box and the Inverse S-Box

In this section, first, we present a systematic method for the fault detection of the multiplicative inversion of the S-box and the inverse S-box. Then, the new scheme for the entire S-box and the inverse S-box is presented.

3.1 The Systematic Scheme for the Multiplicative Inversion

The multiplication-based fault detection scheme [19] for the multiplicative inversion of the S-box is shown in Fig. 1. In this scheme, the 8-bit input of the multiplicative inversion is multiplied by the 8-bit output and the n-bit result,



s
S–box
8
8
Inversion in $\mathbf{GF(2^8)}$
s^{-1}
8
8
Affine Transformation
8
s'
Multiplication (Signature)
n
Actual partial result (Signature)
Predicted partial result (Predicted signature)
n
Comparator
1
Error indication flag

Fig. 1. The multiplication-based scheme for the fault detection of the multiplicative inversion [19].

$1 \leq n \leq 8$, of the multiplication is compared with the n-bit actual result, i.e., $1 \in GF(2^8)$ if $s \neq 0$ and $0 \in GF(2^8)$ if $s = 0$. Because the multiplicative inversion is also used in the inverse S-box, the same scheme can be used for the inverse S-box.

In what follows, we present a systematic method for the fault detection scheme for the multiplicative inversion by deriving the matrix-based formulations for the multiplicative inversion in the S-box/inverse S-box.

We use the following theorem from [22] to obtain the multiplication of field elements $A = \sum_{i=0}^{m-1} a_i\alpha^i$ and $B = \sum_{i=0}^{m-1} b_i\alpha^i$ in the finite field $GF(2^m)$ constructed by the irreducible polynomial of $P(x)$ with the primitive root of α.

Theorem 1 [22]. *Let $C = \sum_{i=0}^{m-1} c_i\alpha^i$ be the multiplication of A and $B \in GF(2^m)$. Then, the coordinates of C can be obtained from*

$$[c_0, c_1, \dots, c_{m-1}]^T = (\boldsymbol{L} + \boldsymbol{Q}^T\boldsymbol{U})\boldsymbol{b}, \tag{9}$$

where $\boldsymbol{b} = [b_0, b_1, \dots, b_{m-1}]^T$,

$$\boldsymbol{L} = \begin{pmatrix} a_0 & 0 & 0 & 0 & \dots & 0 \\ a_1 & a_0 & 0 & 0 & \dots & 0 \\ a_2 & a_1 & a_0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ a_{m-2} & a_{m-3} & \dots & a_1 & a_0 & 0 \\ a_{m-1} & a_{m-2} & \dots & a_2 & a_1 & a_0 \end{pmatrix}, \tag{10}$$

$$\boldsymbol{U} = \begin{pmatrix} 0 & a_{m-1} & a_{m-2} & \dots & a_2 & a_1 \\ 0 & 0 & a_{m-1} & \dots & a_3 & a_2 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & a_{m-1} & a_{m-2} \\ 0 & 0 & \dots & 0 & 0 & a_{m-1} \end{pmatrix}, \tag{11}$$

and the $m-1 \times m$ binary matrix $\boldsymbol{Q}$ is obtained as follows:

$$\begin{aligned} &[\alpha^m, \alpha^{m+1}, \dots, \alpha^{2m-2}]^T = \\ &\boldsymbol{Q}[1, \alpha, \alpha^2, \dots, \alpha^{m-1}]^T (\mathrm{mod} P(\alpha)). \end{aligned} \tag{12}$$

Let $s = s_7\alpha^7 + s_6\alpha^6 + s_5\alpha^5 + s_4\alpha^4 + s_3\alpha^3 + s_2\alpha^2 + s_1\alpha + s_0$ and $s^{-1} = s_7^{-1}\alpha^7 + s_6^{-1}\alpha^6 + s_5^{-1}\alpha^5 + s_4^{-1}\alpha^4 + s_3^{-1}\alpha^3 + s_2^{-1}\alpha^2 + s_1^{-1}\alpha + s_0^{-1}$ be the 8-bit input and output of the multiplicative inversion in the binary field $GF(2^8)$, respectively (see Fig. 1). Considering the fact that the result of the multiplication of the 8-bit input s, $s \neq 0$, and the output s^{-1} of the multiplicative inversion is the unity polynomial $1 \in GF(2^8)$, the following is derived from Theorem 1 for the relation between s and s^{-1}.

Corollary 1. *Let $\boldsymbol{s} = [s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7]^T$ and $\boldsymbol{s}^{-1} = [s_0^{-1}, s_1^{-1}, s_2^{-1}, s_3^{-1}, s_4^{-1}, s_5^{-1}, s_6^{-1}, s_7^{-1}]^T$ be the vectors corresponding to the input and output of the multiplicative inversion. Then, the matrix formulation of the multiplicative inversion of the S-box (respectively, the inverse S-box) is as follows:*

$$\boldsymbol{Z}\boldsymbol{s}^{-1} = \boldsymbol{u}, \tag{13}$$

where

$$\boldsymbol{Z} = \begin{pmatrix} s_0 & s_7 & s_6 & s_5 & s_4 & s_{7,3} & s_{7,6,2} & s_{6,5,1} \\ s_1 & s_{7,0} & s_{7,6} & s_{6,5} & s_{5,4} & s_{7,4,3} & s_{6,3,2} & s_{7,5,2,1} \\ s_2 & s_1 & s_{7,0} & s_{7,6} & s_{6,5} & s_{5,4} & s_{7,4,3} & s_{6,3,2} \\ s_3 & s_{7,2} & s_{6,1} & s_{7,5,0} & s_{7,6,4} & s_{7,6,5,3} & s_{7,6,5,4,2} & s_{7,6,5,4,3,1} \\ s_4 & s_{7,3} & s_{7,6,2} & s_{6,5,1} & s_{7,5,4,0} & s_{6,4,3} & s_{5,3,2} & s_{7,4,2,1} \\ s_5 & s_4 & s_{7,3} & s_{7,6,2} & s_{6,5,1} & s_{7,5,4,0} & s_{6,4,3} & s_{5,3,2} \\ s_6 & s_5 & s_4 & s_{7,3} & s_{7,6,2} & s_{6,5,1} & s_{7,5,4,0} & s_{6,4,3} \\ s_7 & s_6 & s_5 & s_4 & s_{7,3} & s_{7,6,2} & s_{6,5,1} & s_{7,5,4,0} \end{pmatrix}, \tag{14}$$

$\boldsymbol{u} = [u, 0, 0, 0, 0, 0, 0, 0]^T$, and u is obtained by logical OR operations of all inputs and outputs, i.e., $u = (s_0 \vee s_1 \vee \ldots s_7) \vee (s_0^{-1} \vee s_1^{-1} \vee \ldots s_7^{-1})$. Moreover, in (14), the modulo-2 additions (XOR operations) of the coordinates of s are shown with commas in indices, e.g., $s_{7,0} = s_7 + s_0$.

Proof. We prove (13) for two cases of $s \neq 0$ and $s = 0$ separately. Let the input s be a nonzero field element in $GF(2^8)$ generated by $P(x) = x^8 + x^4 + x^3 + x + 1$. Then, the multiplicative inversion should generate s^{-1}. Using (12) in Theorem 1 and considering the irreducible polynomial of $P(x)$, the 7×8 matrix $\boldsymbol{Q}$ can be obtained as

$$\boldsymbol{Q} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{15}$$

This matrix is obtained by using the representations of $\alpha^8, \alpha^9, \ldots, \alpha^{14}$ with respect to the polynomial basis for different rows of $\boldsymbol{Q}$. Considering $A = s \neq 0$ and $B = s^{-1}$ in Theorem 1, the matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ in (10) and (11) are functions of the 8-bit input vector $\boldsymbol{s}$ as

$$\boldsymbol{L} = \begin{pmatrix} s_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ s_1 & s_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ s_2 & s_1 & s_0 & 0 & 0 & 0 & 0 & 0 \\ s_3 & s_2 & s_1 & s_0 & 0 & 0 & 0 & 0 \\ s_4 & s_3 & s_2 & s_1 & s_0 & 0 & 0 & 0 \\ s_5 & s_4 & s_3 & s_2 & s_1 & s_0 & 0 & 0 \\ s_6 & s_5 & s_4 & s_3 & s_2 & s_1 & s_0 & 0 \\ s_7 & s_6 & s_5 & s_4 & s_3 & s_2 & s_1 & s_0 \end{pmatrix}, \tag{16}$$

$$\boldsymbol{U} = \begin{pmatrix} 0 & s_7 & s_6 & s_5 & s_4 & s_3 & s_2 & s_1 \\ 0 & 0 & s_7 & s_6 & s_5 & s_4 & s_3 & s_2 \\ 0 & 0 & 0 & s_7 & s_6 & s_5 & s_4 & s_3 \\ 0 & 0 & 0 & 0 & s_7 & s_6 & s_5 & s_4 \\ 0 & 0 & 0 & 0 & 0 & s_7 & s_6 & s_5 \\ 0 & 0 & 0 & 0 & 0 & 0 & s_7 & s_6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & s_7 \end{pmatrix}. \tag{17}$$

Substituting $\boldsymbol{Q}$, $\boldsymbol{L}$, and $\boldsymbol{U}$ from (15)-(17) into (9) and denoting $\boldsymbol{Z} = \boldsymbol{L} + \boldsymbol{Q}^T\boldsymbol{U}$, one can obtain the matrix $\boldsymbol{Z}$ presented in (14). Since $s \neq 0 = (0, 0, \ldots, 0) \in GF(2^8)$, $u = 1$ and the result of multiplication is

$$C = A.B \; mod \; P(x) = 1 \in GF(2^8),$$

i.e., $[c_0, c_1, \ldots, c_7]^T = [1, 0, \ldots, 0]^T$. Therefore, using (9), one can prove that (13) is valid for $s \neq 0$. Moreover, for $s = 0$, the output of the multiplicative inversion generates $0 = (0, 0, \ldots, 0)$. Thus, all entries of the matrix Z, and hence, all eight entries of the left-hand side vector of (13) are equal to zero. In such a case, the vector $\boldsymbol{u} = [0, 0, \ldots, 0]^T$ since the result of the OR operation among all s_is and s_i^{-1}s are zero, i.e., $u = 0$. Therefore, the proof is complete. □

The validity of (13) can be used to detect specific faults in the inversion block of Fig. 1. Let us consider (13) for three special cases. If both the input and the output are zero, i.e., $s = s^{-1} = 0 \in GF(2^8)$, the output is error-free. Then, both sides of (13) are zero, and thus, it holds which means that no fault is detected. On the other hand, the left-hand side of (13) is zero, while in the right-hand side, $u = 1$ in the following two cases: 1) the input is zero ($s = 0$) and the erroneous output is not zero, i.e., $s^{-1} \neq 0$ and 2) the input is not zero, i.e., $s \neq 0$, but the erroneous output is zero ($s^{-1} = 0$). Thus, in both cases, (13) does not hold which indicates that the errors in the output of the multiplicative inversion have been occurred.

One can figure out that implementing (13) needs 64 ANDs, 15 ORs, and 143 XOR gates. It is noted that using subexpression sharing, one can reduce the number of XOR gates to 84. If one implements the S-box using the composite field presented in [23], it requires 36 AND gates and 123 XOR gates for the original S-box implementation. Then, adding this fault detection scheme would require approximately 91 percent area overhead. This is derived assuming that an XOR gate is implemented by 10 transistors [24] and the silicon area of an AND is 0.6 that of an XOR gate. Furthermore, the upper bound delay of the multiplication can be derived as $T_M \leq T_A + 5T_X$, where T_A and T_X are the delays for an AND and an XOR gate, respectively, [22]. This is the delay overhead after the derivation of the output of the SubBytes transformation. As a result of this high overhead, this scheme may not be applied for the area-/delay-constrained applications.

As mentioned above, comparing the actual result of the multiplication of the input and the output of the multiplicative inversion with the predicted one is not area efficient. Therefore, considering our derivations of matrix Z in (14), the complexity of the fault detection scheme of the multiplicative inversion can be reduced by deriving the partial result of the multiplication of the input and the output based on the rows that have the lowest overhead. Therefore, one can use this low-complexity signature for the fault detection of the multiplicative inversion.

3.2 The Proposed Scheme for the S-Box and the Inverse S-Box

The scheme in [19] does not take the affine transformation into account and checks it separately with an additional overhead. Furthermore, if one implements SubBytes in the AES using LUTs, there is no access to the output of the multiplicative inversion. Therefore, the aforementioned scheme cannot be used. In what follows, we propose a new scheme which is independent of the way the S-box and the inverse S-box are implemented. First, we obtain the matrix-based S-box formulations as follows:

Theorem 2. *Let* $s = s_7\alpha^7 + s_6\alpha^6 + s_5\alpha^5 + s_4\alpha^4 + s_3\alpha^3 + s_2\alpha^2 + s_1\alpha + s_0$ *and* $s' = s'_7\alpha^7 + s'_6\alpha^6 + s'_5\alpha^5 + s'_4\alpha^4 + s'_3\alpha^3 + s'_2\alpha^2 + s'_1\alpha + s'_0$ *be the 8-bit input and output of the S-box. Then, one can obtain the relation between the input and output of the S-box as*

$$\boldsymbol{M}\boldsymbol{s}' + \boldsymbol{m} = \boldsymbol{u}', \tag{18}$$

where $\boldsymbol{u}' = [u', 0, 0, 0, 0, 0, 0, 0]^T$, $u' = (s_0 \vee s_1 \vee \dots s_7) \vee (\overline{s'_0} \vee \overline{s'_1} \vee s'_2 \vee s'_3 \vee s'_4 \vee \overline{s'_5} \vee \overline{s'_6} \vee s'_7)$, $\boldsymbol{s}' = [s'_0, s'_1, s'_2, s'_3, s'_4, s'_5, s'_6, s'_7]^T$, *and* $\boldsymbol{m} = [s_{6,0},\ s_{7,6,1},\ s_{7,2,0},\ s_{6,3,1},\ s_{7,6,4,2}, s_{7,5,3},\ s_{6,4},\ s_{7,5}]^T$. *Furthermore, the* 8×8 *matrix* $\boldsymbol{M}$ *is denoted as follows:*

$$\boldsymbol{M} = \begin{pmatrix} s_{6,5,2} & s_{5,4,1} & s_{7,5,3,0} & s_{6,4,2} & s_{7,5,3,1} & s_{7,6,5,2,0} & s_{7,6,5,4,1} & s_{7,6,3,0} \\ s_{7,5,3,2,0} & s_{6,4,2,1} & s_{7,6,5,4,3,1} & s_{7,6,5,4,3,2,0} & s_{7,6,5,4,3,2,1} & s_{5,3,2,1} & s_{4,2,1,0} & s_{6,4,3,1} \\ s_{6,4,3,1} & s_{7,5,3,2,0} & s_{7,6,5,4,2} & s_{7,6,5,4,3,1} & s_{7,6,5,4,3,2,0} & s_{6,4,3,2} & s_{5,3,2,1} & s_{7,5,4,2,0} \\ s_{7,6,4,0} & s_{6,5,3} & s_{6,0} & s_{7,5} & s_{6,4} & s_{6,4,3,2,0} & s_{7,5,3,2,1} & s_{7,5,1} \\ s_{7,6,2,1} & s_{7,6,5,1,0} & s_{5,3,1} & s_{4,2,0} & s_{3,1} & s_{6,4,3,2,1} & s_{7,5,3,2,1,0} & s_{7,3,2} \\ s_{7,3,2} & s_{7,6,2,1} & s_{6,4,2,0} & s_{5,3,1} & s_{4,2,0} & s_{7,5,4,3,2} & s_{6,4,3,2,1} & s_{4,3,0} \\ s_{4,3,0} & s_{7,3,2} & s_{7,5,3,1} & s_{6,4,2,0} & s_{5,3,1} & s_{6,5,4,3,0} & s_{7,5,4,3,2} & s_{5,4,1} \\ s_{5,4,1} & s_{4,3,0} & s_{6,4,2} & s_{7,5,3,1} & s_{6,4,2,0} & s_{7,6,5,4,1} & s_{6,5,4,3,0} & s_{6,5,2} \end{pmatrix}. \tag{19}$$

Proof. We prove (18) for two cases of $s \neq 0$ and $s = 0$ separately. Let the 8-bit input s be a nonzero field element in $GF(2^8)$. Considering (2), one can obtain

$$\boldsymbol{s}^{-1} = \boldsymbol{\Gamma}^{-1}\boldsymbol{s}' + \boldsymbol{\Gamma}^{-1}\boldsymbol{\gamma}$$
$$= \begin{pmatrix} 0&0&1&0&0&1&0&1 \\ 1&0&0&1&0&0&1&0 \\ 0&1&0&0&1&0&0&1 \\ 1&0&1&0&0&1&0&0 \\ 0&1&0&1&0&0&1&0 \\ 0&0&1&0&1&0&0&1 \\ 1&0&0&1&0&1&0&0 \\ 0&1&0&0&1&0&1&0 \end{pmatrix} \boldsymbol{s}' + \begin{pmatrix} 1\\0\\1\\0\\0\\0\\0\\0 \end{pmatrix}. \tag{20}$$

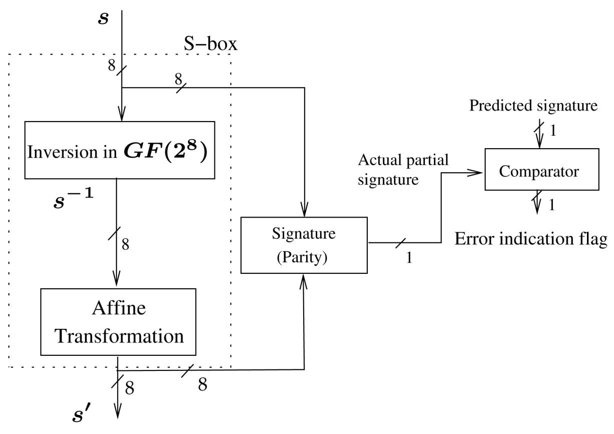


Fig. 2. The proposed structure-independent fault detection scheme of the S-box.

By substituting $\boldsymbol{s}^{-1}$ from (20) into (13), one reaches $\boldsymbol{Z}\boldsymbol{\Gamma}^{-1}\boldsymbol{s}' + \boldsymbol{Z}\boldsymbol{\Gamma}^{-1}\boldsymbol{\gamma}$ which is the same as the left-hand side of (18). Now, let us denote $\boldsymbol{Z}\boldsymbol{\Gamma}^{-1} = \boldsymbol{M}$ and $\boldsymbol{Z}\boldsymbol{\Gamma}^{-1}\boldsymbol{\gamma} = \boldsymbol{m}$. Then, the left-hand side of (18) is obtained. Since $s \neq 0 = (0, 0, \dots, 0) \in GF(2^8)$, $u' = 1$. Moreover, according to the proof of Corollary 1, for $s \neq 0$, the left-hand side of (13) is $[1, 0, \dots, 0]^T$, i.e., the result of multiplication $C = A.B \mod P(x) = 1 \in GF(2^8)$. This implies that the left-hand side of (13) be $\boldsymbol{Z}\boldsymbol{s}^{-1} = [1, 0, \dots, 0]^T = \boldsymbol{u}'$. Furthermore, because we have $\boldsymbol{Z}\boldsymbol{s}^{-1} = \boldsymbol{M}\boldsymbol{s}' + \boldsymbol{m}$, one can prove that (18) is valid for $s \neq 0$. Moreover, according to (2), for the input $s = 0 = (0, 0, \dots, 0) \in GF(2^8)$, we have the output as $\boldsymbol{s}' = [s'_0, s'_1, \dots, s'_7]^T = [1, 1, 0, 0, 0, 1, 1, 0]^T$ which corresponds to the field element $s' = \{63\}_h = (0, 1, 1, 0, 0, 0, 1, 1) \in GF(2^8)$. Therefore, as seen in Theorem 2, $\boldsymbol{u}' = [0, 0, \dots, 0]^T$ since we have $u' = (s_0 \vee s_1 \vee \dots s_7) \vee (\overline{s'_0} \vee \overline{s'_1} \vee s'_2 \vee s'_3 \vee s'_4 \vee \overline{s'_5} \vee \overline{s'_6} \vee s'_7) = 0$. In addition, for $s = 0$, all the entries of the matrix $\boldsymbol{M}$ and the vector $\boldsymbol{m}$ in the left-hand side of (18) are equal to zero. This results in the vector $[0, 0, \dots, 0]^T = \boldsymbol{u}'$ for the left-hand side of (18). Therefore, the proof is complete. □

Let us consider (18) for the input $s = 0 = (0, 0, \dots, 0) \in GF(2^8)$. For this input, the correct output is $s' = \{63\}_h = (0,1, 1, 0, 0, 0, 1, 1) \in GF(2^8)$ (see (2)). If the erroneous output is not $s' = \{63\}_h = (0, 1, 1, 0, 0, 0, 1, 1) \in GF(2^8)$, in the right-hand side of (18), we have $u' = 1$, whereas the left-hand side is zero. Therefore, the erroneous output is detected.

Proposition 1. *Using subexpression sharing, the implementation of the left-hand side of (18) needs 64 AND gates and 111 XOR gates. Furthermore, the upper bound delay of the relation in the left-hand side of (18) is* $T_A + 6T_X$, *where* T_A *and* T_X *are the delays for an AND and an XOR gate, respectively.*

Although checking the formulation of (18) detects all errors in the output of the S-box, its implementation is very costly (see Proposition 1). To reduce the overhead of the fault detection scheme, as shown in Fig. 2, we have obtained the single-bit parity for the formulation of (18). As shown in this figure, this is obtained in order to compare only 1 bit for an 8-bit data to detect any combination of odd number of erroneous bits at the result of the left-hand side of (18).

Thus, one can check the parity of two sides of (18) to obtain 1-bit equation for checking the S-box as follows:

Theorem 3. *Let $s = s_7\alpha^7 + s_6\alpha^6 + s_5\alpha^5 + s_4\alpha^4 + s_3\alpha^3 + s_2\alpha^2 + s_1\alpha + s_0 \in GF(2^8)$ and $s' = s'_7\alpha^7 + s'_6\alpha^6 + s'_5\alpha^5 + s'_4\alpha^4 + s'_3\alpha^3 + s'_2\alpha^2 + s'_1\alpha + s'_0 \in GF(2^8)$ be the 8-bit input and output of the S-box. Then, the following equation holds for all the possible patterns of s and s':*

$$P_{(\boldsymbol{M}s'+\boldsymbol{m})} = s_0(s'_b + s'_c) + s_1 s'_b + s_2 s'_d + s_3 s'_4 + s_4(s'_c + s'_3) + s_5 s'_a + s_6\left(s'_d + \overline{s'_6}\right) + s_7\left(\overline{s'_5 + s'_4}\right) = u', \tag{21}$$

where $s'_a = s'_0 + s'_2 + s'_3 + s'_5$, $s'_b = s'_a + s'_7$, $s'_c = s'_1 + s'_4 + s'_6$, and $s'_d = s'_2 + s'_7$.

Proof. After obtaining the parity of two sides of (18), we have

$$P_{(\boldsymbol{M}s'+\boldsymbol{m})} = P_{\boldsymbol{u}'} = u', \tag{22}$$

where $\boldsymbol{M}$, $\boldsymbol{m}$, and $\boldsymbol{u}'$ are presented in Theorem 2. Considering the fact that parity is a linear operation, one can obtain the left-hand side of (22) as $P_{(\boldsymbol{M}s'+\boldsymbol{m})} = P_{\boldsymbol{M}s'} + P_{\boldsymbol{m}}$. Then, using $\boldsymbol{M}$ and $\boldsymbol{m}$ defined in Theorem 2, one can obtain

$$P_{\boldsymbol{M}s'} = s'_0 s_a + s'_1 s_b + s'_2 s_c + s'_3(s_a + s_4) + s'_4(s_b + s_3 + s_7) + s'_5(s_a + s_7) + s'_6(s_b + s_6) + s'_7(s_5 + s_c)$$

and $P_{\boldsymbol{m}} = s_6 + s_7$, where $s_a = s_0 + s_1 + s_5$, $s_b = s_0 + s_4$, and $s_c = s_a + s_2 + s_6$. After rearranging, one reaches (21) and the proof is complete. □

To implement (21), 18 XOR gates, eight AND gates, and two NOT gates are needed. Also, the delay overhead associated with this implementation is the delay of six XORs and one AND after the completion of the S-box. It is noted that this delay can be overlapped by other AES round transformations, and hence, it will not reduce the speed of the entire fault detection AES implementation. More details on this will be presented in Section 4 of this paper. The parity obtained by the parity circuit is then compared with u' (see Theorems 2 and 3) to obtain the error indication flag of each S-box, i.e., $e_{r,c}$, $0 \leq r, c \leq 3$. It is noted that using an OR tree for the error indication flags of 16 S-boxes, the final error indication flag of the entire SubBytes transformation is obtained. The final error indication flag of the SubBytes transformation signals the errors if at least one of the error indication flags of 16 S-boxes detects errors.

Now, we want to present the fault detection scheme for the inverse S-box in the AES decryption. The inverse S-box of the decryption consists of the inverse affine transformation (the inverse of the affine transformation in (2)) followed by the multiplicative inversion. In other words, one can obtain the inverse S-box by removing the affine transformation and adding the inverse affine one. This uses the input of s' and the output of s^{-1} with the following multiplicative inversion having the input of s^{-1} and the output of s. Therefore, Theorems 2 and 3 are also valid for the inverse S-box, and hence, we can conclude the following for the inverse S-box:

Corollary 2. *For the fault detection of the inverse S-box, one can use (21) by changing the place of the input and output, i.e., swapping the coordinates of s with s'.*

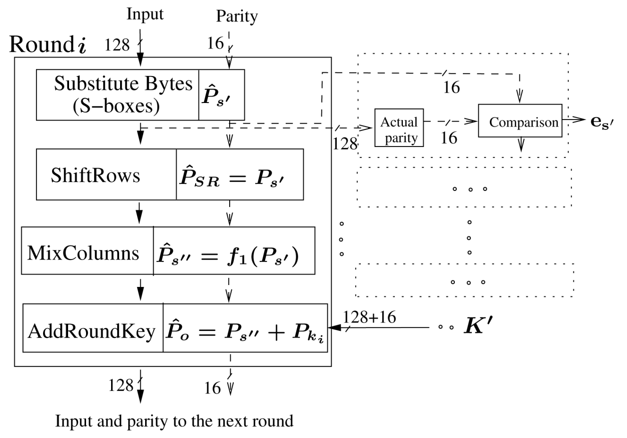


Fig. 3. Recursive parity-based fault detection structure of the ith round for encryption in AES [8].

4 Proposed Fault Detection Schemes for the AES

The basic approaches using parity bits for fault detection are presented in [8] and [9]. For one round of the AES encryption, the parity-based scheme presented in [8] is shown in Fig. 3. Similar fault detection scheme is used for other rounds. As seen in this figure, the output parity bits of each transformation in every round are predicted from the inputs (obtained using the blocks denoted by $\hat{P}$ notations). Then, the comparisons between the predicted and the actual parities (obtained using the actual parity block) can be scheduled so that the desired error coverage is obtained [8]. It is noted that in Fig. 3, $\boldsymbol{K'}$ consists of the 128-bit round i key and the 16-bit key parity, i.e., P_{k_i}.

The parity-based scheme proposed in [8] is one of the first fault detection schemes and has received attention in the literature. Although the approach in [8] is a good scheme in terms of the fault detection capability, it has two drawbacks. First, this approach is based on using the expanded S-boxes and inverse S-boxes for parity predictions, i.e., two blocks of 256×9 memory cells. Not only does this restrict the AES encryption and decryption implementations to LUT-based S-boxes and inverse S-boxes, but it has also the area overhead of greater than 100 percent for either the S-box or the inverse S-box. To counteract this drawback, one may use the proposed fault detection scheme for the S-box or the inverse S-box. As an example, for the AES encryption, one may use (21) for the S-boxes. This results in obtaining the output parity of each S-box concurrently without having an extra circuit for deriving it, i.e., $P_{s'} = \sum_{i=0}^{7} s'_i = s'_b + s'_c$ in (21). This simplifies the fault detection circuit of the AES when the output parities of the S-boxes are utilized for the fault detection of other transformations in the AES rounds in [8] (see Fig. 3). More specifically, if one uses the scheme presented in [8] for the fault detection of the MixColumns transformation, the predicted parities of this transformation become functions of the output parities of the ShiftRows (SubBytes) transformation (f_1 in Fig. 3). Using the proposed scheme for the S-box in this paper, one can easily utilize the output parities of the S-boxes to predict the parities of the MixColumns transformation.

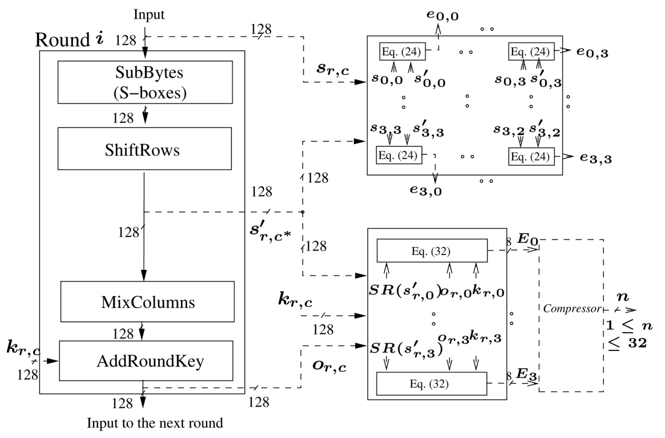


Fig. 4. The proposed fault detection scheme for the ith round of the AES encryption.

The second drawback of the approach in [8] is the relatively high area complexity of the parity predictions of MixColumns in the AES encryption. For the AES decryption, the area complexity of the predicted parities of InvMixColumns is even more [10]. The implementation results presented in Section 6 show the high area overhead of this scheme. Considering the fact that a low-cost fault detection scheme for the AES encryption and decryption is preferred, in this section, we propose signature-based low-complexity fault detection schemes for the transformations in the AES encryption and decryption. We consider AES-128 (which is denoted as AES in the remaining of this paper) for the sake of brevity. It is noted that the proposed schemes can also be applied to AES-192 and AES-256. The proposed schemes for the AES transformations are based on deriving the low-cost output signatures of the transformations in the AES rounds and comparing them with their actual signatures for reaching the error indication flags.

4.1 AES Encryption

We present the new fault detection structure for the AES encryption in the following. A typical AES encryption round (except for the last round) consists of four transformations, and the fault detection schemes are shown in Fig. 4 and presented in details below.

4.1.1 SubBytes and ShiftRows

In the AES encryption, the SubBytes transformation consists of 16 S-boxes (see (3)). Let $e_{r,c}$, $0 \leq r, c \leq 3$, be the error indication flag for the S-box with the input and the output of $s_{r,c}$ and $s'_{r,c}$, respectively. The output state of such flags can be rewritten as 16 formulations as follows:

$$e_{r,c} = P_{(\boldsymbol{M}_{r,c}\boldsymbol{s'}_{r,c}+\boldsymbol{m}_{r,c})} + u'_{r,c}, \quad 0 \leq r, c \leq 3, \tag{23}$$

where $u'_{r,c}$ is defined in Theorem 2 and for a typical S-box, $P_{(\boldsymbol{M}_{r,c}\boldsymbol{s'}_{r,c}+\boldsymbol{m}_{r,c})}$ is presented in (21).

The 128-bit output of the SubBytes transformation acts as the input to ShiftRows. As seen in (4), the output state of ShiftRows is obtained by shifting the state entries in (3). Therefore, by considering the corresponding output of ShiftRows in (4), one can check two transformations of SubBytes and ShiftRows together using 16 error indication flags. According to (4) and considering (23), for row r and column c, the output state of the flags can be rewritten as 16 formulations as follows:

$$e_{r,c} = P_{(\boldsymbol{M}_{r,c^*}\boldsymbol{s'}_{r,c^*}+\boldsymbol{m}_{r,c^*})} + u'_{r,c^*}, \quad 0 \leq r, c \leq 3, \tag{24}$$

where $c^* = (r + c) \bmod 4$.

According to (24), 16 error indication flags for the SubBytes and ShiftRows transformations, i.e., one error indication flag for each byte, are obtained. This is shown in Fig. 4. As seen in this figure, (24), i.e., instances of the hardware implementation of (21), is utilized for obtaining 16 error indication flags.

4.1.2 MixColumns and AddRoundKey

The third and the fourth transformations in a typical AES encryption round are MixColumns and AddRoundKey. It is noted that MixColumns is constructed using (5). Furthermore, according to (6), AddRoundKey is the modulo-2 addition of the input state with the roundkey. In what follows, we present a key formulation that is used for deriving a low-complexity fault detection scheme for MixColumns and AddRoundKey combined.

Theorem 4. *Let $SR(\boldsymbol{S'}) = [s'_{r,c^*}]_{r,c=0}^{3}$ and $\boldsymbol{K} = [k_{r,c}]_{r,c=0}^{3}$ be the input and the roundkey input of MixColumns and AddRoundKey in round i, respectively. Let the output of AddRoundKey be $\boldsymbol{O} = [o_{r,c}]_{r,c=0}^{3}$ (see (6)). Then, the following holds:*

$$\sum_{r=0}^{3}(s'_{r,c^*} + k_{r,c} + o_{r,c}) = 0 \in GF(2^8), \quad 0 \leq c \leq 3, \tag{25}$$

where $c^* = (r + c) \bmod 4$, and each summation is over $GF(2^8)$ which consists of eight modulo-2 additions.

Proof. After adding the columns of S'' in (5), one reaches the following:

$$s''_{0,0} + s''_{1,0} + s''_{2,0} + s''_{3,0} = (\{2\}_{16} + \{1\}_{16} + \{1\}_{16} + \{3\}_{16})(s'_{0,0} + s'_{1,1} + s'_{2,2} + s'_{3,3}), \tag{26}$$

$$s''_{0,1} + s''_{1,1} + s''_{2,1} + s''_{3,1} = (\{2\}_{16} + \{1\}_{16} + \{1\}_{16} + \{3\}_{16})(s'_{0,1} + s'_{1,2} + s'_{2,3} + s'_{3,0}), \tag{27}$$

$$s''_{0,2} + s''_{1,2} + s''_{2,2} + s''_{3,2} = (\{2\}_{16} + \{1\}_{16} + \{1\}_{16} + \{3\}_{16})(s'_{0,2} + s'_{1,3} + s'_{2,0} + s'_{3,1}), \tag{28}$$

$$s''_{0,3} + s''_{1,3} + s''_{2,3} + s''_{3,3} = (\{2\}_{16} + \{1\}_{16} + \{1\}_{16} + \{3\}_{16})(s'_{0,3} + s'_{1,0} + s'_{2,1} + s'_{3,2}). \tag{29}$$

Considering the fact that $\{3\}_{16} = \{1\}_{16} + \{2\}_{16}$, we have $(\{2\}_{16} + \{1\}_{16} + \{1\}_{16} + \{3\}_{16}) = \{1\}_{16}$. Moreover, the right-hand sides of (26)-(29) are the additions of the columns of matrix $SR(\boldsymbol{S'})$ in (4). Therefore, the addition of the column elements of $\boldsymbol{S''}$ is equal to that of the corresponding column of $SR(\boldsymbol{S'})$, i.e., $\sum_{r=0}^{3} s''_{r,c} = \sum_{r=0}^{3} s'_{r,c^*}$, $0 \leq c \leq 3$. Furthermore, according to (6), we have

$$\sum_{r=0}^{3} o_{r,c} = \sum_{r=0}^{3} s''_{r,c} + \sum_{r=0}^{3} k_{r,c}, \quad 0 \leq c \leq 3. \tag{30}$$

Therefore, considering (30), we reach

$$\sum_{r=0}^{3} o_{r,c} = \sum_{r=0}^{3} s'_{r,c^*} + \sum_{r=0}^{3} k_{r,c}, \quad 0 \le c \le 3. \tag{31}$$

Considering (31), we have $\sum_{r=0}^{3}(s'_{r,c^*} + k_{r,c} + o_{r,c}) = (0, 0, \ldots, 0) \in GF(2^8)$ and the proof is complete. □

Now, let us introduce the four 8-bit error indication flags for four columns of the state as

$$E_c = \sum_{r=0}^{3}(s'_{r,c^*} + k_{r,c} + o_{r,c}), \quad 0 \le c \le 3. \tag{32}$$

One can use Theorem 4 to verify that for the error-free situation, all 32 bits of such flags in (32) are zero, i.e., $E_c = 0 = (0, 0, \ldots, 0) \in GF(2^8)$, $0 \le c \le 3$. These 32 error indication flags can be used for the MixColumns and AddRoundKey transformations combined, i.e., eight error indication flags for each column of the state matrix. This is shown in Fig. 4. It is noted that in Fig. 4, $[k_{r,c}]_{r,c=0}^{3}$ is the round i key. As seen in this figure, using (32), 32 error indication flags are obtained. It is noted that these error indication flags can be compressed so that n, $1 \le n \le 32$, error indication flags for these two transformations are achieved. This can be performed by ORing different combinations of the 32 error indication flags obtained in (32) as denoted by the compressor block in Fig. 4. This gives us the freedom in the number of the error indication flags used in the fault detection scheme of the MixColumns and AddRoundKey transformations. It is interesting to note that although up to 32 flags can be used, our simulations show that using 16 error indication flags (the same number as the flags derived for SubBytes and ShiftRows), greater than 99 percent of the errors are covered.

The last round of the AES encryption (round 10 in AES-128 encryption) consists of three transformations, i.e., SubBytes, ShiftRows, and AddRoundKey. In other words, compared to the other encryption rounds, the MixColumns transformation has been removed. We present the following for the fault detection of this round.

Remark 1. Similar to the fault detection scheme for the other rounds of the AES encryption, one can use (24) for the last encryption round to derive 16 error indication flags for SubBytes and ShiftRows combined. Furthermore, one can use (31) for the relation of the inputs and the output of AddRoundKey (see also Fig. 4 by removing MixColumns). Therefore, (32) can also be used for the last round. Consequently, by removing the MixColumns transformation, one can also utilize the fault detection scheme in Fig. 4 for the last encryption round of the AES.

4.1.3 Further Improvements

The proposed fault detection scheme for a typical round of the AES encryption can be modified so that the complexity of the scheme is reduced. This improvement is based on the fact that using subexpression sharing, one can reduce the number of logic gates utilized in obtaining two sets of the error indication flags, as shown in Fig. 4. Specifically, in this paper, we propose a fault detection scheme for the MixColumns transformation which has 25 percent less area overhead than the scheme presented in [8] and [10].

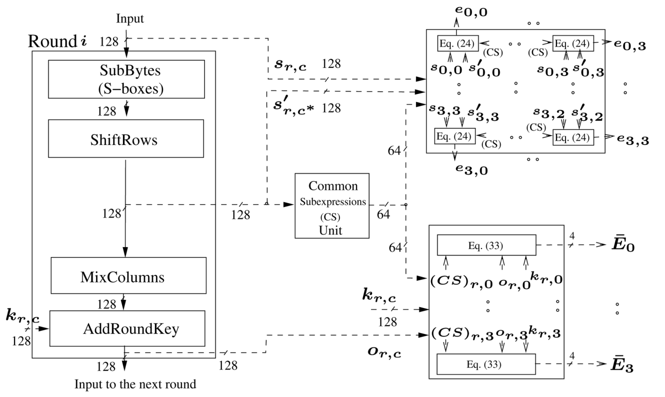


Fig. 5. The proposed low-complexity fault detection scheme for the ith round of the AES encryption utilizing subexpression sharing.

As shown in Fig. 4, the error indication flags of SubBytes and ShiftRows are obtained utilizing the output state of ShiftRows, i.e., $SR(S')$ in (4). Furthermore, as shown in this figure, this state is also used in obtaining the error indication flags of MixColumns and AddRoundKey. This leads us to perform subexpression sharing in deriving these two sets of error indication flags to have low-complexity fault detection scheme of the AES encryption. We use (32) to derive 16 low-complexity signatures for the MixColumns and AddRoundKey transformations, i.e., four signatures for each column of the state matrix. This is performed by modulo-2 addition of two sets of four coordinates of (32) for each column, i.e., $E_c = (e_{c,7}, e_{c,6}, \ldots, e_{c,0}) \in GF(2^8)$, $0 \le c \le 3$. Let $\hat{E}_c = (e_{c,4}, e_{c,2}, e_{c,1}, e_{c,0})$ and $\check{E}_c = (e_{c,5}, e_{c,7}, e_{c,6}, e_{c,3})$. Then, the four error indication flags for column c of the state are

$$\overline{E}_c = \hat{E}_c + \check{E}_c, \quad 0 \le c \le 3. \tag{33}$$

One can utilize four sets of modulo-2 additions of the output bits of each S-box precomputed in (21), i.e., $s'_4 + s'_5$, $s'_2 + s'_7$, $s'_1 + s'_6$, and $s'_0 + s'_3$, to obtain the low-complexity error indication flags in (33). This is shown in Fig. 5. As seen in this figure, the *Common Subexpressions* (CSs) unit has been utilized to obtain 64 common subexpressions, i.e., 4 for each of the 16 S-boxes in the SubBytes transformation. As depicted in Fig. 5, these outputs are then used in obtaining the two sets of 16 error indication flags for SubBytes and ShiftRows combined, i.e., $e_{r,c}$, $0 \le r, c \le 3$, and for MixColumns and AddRoundKey combined, i.e., $\overline{E}_c$, $0 \le c \le 3$, respectively. In Fig. 5, realizing (24) is less complex than the one in Fig. 4. This is because (24) utilizes the hardware implementation of (21) which is less complex when the common subexpressions are used. It is noted that if any of the two derived sets of error indication flags are one, the error is detected. Whereas if all of them are zero, no error has been detected although the output can be erroneous or correct.

One can compare the complexity of the proposed fault detection scheme for MixColumns with that of [8] and [10]. For comparison, we consider the error indication flags of this transformation separately, i.e., without considering AddRoundKey. In the fault detection scheme of MixColumns, we only need three XOR gates for each signature, i.e., modulo-2 adding of the four common subexpressions presented above, e.g., $s'_4 + s'_5$, in four rows. Therefore, we have the following remark:

Remark 2. For having 16 signatures for the MixColumns transformation, 48 XOR gates are needed. Comparing this with the parity-based scheme presented in [8] and [10] which needs 64 XOR gates for the predicted parities, this is a 25 percent area overhead reduction. Moreover, there are two XORs in the critical path delay of the proposed scheme for MixColumns compared to three XORs for the scheme in [8] and [10] which is a 33 percent reduction in the critical path delay.

4.2 AES Decryption

We present the fault detection scheme for the AES decryption in what follows. It is noted that the AES decryption rounds (except for the last round) consist of four transformations, i.e., InvShiftRows, InvSubBytes, AddRoundKey, and InvMixColumns. The fault detection schemes of these transformations are presented in details in the following.

4.2.1 InvShiftRows and InvSubBytes

As seen in (7), in the AES decryption, the 128-bit input to InvShiftRows, i.e., the state matrix S' entries, is cyclically shifted to the right with the first row remaining unchanged. Therefore, this transformation is just a rewiring in hardware.

The output state of the InvShiftRows transformation, i.e., $ISR(S')$ in (7), acts as the input to InvSubBytes. The InvSubBytes transformation in the AES decryption consists of 16 inverse S-boxes. One can use Corollary 2 for the fault detection scheme of the inverse S-boxes. Then, the fault detection scheme for InvShiftRows and InvSubBytes combined can be derived so that we are able to check these two transformations together. Let $e_{r,c}$, $0 \leq r, c \leq 3$, be the error indication flag of each byte of these two transformations combined with the input and the output of $s'_{r,c}$ and $s_{r,c}$, respectively. Then, according to (18), the output state of such flags can be rewritten as 16 formulations as follows:

$$e_{r,c} = P_{(\boldsymbol{M}_{r,c}\boldsymbol{s'}_{r,c^{**}}+\boldsymbol{m}_{r,c})} + u'_{r,c}, \quad 0 \leq r, c \leq 3, \tag{34}$$

where $c^{**} = |r - c|$.

According to (34), 16 error indication flags for the InvShiftRows and InvSubBytes transformations, i.e., one error indication flag for each byte, are obtained. This is shown in Fig. 6. As seen in this figure, (34), i.e., instances of the hardware implementation of (21), is utilized for obtaining these 16 error indication flags.

4.2.2 AddRoundKey and InvMixColumns

As shown in Fig. 6, the third and the forth transformations in a typical AES decryption round are AddRoundKey and InvMixColumns. In the AddRoundKey transformation, the input state, i.e., S, is added with the roundkey input state, i.e., K. Furthermore, the InvMixColumns transformation is equivalent to multiplying the input state with the constant matrix in (8). In what follows, we present a key formulation used for deriving a low-complexity fault detection scheme for these two transformations combined.

Theorem 5. *Let* $K = [k_{r,c}]_{r,c=0}^{3}$ *and* $S = [s_{r,c}]_{r,c=0}^{3}$ *be the roundkey input and the input of AddRoundKey in round* i*, respectively. Let the output of InvMixColumns be* $O = [o_{r,c}]_{r,c=0}^{3}$ *(see (8)). Then, the following holds:*

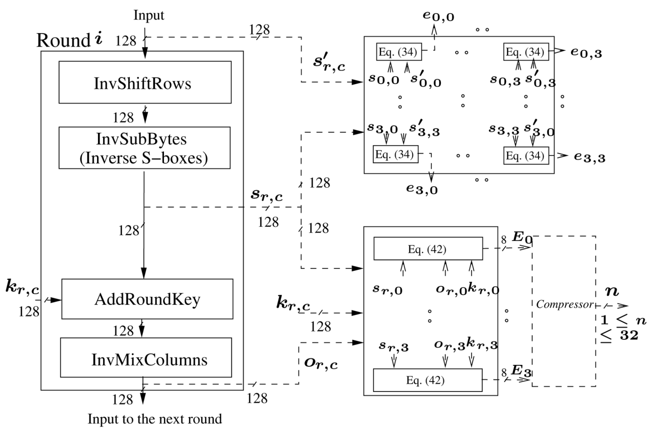


Fig. 6. The proposed fault detection scheme for the ith round of the AES decryption.

$$\sum_{r=0}^{3}(s_{r,c} + k_{r,c} + o_{r,c}) = 0 \in GF(2^8), \quad 0 \leq c \leq 3, \tag{35}$$

where each summation is over $GF(2^8)$ *which consists of eight modulo-2 additions.*

Proof. After adding the columns of O, according to (8), one reaches

$$o_{0,0} + o_{1,0} + o_{2,0} + o_{3,0} = (\{e\}_{16} + \{9\}_{16} + \{d\}_{16} + \{b\}_{16})(s''_{0,0} + s''_{1,0} + s''_{2,0} + s''_{3,0}), \tag{36}$$

$$o_{0,1} + o_{1,1} + o_{2,1} + o_{3,1} = (\{e\}_{16} + \{9\}_{16} + \{d\}_{16} + \{b\}_{16})(s''_{0,1} + s''_{1,1} + s''_{2,1} + s''_{3,1}), \tag{37}$$

$$o_{0,2} + o_{1,2} + o_{2,2} + o_{3,2} = (\{e\}_{16} + \{9\}_{16} + \{d\}_{16} + \{b\}_{16})(s''_{0,2} + s''_{1,2} + s''_{2,2} + s''_{3,2}), \tag{38}$$

$$o_{0,3} + o_{1,3} + o_{2,3} + o_{3,3} = (\{e\}_{16} + \{9\}_{16} + \{d\}_{16} + \{b\}_{16})(s''_{0,3} + s''_{1,3} + s''_{2,3} + s''_{3,3}). \tag{39}$$

We have $\{e\}_{16} + \{9\}_{16} + \{d\}_{16} + \{b\}_{16} = \{1\}_{16}$. Noting that the right-hand sides of (36)-(39) are the additions of the columns of the output state of InvMixColumns, the addition of the column elements of S'' is equal to that of the corresponding column of O, i.e., $\sum_{r=0}^{3} s''_{r,c} = \sum_{r=0}^{3} o_{r,c}$, $0 \leq c \leq 3$. Furthermore, for the AddRoundKey transformation, we have

$$\sum_{r=0}^{3} s''_{r,c} = \sum_{r=0}^{3} s_{r,c} + \sum_{r=0}^{3} k_{r,c}, \quad 0 \leq c \leq 3. \tag{40}$$

Therefore, according to (40), we reach

$$\sum_{r=0}^{3} o_{r,c} = \sum_{r=0}^{3} s_{r,c} + \sum_{r=0}^{3} k_{r,c}, \quad 0 \leq c \leq 3. \tag{41}$$

Considering (41), one can obtain $\sum_{r=0}^{3}(s_{r,c} + k_{r,c} + o_{r,c}) = (0, 0, \ldots, 0) \in GF(2^8)$ and the proof is complete. □

Similar to the AES encryption, for the AES decryption, we introduce the four 8-bit error indication flags for four columns of the state as

$$E_c = \sum_{r=0}^{3}(s_{r,c} + k_{r,c} + o_{r,c}), \quad 0 \le c \le 3. \tag{42}$$

These 32 error indication flags for four columns of the state can be utilized for the fault detection of the AddRoundKey and InvMixColumns transformations combined. This is shown in Fig. 6. It is noted that like the AES encryption, these error indication flags can be compressed so that n, $1 \le n \le 32$, error indication flags for these two transformations are achieved. This gives us the freedom in the number of the error indication flags used in the fault detection scheme of the AddRoundKey and InvMixColumns transformations. It is interesting to note that our simulations for the AES decryption show that using 16 error indication flags, more than 99 percent of the errors are covered.

Similar to the AES encryption, in the last round of the AES decryption, three transformations are used, i.e., InvMixColumns is removed. We present the following for the fault detection of this round.

Remark 3. Similar to the fault detection scheme for the other rounds of the AES decryption, one can use (34) for the last decryption round to derive 16 error indication flags for InvShiftRows and InvSubBytes combined. Furthermore, one can use (41) for the relation of the inputs and the output of AddRoundKey (see also Fig. 6 by removing InvMixColumns). Therefore, (42) can also be used for the last round. Consequently, by removing the InvMixColumns transformation, one can also utilize the fault detection scheme in Fig. 6 for the last decryption round of the AES.

4.2.3 Further Improvements

Using subexpression sharing, the proposed fault detection scheme for a typical AES decryption round can be modified so that its hardware complexity is reduced. As shown in Fig. 6, the error indication flags of InvShiftRows and InvSubBytes are obtained utilizing the output state of InvSubBytes, i.e., S. As shown in Fig. 6, this output state is also used in obtaining the error indication flags of AddRoundKey and InvMixColumns. Therefore, similar to the fault detection scheme for the AES encryption, we can perform subexpression sharing to obtain these two sets of error indication flags to have low-complexity fault detection scheme of the AES decryption. First, we present the following for the inverse S-boxes by rearranging Corollary 2 so that we are able to present a low-complexity fault detection scheme for the AES decryption.

Corollary 3. *Let* $s' = s'_7\alpha^7 + s'_6\alpha^6 + s'_5\alpha^5 + s'_4\alpha^4 + s'_3\alpha^3 + s'_2\alpha^2 + s'_1\alpha + s'_0 \in GF(2^8)$ *and* $s = s_7\alpha^7 + s_6\alpha^6 + s_5\alpha^5 + s_4\alpha^4 + s_3\alpha^3 + s_2\alpha^2 + s_1\alpha + s_0 \in GF(2^8)$ *be the 8-bit input and output of the inverse S-box. Then, the following equation holds for all the possible patterns of s and s':*

$$\begin{aligned} P_{(Ms'+m)} &= s'_0 s_a + s'_1 s_b + s'_2 s_c + s'_3(s_a + s_4) \\ &+ s'_4(s_b + s_3 + s_7) + s'_5(s_a + s_7) \\ &+ s'_6(s_b + s_6) + s'_7(s_5 + s_c) + s_6 + s_7 = u', \end{aligned} \tag{43}$$

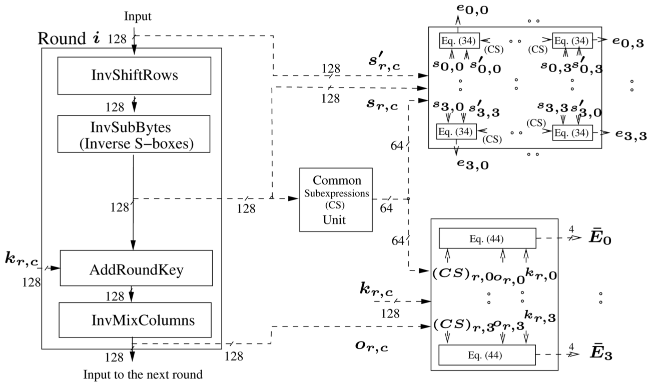


Fig. 7. The proposed low-complexity fault detection scheme for the ith round of the AES decryption utilizing subexpression sharing.

where

$$\begin{aligned} &s_a = s_0 + s_1 + s_5, s_b = s_0 + s_4, \\ &s_c = s_a + s_2 + s_6, \ \ and \\ &u' = (s_0 \vee s_1 \vee \dots s_7) \\ &\quad \vee \left(\overline{s'_0} \vee \overline{s'_1} \vee s'_2 \vee s'_3 \vee s'_4 \vee \overline{s'_5} \vee \overline{s'_6} \vee s'_7\right). \end{aligned}$$

Proof. According to Theorem 3 and Corollary 2, one can rewrite (21) and swap the input and the output to derive (43). Therefore, the proof is complete. □

To implement the signature presented in the left-hand side of (43), 20 XOR gates and eight AND gates are needed. Then, it is compared with u' to obtain the error indication flag of each inverse S-box.

Using Corollary 3 and Theorem 5, we derive 16 low-complexity signatures for the AddRoundKey and InvMixColumns transformations, i.e., four signatures for each column of the state matrix. This is performed by modulo-2 addition of two sets of four coordinates of (42) for each column, i.e., $E_c = (e_{c,7}, e_{c,6}, \dots, e_{c,0}) \in GF(2^8)$, $0 \le c \le 3$. For the AES decryption, let $\acute{E}_c = (e_{c,3}, e_{c,2}, e_{c,1}, e_{c,0})$ and $\grave{E}_c = (e_{c,7}, e_{c,6}, e_{c,5}, e_{c,4})$. Then, the four error indication flags for column c of the state are

$$\overline{E}_c = \acute{E}_c + \grave{E}_c, \quad 0 \le c \le 3. \tag{44}$$

One can utilize four sets of modulo-2 additions of the output bits of each inverse S-box precomputed in Corollary 3, i.e., $s_0 + s_4$, $s_1 + s_5$, $s_2 + s_6$, and $s_3 + s_7$, to obtain the low-complexity error indication flags in (44). This is shown in Fig. 7. As seen in this figure, similar to the AES encryption, the *CSs* unit has been utilized to obtain 64 common subexpressions. Then, these outputs are used in obtaining the two sets of 16 error indication flags for the AES decryption, respectively. It is noted that in Fig. 7, the hardware implementation of (43) is used in (34) which is less complex when the common subexpressions are used.

The proposed fault detection scheme for InvMixColumns requires 48 XOR gates with two XOR gates in the critical path. Compared to the scheme presented in [10] for the InvMixColumns transformation, the proposed scheme has less area and critical path delay. It is noted that the authors in [10] have not presented the equations for the parity-based fault detection scheme of InvMixColumns, mentioning that they have the same structure as those of MixColumns but they are more complicated. Therefore, at least a 25 percent area overhead reduction and a 33 percent reduction in the critical path delay are expected for the proposed scheme.

5 Error Simulations

We have considered both single and multiple stuck-at errors for the proposed scheme. These models cover both natural faults and fault attacks [25]. If exactly 1 bit error appears at the output of the AES encryption or decryption rounds, the presented parity-based fault detection scheme is able to detect it and the error coverage of the proposed scheme is about 100 percent. This is because in this case, one of the 8-bit four error indication flags in (33) or (44) alarms the error. However, due to the technological constraints, single stuck-at error may not be applicable for an attacker to flip exactly 1 bit to gain more information [25]. Thus, multiple bits will actually be flipped, and hence, multiple stuck-at errors are also considered in this paper.

For the multiple stuck-at error models, we rely on simulations for both burst and random errors. In the case of fault attacks, it is more likely that a transient burst error appears instead of 1-bit flips due to the present constraints [25]. Moreover, most internal faults are modeled by transient random errors [25]. It is noteworthy that the results of our simulations are valid for the transient errors. Furthermore, in case of occurring permanent internal faults, the same simulation results are achieved.

We use stuck-at error model at the outputs of the AES transformations. This type of error forces multiple nodes to be stuck at logic one (for stuck-at one) or zero (for stuck-at zero) independent of the error-free values. It is noted that we use Fibonacci implementation of the Linear Feedback Shift Registers (LFSRs) with 128 output taps for injecting random multiple errors, where the numbers, locations, and types of the errors are randomly chosen. In this regard, maximum sequence length polynomial for the feedback is selected as $L(X) = X^{128} + X^{29} + X^{27} + X^2 + 1$ according to the maximum sequence length taps presented in [26].

We use the fault detection schemes presented in the previous section and shown in Figs. 5 and 7 for the AES encryption and decryption, respectively. In our simulations using Xilinx ISE version 9.1 Simulator [4], we use the error indication flags at the outputs of ShiftRows (cover the errors for SubBytes and ShiftRows) and AddRoundKey (cover the errors for MixColumns and AddRoundKey) for the AES encryption in Fig. 5. Moreover, for the AES decryption in Fig. 7, we obtain the error indication flags at the outputs of InvSubBytes (cover the errors for InvShiftRows and InvSubBytes) and InvMixColumns (cover the errors for AddRoundKey and InvMixColumns). The results of our simulations show that by having these two sets of error indication flags, an acceptable error coverage is achieved.

In our simulations, we inject errors in two manners, i.e., burst and random errors, and obtain the error coverage for these two cases, the details of which are as follows:

Burst errors. The first type of errors that we consider is the burst errors. For this type of errors, we assume that stuck-at errors occur at the output of only one transformation at a time, i.e., the errors are injected at the 128-bit output of only one transformation in the AES encryption/decryption in Figs. 5 and 7. This includes both stuck-at zero and stuck-one errors. Then, using two series of 16-bit signatures shown in these figures, the error coverage is obtained. The results of our simulations for the burst errors in the AES encryption and decryption are shown in Fig. 8. In this figure, the solid and dashed lines represent the error coverage for the AES encryption and decryption, respectively. As seen in this figure, we have injected up to 700,000 burst errors at the transformation outputs, one at a time, and have monitored the errors that are covered by the error indication flags. It is noted that because the errors are injected only at the output of one transformation, only one of the two series of the error indication flags can detect them. As seen in this figure, after injecting up to 700,000 burst errors, for both the AES encryption and decryption, the error coverage for the two sets of error indication flags is greater than 99.996 percent.

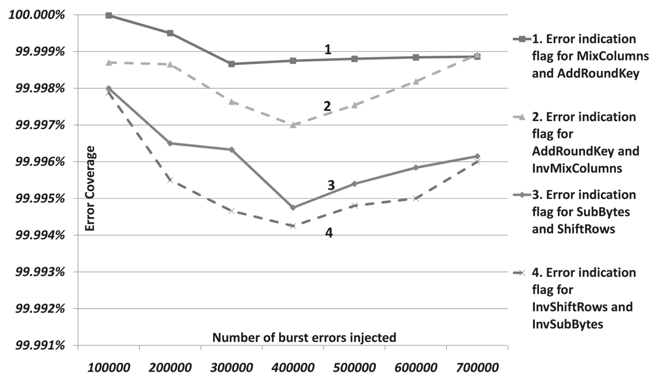


Fig. 8. Simulation results for the error coverages of the proposed fault detection schemes.

Random errors. The second type of errors is random errors, where errors are injected at random locations, i.e., four 128-bit outputs of the transformations. Our simulations show that after injecting up to 700,000 random errors, the higher error coverages of very close to 100 percent are obtained, i.e., all the errors are covered by at least one of the two series of the error indication flags. We also expect the error coverage of close to 100 percent if we increase the number of errors injected. The high error coverages of the proposed scheme for the AES rounds are suitable for the security-constrained applications on FPGAs. These include any AES algorithms implemented on the FPGAs as well as the bitstream security mechanisms.

6 AES FPGA Implementations and Comparisons

The proposed schemes in this paper are structure-independent and can be applied to the AES using both the LUT-based and the composite field S-boxes and inverse S-boxes. In this section, we have implemented both of these structures so that we are able to compare the results for the presented schemes with those using LUTs and composite fields. In what follows, we consider the implementation of both the AES encryption and decryption.

For the FPGA implementations, we have used VHDL as the design entry for ISE version 9.1. Furthermore, the synthesis is performed using Xilinx Synthesis Tool (XST) on

TABLE 1
Comparisons of the Implementations of the Fault Detection Schemes of the AES Using LUT S-Boxes and Inverse S-Boxes on Xilinx FPGAs

FPGA family (Device)	FDS		Encryption				Decryption			
	Scheme	EC(%)	Slice *(overhead)*	Freq. (MHz)	Thro. (Gbps)	Eff.(Mbps /slice)	Slice *(overhead)*	Freq. (MHz)	Thro. (Gbps)	Eff.(Mbps /slice)
Virtex™-4 (xc4vlx160 -12)	Original	-	18335 *(-)*	240.5	30.8	1.7	19322 *(-)*	203.5	26.0	1.3
	Algorithm-level [14]	100%	38273 *(108.7%)*	194.9	24.9[a]	0.6	38273 *(98.1%)*	194.9	24.9[a]	0.6
	Hardware Redundancy	100%	28905 *(57.6%)*[b]	240.5	30.8	1.1	35421 *(83.3%)*	203.5	26.0	0.7
	FD in [8][c] for encryption FD in [10][c] for decryption	99.997%	39104 *(113.3%)*	163.5	20.9	0.5	40244 *(108.3%)*	145.4	18.6	0.5
	FD in [13][d] from the general scheme in [9]	98.7% sin. 48-53% mult.	21211 *(15.7%)*	240.5	30.8	1.4	22280 *(15.3%)*	203.5	26.0	1.1
	Proposed (Figs. 5,7)	**99.996%**	**20127** *(9.8%)*	**240.5**	**30.8**	**1.5**	**20909** *(8.2%)*	**203.5**	**26.0**	**1.2**
Virtex™-5 (xc5vlx110 -3)	Original	-	2960 *(-)*	371.7	47.6	16.1	3906 *(-)*	296.3	37.9	9.7
	Algorithm-level [14]	100%	5849 *(97.6%)*	284.4	36.4[a]	6.2	5849 *(49.7%)*	284.4	36.4[a]	6.2
	Hardware Redundancy	100%	4637 *(56.7%)*[b]	371.7	47.6	10.2	7200 *(84.3%)*	296.3	37.9	5.5
	FD in [8][c] for encryption FD in [10][c] for decryption	99.997%	5590 *(88.9%)*	282.8	36.2	6.5	6688 *(71.2%)*	260.2	33.3	4.9
	FD in [13][d] from the general scheme in [9]	98.7% sin. 48-53% mult.	3619 *(22.3%)*	304.0	38.9	10.7	4426 *(13.3%)*	277.0	35.5	8.0
	Proposed (Figs. 5,7)	**99.996%**	**3757** *(26.9%)*	**371.7**	**47.6**	**12.7**	**4286** *(9.7%)*	**296.3**	**37.9**	**8.8**

a. The latency is twice as much as the original AES encryption or decryption.
b. Although the overhead of greater than 100% is expected, the overhead for the number of occupied slices is less.
c. Using two (256×9) memories for the fault detection of each S-box or inverse S-box.
d. Using (256×9) memories for the fault detection of each S-box or inverse S-box.

Virtex-4 and Virtex-5 families [4]. It is noted that the results of the implementations in this section, i.e., the number of occupied slices and the minimum periods (maximum working frequencies), are all postplace and route results.

We have implemented the original AES using LUT-based S-boxes and inverse S-boxes on Virtex-4 (xc4vlx160-12) and Virtex-5 (xc5vlx110-3) devices. It is noted that these larger devices are chosen to have enough number of slices needed for the fault detection scheme in [8] and [10]. We have used pipelined distributed memories for the LUT-based S-boxes and inverse S-boxes in the AES to increase the design speed and the overall frequency. The XST uses the LUT resources in the FPGAs in order to implement the distributed memories. Furthermore, pipelining is achieved by describing the necessary registers in the design-entry language. The schemes in [14], [8], [10], [13], Hardware Redundancy, and the proposed ones in this paper (see Section 4) have been implemented and the results are depicted in Table 1. As seen in this table, the Error Coverage (EC percent), the number of occupied slices, the maximum working frequency (in megahertz), the throughput (in gigabits per second), and the efficiency (in megabits per second/slice) for the original schemes and the Fault Detection (FD) ones are derived. Moreover, the slice overheads (overheads for the number of occupied slices) are presented. It is noted that there is a difference in the implementations of the LUT-based S-boxes and inverse S-boxes using distributed memories for the selected FPGAs. Specifically, for Virtex-5 and Virtex-4, 256 and 64 bits per CLB are specified for the distributed memories, respectively. This causes the LUT implementations for Virtex-5 to be more compact as compared to those on Virtex-4 [4]. This can be observed in Table 1. In this regard, the number of slices for the original AES encryption and decryption using LUTs and the slice overhead for the scheme in [8] and [10] whose area overhead is dominated by the expansion of the S-box to 512×9 memories is less on Virtex-5. This makes Virtex-5 a suitable device family for the AES using memory-based S-boxes and inverse S-boxes and their fault detection schemes. Because of the higher number of slices for the original AES encryption and decryption on Virtex-4, the slice overheads of the proposed schemes and the scheme in [13] are less as compared to those for Virtex-5.

As shown in Table 1, the number of slices for the original decryption is more than that of the encryption. This is mainly because of the InvMixColumns transformation which is more complex than MixColumns in the AES encryption. Furthermore, the slice overhead for the scheme in [10] in which the LUTs sizes are expanded to 512×9 is less on Virtex-5 family compared to Virtex-4. As shown in Table 1 in bold faces, the proposed structure-independent scheme for the AES decryption is the most efficient and the most compact one among the other schemes. Moreover, for the Virtex-5, the proposed scheme for the AES encryption has the least slice overhead. However, the slice overhead of the proposed scheme implemented on Virtex-4 is slightly more than that of the scheme in [13]. It is noted that the low overhead of the scheme in [13] is because it uses 1-bit signatures for the 128-bit block of data. While the proposed schemes and the one in [8] and [10] use 16 bits for each 128-bit block. As shown in Table 1, this leads to much higher error coverage.

The scheme in [19] is based on using the output of the multiplicative inversion (not that of the S-box) to obtain a signature for fault detection. This scheme cannot be applied to the S-boxes using LUTs where the output of the multiplicative inversion is not accessible. Therefore, we have implemented the original AES encryption which uses the S-boxes using polynomial basis and composite fields in order to have access to the output of the multiplicative inversion. For this reason, we utilize the AES presented in [23]. This implementation of the AES is a hardware optimization for the scheme in [27], which is extensively used in the literature, see, for example, [28], [29]. Then, we have implemented the scheme of [19] and compared it with the proposed scheme (see Section 4) presented in this paper. Moreover, the scheme in [14] and Hardware Redundancy have been implemented.

TABLE 2
Implementation Comparisons of the Fault Detection Schemes of the AES Encryption Using Composite Field S-Boxes on Xilinx FPGAs

FPGA	S-boxes structures[a]	FDS	Slice			Thro. (Gbps)		Eff. (Mbps/slice)		
			Org.	FDS	Over.	Org.	FDS	Org.	FDS	deg.
Virtex™-4	PB[b]	Algorithm-level [14]	7498	17075	127.7%	14.6	14.6[c]	1.9	0.9	52.6%
	PB[b]	Hardware Redundancy	7498	14968	99.6%	14.6	14.6	1.9	1.0	47.4%
	PB[b]	[19]	7498	10340	37.9%	14.6	12.3	1.9	1.2	36.8%
	PB[b]	**Proposed (Sec. 4, Fig. 5)**	**7498**	**9252**	**23.4%**	**14.6**	**12.6**	**1.9**	**1.4**	**26.3%**
	NB[d]	**Proposed[e]**	**6752**	**9325**	**38%**	**17.1**	**12.9**	**2.5**	**1.4**	**44%**
	NB[d]	**Proposed (Sec. 4, Fig. 5)**	**6752**	**8216**	**21.7%**	**17.1**	**12.3**	**2.5**	**1.8**	**28.0%**
Virtex™-5	PB[b]	Algorithm-level [14]	3718	7492	101.5%	18.2	15.7	4.9	2.1	57.1%
	PB[b]	Hardware Redundancy	3718	7162	92.6%	18.2	16.7	4.9	2.3	53.1%
	PB[b]	[19]	3718	4750	27.8%	18.2	14.2	4.9	3.0	38.8%
	PB[b]	**Proposed (Sec. 4, Fig. 5)**	**3718**	**4354**	**17.1%**	**18.2**	**14.6**	**4.9**	**3.4**	**30.6%**
	NB[d]	**Proposed[e]**	**3692**	**4683**	**26.8%**	**17.9**	**14.8**	**4.8**	**3.1**	**35.4%**
	NB[d]	**Proposed (Sec. 4, Fig. 5)**	**3692**	**4286**	**16.1%**	**17.9**	**14.5**	**4.8**	**3.4**	**29.2%**

a. The original AES encryption implementations differ only in the S-boxes structures.
b. The S-boxes using polynomial basis presented in [23].
c. Although the throughput is the same as the original AES, the latency is twice as much as the original one.
d. The original S-boxes using normal basis in composite fields proposed in [30] and verified with the FPGA implementations in [18].
e. Fault detection scheme for the S-boxes from [18] and the proposed scheme in Section 4 for the other transformations.

The results of the implementations are shown in Table 2. It is worth noting that in [19], the fault detection scheme for the AES decryption is not presented. Therefore, no comparison for the AES decryption with this scheme is presented in this table. It is noted that we have not used subpipelining for the implementations and registers are only used at the output of each round. Using subpipelining for the S-boxes using composite fields, one can reach higher working frequencies compared to those for LUT-based S-boxes. As seen in this table, the number of slices for the original AES encryption using S-boxes in composite fields is less than those of the LUTs for Virtex-4 (compare Tables 1 and 2 for Virtex-4). However, the original AES using LUT-based S-boxes is more compact when Virtex-5 is used. As mentioned before, this is due to the low number of slices needed for the implementation of the memories in this device family. As shown in Table 2, the proposed scheme is the most compact and the most efficient scheme compared to the scheme in [19], i.e., the efficiency degradations (percent degradation from the efficiency of the original operations) and the slice overheads are the least for two devices. It is noted that the proposed scheme in this paper uses 16 error indication flags for the 128-bit output states of the transformations. However, the scheme in [19] utilizes 32 error indication flags for each output state. Therefore, more slice overhead and greater error coverage are expected for that scheme. However, as discussed earlier, this scheme cannot be applied to the AES using LUTs.

Furthermore, we have compared the proposed schemes in this paper with the lightweight concurrent fault detection scheme for the AES S-boxes presented in [18]. This scheme is based on using normal basis for logic gate implementations of the S-boxes in the AES encryption. In this fault detection scheme, the structure of the S-box using normal basis has been divided into five blocks. Then, the predicted parities of these blocks are obtained. Moreover, through an exhaustive search among all available composite fields, the optimum solution for the least overhead S-box and its parity predictions is achieved. We have implemented the AES encryption with the original S-boxes using normal basis in composite fields proposed in [30] and verified with the FPGA implementations in [18]. Then, the fault detection scheme for the S-boxes in [18] has been utilized for the SubBytes transformation, while the proposed scheme in this paper (see Section 4) is used for the other transformations. In other words, we derive five error indication flags for each S-box in SubBytes ($5 \times 16 = 90$ flags for the entire SubBytes transformation), while the scheme in Fig. 5 is used for other AES encryption transformations using 16-bit flags. Moreover, the proposed signature-based structure-independent scheme in this paper, i.e., the scheme in Fig. 5, has been implemented for the AES encryption with the S-boxes using normal basis. The results of these implementations are also presented and compared in Table 2. As seen in this table, the FPGA implementations of the original AES encryption with the S-boxes using normal basis representation in composite fields have less area compared to the traditional ones using polynomial basis, i.e., 6,752 and 3,692 compared to 7,498 and 3,718 for two devices, respectively. In addition, the proposed structure-independent scheme in this paper has the least area overhead complexities and the most efficiencies for both FPGA families. At this point, we would like to mention that for the scheme in [18], higher error coverage and slightly higher throughput are achieved compared to the proposed scheme in this paper. However, this is at the cost of the higher area overhead complexity. It is also noted that the fault detection scheme in [18] not only can be only applied for the composite field S-boxes, but is also dependent on the composite fields and normal basis chosen, i.e., the parity predictions would be different if other composite fields are used. Whereas the proposed scheme in this paper is independent of the structures of the S-boxes used in the AES encryption.

Very recently, a fault-tolerant approach which is resistant to fault attacks is proposed in [21]. This approach is based on protecting the logic blocks and memories of the AES. To protect the combinational logic blocks used in the four rounds of the AES, either the parity-based scheme proposed in [10] or the duplication one presented in [31] is implemented. Furthermore, to protect the memories used for storing the expanded key and the state matrix either the Hamming or Reed-Solomon error correcting code is implemented. The results of the comparison of the proposed scheme in this paper with the parity-based scheme of [8] and [10] for protecting the combinational logic elements of the AES are depicted in Table 1. Moreover, for certain AES implementations containing storage elements, one can use the error correcting code-based approach presented in [21] in addition to the proposed scheme in this paper to make a more reliable AES implementation.

7 Conclusions

In this paper, we have studied a number of fault detection schemes for the encryption and the decryption of the AES. New fault detection schemes which are independent of the structures of the S-boxes and the inverse S-boxes have been proposed. Our simulations show that for the AES encryption and decryption, these structure-independent schemes reach the error coverage of approximately 100 percent.

Furthermore, our proposed fault detection schemes and almost all of the previously reported ones have been implemented on the recent Xilinx Virtex FPGAs. Their area and delay overheads for the AES encryption and decryption have been derived and compared. In our implementations, we have considered using both the lookup-table-based and the composite field AES structures. Our FPGA implementations show that for the AES encryption, the slice overhead of the proposed scheme is around 9.8-26.9 percent, depending on the FPGA family and the AES implementation. In addition, for the AES decryption, lower slice overhead is achieved. These slice overheads are less than those for the other schemes which have the same error coverages.

According to our simulation and implementation results, with acceptable error coverages, the structure-independent schemes proposed in this paper have the highest efficiencies, showing reasonable area and time complexity overheads. Based on the AES structure chosen, the performance goals to achieve, and the resources available, one can use combinations of the presented schemes in order to have much more reliable AES encryption and decryption structures.

Acknowledgments

The authors would like to thank the reviewers for their comments. A preliminary version of parts of the paper was presented at the fourth Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC '07) [32]. This work has been supported in part by an NSERC Discovery grant awarded to Arash Reyhani-Masoleh.

References

[1] National Institute of Standards and Technologies, "Announcing the Advanced Encryption Standard (AES),"Federal Information Processing Standards Publication, no. 197, Nov. 2001.
[2] M. Akkar and C. Giraud, "An Implementation of DES and AES, Secure against Some Attacks," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '01)*, pp. 315-325, May 2001.
[3] S. Trimberger, "Security in SRAM FPGAs," *IEEE Design and Test of Computers,* vol. 24, no. 6, p. 581, Nov. 2007.
[4] Xilinx, http://www.xilinx.com/, 2010.
[5] P. Dusart, G. Letourneux, and O. Vivolo, "Differential Fault Analysis on AES," *Proc. Int'l Conf. Applied Cryptography and Network Security (ACNS '03),* pp. 293-306, Oct. 2003.
[6] G. Piret and J.J. Quisquater, "A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '03),* pp. 77-88, Sept. 2003.
[7] J. Blomer and V. Krummel, "Fault Based Collision Attacks on AES," *Proc. Int'l Workshop Fault Diagnosis and Tolerance in Cryptography (FDTC '06),* pp. 106-120, Oct. 2006.
[8] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "A Parity Code Based Fault Detection for an Implementation of the Advanced Encryption Standard," *Proc. IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '02),* pp. 51-59, Nov. 2002.
[9] R. Karri, G. Kuznetsov, and M. Goessel, "Parity-Based Concurrent Error Detection of Substitution-Permutation Network Block Ciphers," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '03),* pp. 113-124, Sept. 2003.
[10] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and 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.
[11] G. Bertoni, L. Breveglieri, I. Koren, and P. Maistri, "An Efficient Hardware-Based Fault Diagnosis Scheme for AES: Performances and Cost," *Proc. IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '04),* pp. 130-138, Oct. 2004.
[12] L. Breveglieri, I. Koren, and P. Maistri, "Incorporating Error Detection and Online Reconfiguration into a Regular Architecture for the AES," *Proc. IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '05),* pp. 72-80, Oct. 2005.
[13] K. Wu, R. Karri, G. Kuznetsov, and M. Goessel, "Low Cost Concurrent Error Detection for the Advanced Encryption Standard," *Proc. Int'l Test Conf. '04,* pp. 1242-1248, Oct. 2004.
[14] R. Karri, K. Wu, P. Mishra, and K. Yongkook, "Fault-Based Side Channel Cryptanalysis Tolerant Rijndael Symmetric Block Cipher Architecture," *Proc. IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '01),* pp. 418-426, Oct. 2001.
[15] C.H. Yen and B.F. Wu, "Simple Error Detection Methods for Hardware Implementation of Advanced Encryption Standard," *IEEE Trans. Computers,* vol. 55, no. 6, pp. 720-731, June 2006.
[16] T.G. Malkin, F.X. Standaert, and M. Yung, "A Comparative Cost/Security Analysis of Fault Attack Countermeasures," *Proc. Int'l Workshop Fault Diagnosis and Tolerance in Cryptography (FDTC '06),* pp. 159-172, Oct. 2006.
[17] A. Satoh, T. Sugawara, N. Homma, and T. Aoki, "High-Performance Concurrent Error Detection Scheme for AES Hardware," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '08),* pp. 100-112, Aug. 2008.
[18] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A Lightweight Concurrent Fault Detection Scheme for the AES S-Boxes Using Normal Basis," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '08),* pp. 113-129, Aug. 2008.
[19] M. Karpovsky, K.J. Kulikowski, and A. Taubin, "Differential Fault Analysis Attack Resistant Architectures for the Advanced Encryption Standard," *Proc. Conf. Smart Card Research and Advanced Applications (CARDIS '04),* vol. 153, pp. 177-192, Aug. 2004.
[20] P. Maistri and R. Leveugle, "Double-Data-Rate Computation as a Countermeasure against Fault Analysis," *IEEE Trans. Computers,* vol. 57, no. 11, pp. 1528-1539, Nov. 2008.
[21] C. Moratelli, F. Ghellar, E. Cota, and M. Lubaszewski, "A Fault-Tolerant DFA-Resistant AES Core," *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS '08),* pp. 244-247, May 2008.
[22] A. Reyhani-Masoleh and M. Hasan, "Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over $GF(2^m)$," *IEEE Trans. Computers,* vol. 53, no. 8, pp. 945-959, Aug. 2004.
[23] X. Zhang and K.K. Parhi, "High-Speed VLSI Architectures for the AES Algorithm," *IEEE Trans. Very Large Scale Integration Systems,* vol. 12, no. 9, pp. 957-967, Sept. 2004.
[24] R. Zimmermann and W. Fichtner, "Low-Power Logic Styles: CMOS versus Pass-Transistor Logic," *IEEE J. Solid-State Circuits,* vol. 32, no. 7, pp. 1079-1090, 1997.
[25] L. Breveglieri, I. Koren, and P. Maistri, "An Operation-Centered Approach to Fault Detection in Symmetric Cryptography Ciphers," *IEEE Trans. Computers,* vol. 56, no. 5, pp. 534-540, May 2007.

[26] M. George and P. Alfke, "Linear Feedback Shift Registers in Virtex Devices," *Xilinx Application Note 210,* http://www.xilinx.com/support/documentation/application_notes/xapp210.pdf, 2010.

[27] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," *Proc. Int'l Conf. Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT '01),* pp. 239-254, Dec. 2001.

[28] S. Morioka and A. Satoh, "An Optimized S-Box Circuit Architecture for Low Power AES Design," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '02),* pp. 172-186, Aug. 2002.

[29] F.X. Standaert, G. Rouvroy, J.J. Quisquater, and J.D. Legat, "Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '03),* pp. 334-350, Sept. 2003.

[30] D. Canright, "A Very Compact S-Box for AES," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '05),* pp. 441-455, Sept. 2005.

[31] C. Moratelli, E. Cota, and M. Lubaszewski, "A Cryptography Core Tolerant to DFA Fault Attacks," *Proc. Ann. Symp. Integrated Circuits and Systems Design (SBCCI '06),* pp. 190-195, Sept. 2006.

[32] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A Structure-Independent Approach for Fault Detection Hardware Implementations of the Advanced Encryption Standard," *Proc. Int'l Workshop Fault Diagnosis and Tolerance in Cryptography (FDTC '07),* pp. 47-53, Sept. 2007.

Mehran Mozaffari-Kermani received the BSc degree in electrical engineering and computer engineering from the University of Tehran in 2005, and the MESc degree in electrical engineering and computer engineering in 2007 from the University of Western Ontario, where he is currently working toward the PhD degree at the Department of Electrical and Computer Engineering. His current research interests include secure cryptographic systems, fault diagnosis and tolerance, VLSI reliability, and computer arithmetic. He is a student member of the IEEE.

Arash Reyhani-Masoleh received the BSc degree in electrical and electronic engineering from Iran University of Science and Technology in 1989, the MSc degree in electrical and electronic engineering from the University of Tehran in 1991, both with the first rank, and the PhD degree in electrical and computer engineering from the University of Waterloo in 2001. From 1991 to 1997, he was with the Department of Electrical Engineering, Iran University of Science and Technology. From June 2001 to September 2004, he was with the Centre for Applied Cryptographic Research, University of Waterloo. In October 2004, he joined the Department of Electrical and Computer Engineering, University of Western Ontario, London, Canada, as an assistant professor. His current research interests include algorithms and VLSI architectures for computations in finite fields, fault-tolerant computing, and error-control coding. He was awarded a Natural Sciences and Engineering Research Council of Canada (NSERC) Postdoctoral Fellowship in 2002. Currently, he is an associate editor of *Integration*, the VLSI Journal (Elsevier). He is a member of the IEEE and the IEEE Computer Society.