Simple Error Detection Methods for Hardware Implementation of Advanced Encryption Standard

Chih-Hsu Yen and Bing-Fei Wu, *Senior Member*, *IEEE*

Abstract—In order to prevent the Advanced Encryption Standard (AES) from suffering from differential fault attacks, the technique of error detection can be adopted to detect the errors during encryption or decryption and then to provide the information for taking further action, such as interrupting the AES process or redoing the process. Because errors occur within a function, it is not easy to predict the output. Therefore, general error control codes are not suited for AES operations. In this work, several error-detection schemes have been proposed. These schemes are based on the $(n+1, n)$ cyclic redundancy check (CRC) over $GF(2^8)$, where $n \in \{4, 8, 16\}$. Because of the good algebraic properties of AES, specifically the `MixColumns` operation, these error detection schemes are suitable for AES and efficient for the hardware implementation; they may be designed using round-level, operation-level, or algorithm-level detection. The proposed schemes have high fault coverage. In addition, the schemes proposed are scalable and symmetrical. The scalability makes these schemes suitable for an AES circuit implemented in 8-bit, 32-bit, or 128-bit architecture. Symmetry also benefits the implementation of the proposed schemes to achieve that the encryption process and the decryption process can share the same error detection hardware. These schemes are also suitable for encryption-only or decryption-only cases. Error detection for the key schedule in AES is also proposed and is based on the derived results in the data procedure of AES.

Index Terms—Advanced encryption standard, error control code, CRC, differential fault attacks.

1 Introduction

The Advanced Encryption Standard (AES) [10], the successor to the Data Encryption Standard (DES), was finalized in October 2000 by the US National Institute of Standards and Technology (NIST), when the Rijndael algorithm [12] was adopted. The data block size of AES is 128-bit and the key size can be 128-bit, 192-bit, or 256-bit. In AES, although the data block is 128-bit, all operations are byte-oriented over $GF(2)$ or $GF(2^8)$. Therefore, several kinds of AES implementations have been discussed. In general, three main types of AES implementations have been discussed, 8-bit, 32-bit, or 128-bit architecture. Each architecture has its own applications. Feldhofer et al. [6] designed an 8-bit AES chip to provide security for radio frequency identification (RFID). Satoh et al. [13] introduced a 32-bit implementation of AES. Mangard et al. [9] proposed a scalable architecture for AES, which could process 128-bit data or 32-bit data, depending on the number of Sbox.

The hardware implementation of AES would be countered by some side-channel attacks, such as Differential Fault Attacks (DFA) or Differential Power Analysis (DPA). Differential fault attacks was originally proposed by Biham and Shamir [4]. Theses side-channel attacks actually threaten the security of several cryptosystems because they are practical for a crypto module. The idea of DFA is to apply the differential attacks to a crypto module or a crypto chip. The cryptanalyst injects errors by using microwave or ionizing techniques during the encryption or decryption process. These errors cause the encryption results to differ from the correct results; hence, the cryptanalyst will receive the difference of outputs. Therefore, such differential attacks may be carried out in the real world. Dusart et al. [5] broke the 128-bit AES under the assumption that you can physically modify the hardware AES device. This attack required 34 pairs of differential inputs and outputs to obtain the final round key. Piret and Quisquater [11] broke AES with two erroneous ciphertext under the assumption that the errors occur between the antepenultimate and the penultimate `MixColumns`.

To avoid the possibility of suffering such attacks, error detection can be considered while implementing a cipher. In 2002, Karri et al. [7] proposed a general error detection method, called concurrent error detection (CED), for several symmetric block ciphers including RC6, MARS, Serpent, Twofish, and Rijndael. CED requires an inverse operation to check whether errors have occurred in calculations or not and has three levels: the operation level, the round level, and the algorithm level. Taking an operation-level CED in AES as an example, the `InvSubBytes` is required to detect the errors occurring in `SubBytes` and vice versa. This method has very high fault coverage, but it is time-consuming and high hardware cost because inverse operations are required. In 2003, Karri et al. [8] proposed a parity-based detection technique for general substitution-permutation block ciphers. However, the size of the table, required by the substitution box, is enlarged. In addition, the paper did not address the error detection techniques for some specific functions, such as `MixColumns` in AES. In 2004, Wu et al. [14] applied the structure of [8] to AES and used

• *The authors are with the Department of Electrical and Control Engineerng, National Chiao Tung University, 1001 Ta Hsueh Rd., Hsinchu, Taiwan 300, ROC. E-mail: {zsyian, bwu}@cssp.cn.nctu.edu.tw.*

Manuscript received 19 Jan. 2005; revised 27 Aug. 2005; accepted 7 Sept. 2005; published online 21 Apr. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0014-0105.

one-bit parity for a 128-bit data block. The method of Wu et al. [14] can let the parity pass through the MixColumns. Bertoni et al. [1] used an error detection code of 16-bit parity for a 128-bit data block. To be precise, this approach uses one-bit parity for each byte and, thus, can detect all single errors and perhaps all odd errors. In [2], Bertoni et al. used the error detection scheme in [1] not only to detect errors but also to locate errors. In 2004, Bertoni et al. [3] implemented the model proposed in [2]. The introduction of the mode into AES brought the performance 18 percent overhead of area and 26 percent decreasing of throughput. According to the results given in [1], their approach was able to detect most cases of multiple faults. However, this approach is asymmetrical, between MixColumns and InvMixColumns, because the parity prediction of InvMixColumns is more complex than that of MixColumns. Therefore, two circuits are required to predict the parity while merging the encryption and the decryption. Besides, the detection technique for SubBytes doubled the table size of SubBytes in AES, from 256 to 512 bytes. In addition, it cannot be easily applied to an AES implementation of 8-bit architecture because the parity prediction of MixColumns (InvMixColumns) requires information from other bytes and other parities.

This work proposes several error-detection schemes for AES. They are based on the $(n+1, n)$ cyclic redundancy check (CRC) over $GF(2^8)$, where $n \in \{4, 8, 16\}$ is the number of bytes contained in the message. The proposed schemes easily predict the parity of an operation's output. Because AES is byte-oriented and its constants are ingeniously designed, the parity of the output can be predicted from a linear combination of the parity of the input. In most cases, the parity is the summation of the input data; also, the proposed schemes are highly scalable and are suitable for 8-bit, 32-bit, or 128-bit architecture. This is important because many AES designs are in an AES hardware designed as either 8-bit or 32-bit architecture. Another advantage of the proposed approaches is that the parity calculation between the encryption and the decryption is symmetric because the parity generation in encryption is quite similar to the one in decryption. This will bring some benefits while integrating encryption and decryption into one circuit.

This paper is organized as follows: In Section 2, the AES algorithm is briefly described and the notations used throughout are defined. In Section 3, our proposed error detection schemes for AES are described. Derivation of error detection for each operation, including SubBytes, ShiftRows, MixColumns, and AddRoundKey, is explained, as well as the design of the key schedule. The undetectable errors of each proposed method are theoretically analyzed in Section 4, while, in Section 5, the realization issues of three levels, operation level, round level, and algorithm level, are described. In Section 6, advantages and comparisons between this work and other research studies are discussed and, in Section 7, the detection capability of each scheme is simulated. Finally, our conclusions are offered in Section 8.

2 AES Algorithm

The AES [10] consists of two parts, the data procedure and the key schedule. The data procedure is the main body of the encryption (decryption) and consists of four operations, (Inv)SubBytes, (Inv)ShiftRows, (Inv)MixColumns, and (Inv)AddRoundKey. During encryption, these four operations are executed in a specific order—AddRoundKey, a number of rounds, and then the final round. The number of rounds is 10, 12, or 14, respectively, for a key size of 128 bits, 192 bits, or 256 bits. Each round is comprised of the four operations and the final round has SubBytes, ShiftRows, and AddRoundKey. The decryption flow is simply the reverse of the encryption, and each operation is the inverse of the corresponding one in encryption. In the data procedure, the 16-byte (128-bit) data block is rearranged as a 4×4 matrix, called state S,

$$S = \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix}, \tag{1}$$

where s_i denotes the ith byte of the data block. In this context, S denotes the input of an operation and T denotes the output. AES is operated in two fields, $GF(2)$ and $GF(2^8)$. In $GF(2)$, addition is denoted by $\oplus$, and multiplication is denoted by $\otimes$. Similarly, the two symbols, $+$ and $\times$, denote addition and multiplication in $GF(2^8)$.

2.1 SubBytes

Two calculations, the $GF(2^8)$ inversion and the affine transformation, are involved in this operation. SubBytes substitutes each byte s_i of the data block by

$$t_i = As_i^{-1} + \texttt{63}, \tag{2}$$

where s_i^{-1} is the inverse of the input byte, $s_i \in GF(2^8)$, A is an 8×8 circulant matrix of a constant row vector $[1\,0\,0\,0\,1\,1\,1\,1]$ over $GF(2)$, and 63 (the Courier font number representing a hexadecimal value in this paper) belongs to $GF(2^8)$. As_i^{-1} is a matrix-vector multiplication over $GF(2)$.

2.2 ShiftRows

The ShiftRows operation only changes the byte position in the state. It rotates each row with different offsets to obtain a new state as follows:

$$\begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix} \xrightarrow[\texttt{ShiftRows}]{} \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_5 & s_9 & s_{13} & s_1 \\ s_{10} & s_{14} & s_2 & s_6 \\ s_{15} & s_3 & s_7 & s_{11} \end{bmatrix}. \tag{3}$$

The first row is unchanged, the second row is left circular shifted by one, the third row is by two, and the last row is by three.

2.3 MixColumns

The MixColumns operation mixes every consecutive four bytes of the state to obtain four new bytes as follows:

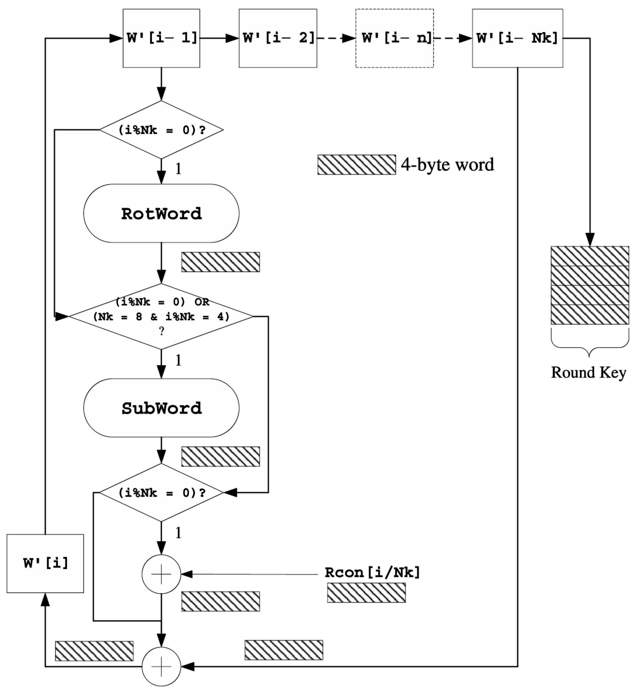

Fig. 1. The block diagram of key expansion in AES.

$$\begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix} \xrightarrow[\texttt{MixColumns}]{} \begin{bmatrix} t_0 & t_4 & t_8 & t_{12} \\ t_1 & t_5 & t_9 & t_{13} \\ t_2 & t_6 & t_{10} & t_{14} \\ t_3 & t_7 & t_{11} & t_{15} \end{bmatrix}. \quad (4)$$

Let s_i, s_{i+1}, s_{i+2}, and s_{i+3} represent every consecutive four bytes, where $i \in \{0, 4, 8, 12\}$. Then, the four bytes are transformed by

$$\begin{bmatrix} t_i \\ t_{i+1} \\ t_{i+2} \\ t_{i+3} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_i \\ s_{i+1} \\ s_{i+2} \\ s_{i+3} \end{bmatrix}. \quad (5)$$

Each entry of the constant matrix in (5) belongs to $GF(2^8)$, hence (5) is a matrix-vector multiplication over $GF(2^8)$.

2.4 AddRoundKey and Key Expansion

Each round has a 128-bit round key which is segmented into 16 bytes k_i as (1); the AddRoundKey operation is simply an addition,

$$t_i = s_i + k_i, \text{where } 0 \leq i \leq 15. \quad (6)$$

The key expansion expands a unique private key as a key stream of $(4r + 4)$ 32-bit words, where r is 10, 12, or 14. The private key is segmented into Nk words according to the key length, where NK is 4, 6, or 8 for a 128-bit, 192-bit, or 256-bit cipher key, respectively. As Fig. 1 shows, then, it generates the ith word (32 bits) by EXORing the $(i - \mathrm{Nk})$th word with either the $(i - 1)$th word or the conditionally transformed $(i - 1)$th word, where $\mathrm{NK} \leq i \leq (4r + 3)$. The $(i - 1)$th word is conditionally transformed by RotWord, SubBytes and EXORing with $\mathrm{Rcon[i/Nk]} = \{02^{\lfloor i/Nk \rfloor}, 00, 00, 00\}$, where the polynomial presentation of $02^{\lfloor i/Nk \rfloor}$ is $x^{\lfloor i/Nk \rfloor}$ over $GF(2^8)$. Finally, the key stream is segmented into several round keys which are involved in the AddRoundKey operation.

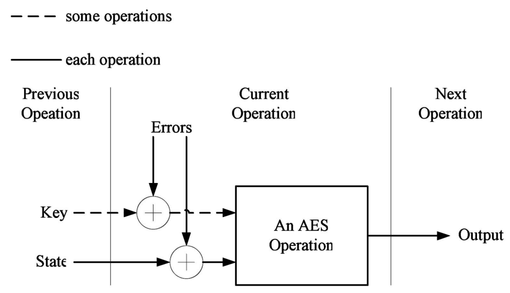

Fig. 2. The error model assumed in this work. The solid line part appears in every operation and the dotted line part appears in some operations.

3 Error Detection Techniques

The parts in decryption can be yielded in a similar way; hence, the following context only addresses the error detection in encryption. The differential faults attacks need differential inputs and outputs to attack a cryptosystem; hence, it is assumed that the states and round keys are polluted by additive errors, as shown in Fig. 2. In this work, one operation is the smallest granule for designing error detection. In Fig. 2, the errors are assumed to be induced between the previous operation and the current operation. If the errors occur in the output of the previous operation, the erroneous input of the current operation will be treated as a different state. Actually, this situation only exists in the first round or in the first operation. The assumed error model is logical, even in the case where the errors occur during the operation. Because each operation of AES is invertible, one unique error block e would exist for an erroneous output T such that $T = f(S + e)$, where f denotes any operation in AES.

This paper adopts a systematic $(n + 1, n)$ cyclic redundancy check (CRC) over $GF(2^8)$ to detect errors occurring during encryption, where $n \in \{4, 8, 16\}$ is the number of bytes contained in the message. The generator polynomial is

$$g(x) = 1 + x, \quad (7)$$

where the coefficients of (7) are over $GF(2^8)$. Giving a message $s(x)$ of degree $n - 1$, a systematic codeword, generated by $g(x)$, can be obtained from the following two steps:

1. Obtain the remainder $p(x)$ from dividing $xs(x)$ by the generator polynomial $g(x)$. The remainder $p(x)$ is a scalar p here because the degree of $g(x)$ is one.
2. Combine $p(x)$ and $xs(x)$ to obtain the codeword polynomial,

$$p(x) + xs(x) = p + s_0 x + s_1 x^2 + \cdots + s_{n-1} x^n, \quad \text{where } p, s_i \in GF(2^8). \quad (8)$$

In Step 1, while $g(x)$ is $1 + x$, the remaining $p(x)$ is the summation of all coefficients of the message,

$$p(x) = \sum_{i=0}^{n} s_i. \quad (9)$$

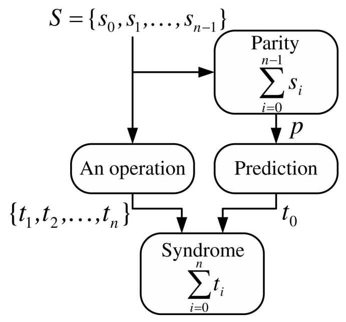

Fig. 3. The block diagram of the error detection in this paper.

Therefore, the parity of a message may be obtained by calculating the summation of the input message over $GF(2^8)$.

Assume that the received polynomial $t(x)$ is

$$t(x) = t_0 + t_1 x + t_2 x^2 + \cdots + t_n x^n, t_i \in GF(2^8). \tag{10}$$

The detection scheme checks whether the syndrome equals zero or not, where syndrome u is

$$u = \sum_{i=0}^{n} t_i. \tag{11}$$

If the syndrome equals zero, then it is assumed that no errors have occurred; otherwise, errors did occur.

In the channel coding field, it is assumed that the message $s(x)$ is transmitted over a noisy channel. The channel does not modify the message if no errors occur. Therefore, it is easy to predict that t_0 is identical to p, with t_0 being used to detect the errors. However, as shown in Fig. 3, the message, $S = \{s_0, s_1, \ldots, s_{n-1}\}$, is transformed into another message, $\{t_1, t_2, \ldots, t_n\}$, by an AES operation; hence, t_0 cannot be obtained instinctively. Therefore, this paper investigates the function, predicting t_0 from p as shown in Fig. 3, for each operation to make error detection possible in AES.

This work applies an $(n+1, n)$ CRC to AES, where $n \in \{4, 8, 16\}$. In the case where, $n = 16$, a 128-bit AES state is treated as a message; hence, only one parity is generated for a 128-bit data block. When $n = 4$, the error detection is designed to check each column of the output state. In other words, four 4-byte column vectors in an AES state, $\{t_{4j+1}, t_{4j+2}, t_{4j+3}, t_{4j+4}\}$, $0 \leq j \leq 3$, are checked separately. Therefore, four parities are required for a 128-bit data block when $n = 4$. For $n = 8$, two parities are required for a 128-bit data block. The following context addresses the two cases, $n = 16$ and $n = 4$, because the $(9, 8)$ CRC for the AES algorithm can be constructed under similar conditions to the $(17, 16)$ or $(5, 4)$ CRC for AES.

3.1 In `SubBytes`

In this paper, two implementation types of `SubBytes` are considered. The first type uses one table instead of the $GF(2^8)$ inversion and the affine transformation. The second type separately calculates the $GF(2^8)$ inversion and the affine transformation and the implementation of the $GF(2^8)$ inversion is not limited to the look-up-table method or the combinational logical circuit. In this paper, the first type is named *united SubBytes* and the second type is *separated SubBytes*.

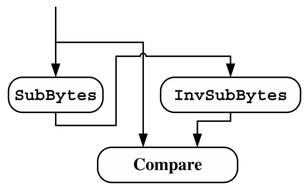

Fig. 4. The error detection for *united SubBytes*.

For *united SubBytes*, it is assumed that both the `Sub Bytes` circuit and the `InvSubBytes` circuit are implemented in a chip. Error detection is achieved by feeding the output of `SubBytes` into `InvSubBytes`, then comparing the input of `SubBytes` and the output of `InvSubBytes`, and vice versa, as Fig. 4 shows. If both are identical, then it is concluded that no errors have occurred. Otherwise, the errors did occur. This error detection method may be time-consuming, if only the `SubBytes` operation is considered. However, in practical terms, normal encryption could be further processed, without waiting for the error detection result, because `SubBytes` is either the first operation or the second operation in each round. In other words, the operation after `SubBytes`, such as `ShiftRows`, `MixColumns`, or `AddRoundKey`, may continue, when the output of the round would be intercepted if errors are detected in `SubBytes`.

If *separated SubBytes* is adopted, error detection must be applied separately to the $GF(2^8)$ inversion and the affine transformation. Considering the error detection for the $GF(2^8)$ inversion first, there are two schemes proposed herein. Similarly to Fig. 4, the first scheme detects errors by using the relationship of the mutual inverse. However, the computation of the $GF(2^8)$ inversion is identical for both `SubBytes` and `InvSubBytes`; hence, this scheme does not require the encryption and decryption circuits to simultaneously exist in one chip. It can be used with the encryption-only or decryption-only hardware.

The second scheme is the $(n+1, n)$ CRC and assumes that the $GF(2^8)$ inversion is implemented in look-up-table approach. Instead of the inverse value of a giving input, the exclusive value of the giving input and its inverse is stored in the table. Therefore, giving an input $\alpha \in GF(2^8)$, the value, $\beta = \alpha + \alpha^{-1}$, is obtained from the table and then the input α is added to β to yield α^{-1}, as the marked block in Fig. 5. The error is detected by the syndrome obtained by the dashed line in Fig. 5. In this diagram, no errors are introduced, hence the syndrome is zero.

For one $GF(2^8)$ inversion, according to Fig. 3 and the error model given in Fig. 2, the errors induce a fault at the input of the $GF(2^8)$ inversion, as shown in Fig. 6. Suppose that the byte s_i is changed into another byte s_i' by adding the error e_0. Then, the syndrome used to detect errors is calculated as

$$(s_i + e_1) + t_{i+1} + (t_{i+1} + t_{i+1}^{-1}) = e_0 + e_1. \tag{12}$$

The one-byte structure of Fig. 5 could be extended to the 4-byte, 8-byte, or 16-byte structure. Taking the 16-byte

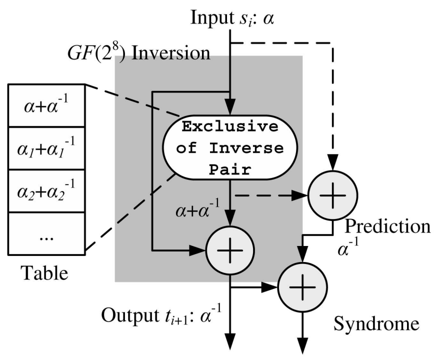

Fig. 5. The block diagram of one $GF(2^8)$ inversion with the error detection.

structure into consideration, the input state is denoted as $S = \{s_0, s_1, \ldots, s_{15}\}$ and then the parity p is $\sum_{i=0}^{15} s_i$ from (9). According to (12) and Fig. 3, the parity of the output parity t_0 could be predicted by

$$\sum_{i=0}^{15} s_i + \sum_{i=0}^{15}(t_{i+1} + t_{i+1}^{-1}), \tag{13}$$

and the syndrome is

$$\begin{aligned} & t_0 + \sum_{i=0}^{15} t_{i+1}, \\ \Rightarrow & \sum_{i=0}^{15} t_{i+1} + p + \sum_{i=0}^{15}(t_{i+1} + t_{i+1}^{-1}). \end{aligned} \tag{14}$$

If no errors have occurred, the value t_{i+1}^{-1} will equal s_i. Therefore, the syndrome (14) is zero.

In this paper, all ShiftRows, MixColumns, and AddRoundKey are protected by error detection code. However, the detection technique of SubBytes is varied with its implementation. According to the error detection scheme for SubBytes, three proposed architectures for AES are denoted by *united-SubBytes detection (USBD, hybrid-SubBytes detection (HSBD)*, and *parity-based-SubBytes detection(PbSBD)*, as shown in Fig. 7.

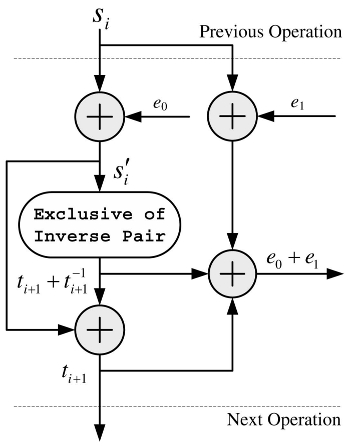

Fig. 6. An error is injected into the input state after entering the $GF(2^8)$ inversion.

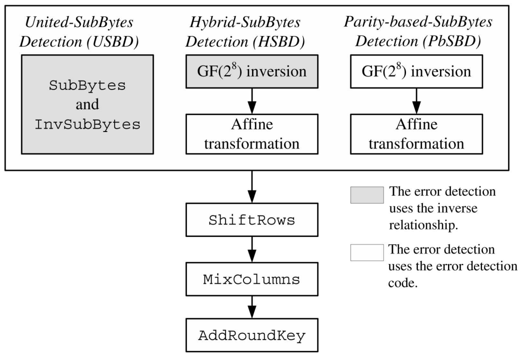
United-SubBytes Detection (USBD)
Hybrid-SubBytes Detection (HSBD)
Parity-based-SubBytes Detection (PbSBD)
SubBytes and InvSubBytes
$GF(2^8)$ inversion
Affine transformation
$GF(2^8)$ inversion
Affine transformation
ShiftRows
MixColumns
AddRoundKey
The error detection uses the inverse relationship.
The error detection uses the error detection code.

Fig. 7. The three proposed architectures for AES.

For the affine transformation, error detection is achieved by the $(n+1, n)$ CRC, where $n \in \{4, 8, 16\}$. Considering $n = 16$ first, and according to (9), the parity p of an input state, $S = \{s_0, s_1, \ldots, s_{15}\}$, where $s_i \in GF(2^8)$, is generated by

$$p = \sum_{i=0}^{15} s_i. \tag{15}$$

The output state is denoted as $T = \{t_0, t_1, \ldots, t_{16}\}$. From (2) and Fig. 3, t_{i+1} is $As_i + 63$, where $0 \le i \le 15$. The hexadecimal constant 63 will be eliminated after taking summation of the output state $T \backslash t_0$, i.e.,

$$\sum_{i=0}^{n-1} t_{i+1} = \sum_{i=0}^{n-1}(As_i + 63) = A\sum_{i=0}^{15} s_i = Ap. \tag{16}$$

Therefore, t_0 can be predicted by (16) with input parity p. If no errors occur, the syndrome u must be zero,

$$u = \sum_{i=0}^{16} t_i = 0. \tag{17}$$

In the case of $(5, 4)$ CRC or $(9, 8)$ CRC, (16) also holds.

3.2 In ShiftRows

From (3), the ShiftRows operation simply rotates the input state S, but does not alter the value of s_i. Therefore, t_0 may be directly predicted by $\sum_{i=0}^{n} s_i$ in the case of $n = 16$. Similarly, the ShiftRows operation is error free if the syndrome is zero

$$\sum_{i=0}^{16} t_i = 0. \tag{18}$$

When $n = 4$, because each column of the output state would be detected, the four parities p_j, where $0 \le j \le 3$, are

$$\begin{aligned} p_0 &= s_0 + s_5 + s_{10} + s_{15}, \\ p_1 &= s_4 + s_9 + s_{14} + s_3, \\ p_2 &= s_8 + s_{13} + s_2 + s_7, \\ p_3 &= s_{12} + s_1 + s_6 + s_{11}; \end{aligned}$$

hence, the $t_{j,0}$ for each output message $\{t_{4j+1}, t_{4j+2}, t_{4j+3}, t_{4j+4}\}$ is p_j. The case of $n = 8$ is analogous to the case of $n = 4$.

3.3 In `MixColumns`

The behavior of the `MixColumns` operation is more complex because each byte in the input state S influences four bytes in the output state T. However, because of the ingenious design of the matrix coefficients, it is also possible to apply the $(n+1, n)$ CRC directly, where $n \in \{4, 8, 16\}$. The `MixColumns` operation works as follows:

$$\underbrace{\begin{bmatrix} t_{4j+1} \\ t_{4j+2} \\ t_{4j+3} \\ t_{4j+4} \end{bmatrix}}_{T'} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \underbrace{\begin{bmatrix} s_{4j} \\ s_{4j+1} \\ s_{4j+2} \\ s_{4j+3} \end{bmatrix}}_{S'}, \text{where } 0 \leq j \leq 3. \tag{19}$$

From (19), it is yielded that the summation of vector T' equals that of vector S'.

$$\begin{aligned} \sum_{k=0}^{3} t_{4j+k+1} = (02 + 01 + 01 + 03)s_{4j} + \\ &(03 + 02 + 01 + 01)s_{4j+1} + \\ &(01 + 03 + 02 + 01)s_{4j+2} + \\ &(01 + 01 + 03 + 02)s_{4j+3}, \\ &= s_{4j} + s_{4j+1} + s_{4j+2} + s_{4j+3}, \\ &= \sum_{k=0}^{3} s_{4j+k}. \end{aligned} \tag{20}$$

Therefore, when the $(5, 4)$ CRC is applied, the output parity $t_{j,0}$ of the jth column vector may be directly predicted from the jth column vector of the input state by $\sum_{k=0}^{3} s_{4j+k}$. Similarly, in the case $n = 16$, t_0 is predicted by

$$\begin{aligned} t_0 &= \sum_{j=0}^{3} \sum_{k=0}^{3} t_{4j+k+1}, \\ &= \sum_{j=0}^{3} \sum_{k=0}^{3} s_{4j+k}, \\ &= \sum_{i=0}^{15} s_i. \end{aligned}$$

Because the summation of `02`, `01`, `01`, and `03` is `01`, (20) can be satisfied for the $(17, 16)$, $(9, 8)$, or $(5, 4)$ CRC. The coefficients of `InvMixColumns` display an identical phenomenon. The summation of the four coefficients used in decryption, `0B`, `0D`, `09`, `0E`, is also `01`. Therefore, t_0 or $t_{j,0}$ can be predicted in the same way as that of `MixColumns`.

3.4 In `AddRoundKey`

Discussing the case $n = 16$ first, it is assumed that each round key already has a parity; hence, the round key is represented as $\{k_0, k_1, \dots, k_{16}\}$, where $k_0 = \sum_{i=0}^{15} k_{i+1}$ is the parity and $\{k_1, \dots, k_{16}\}$ is the normal round key. The `AddRoundKey` operation only adds the input state with a normal key $K = \{k_1, k_2, \dots, k_{16}\}$ to yield the output state as follows:

$$T = S + K. \tag{21}$$

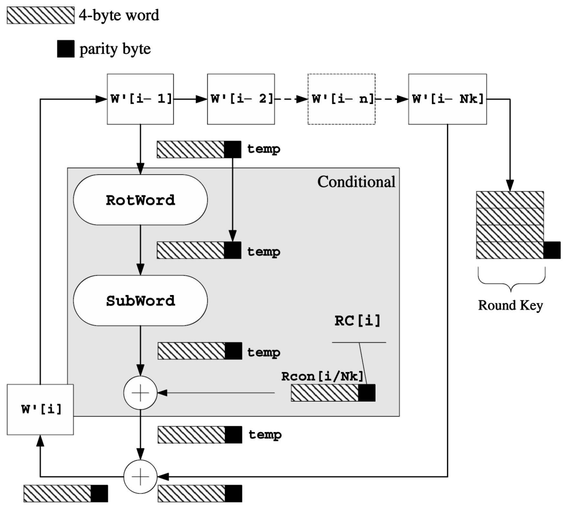

Fig. 8. The error detection scheme for key expansion.

We apply the summation operation to (21) to obtain

$$\sum_{i=0}^{15} t_{i+1} = \sum_{i=0}^{15} s_i + \sum_{i=0}^{15} k_{i+1} = p + k_0. \tag{22}$$

Accordingly, t_0 may be obtained from $p + k_0$. The parities for $n = 4$ or $n = 8$, p_j, are calculated in the same way; however, the round key must also have four or two parities.

3.5 In the Key Expansion

The $(n+1, n)$ CRC is also adopted in key expansion, where $n \in \{4, 8, 16\}$. However, the $(5, 4)$ CRC is always used in the interior of the key expansion. The key expansion and the error detection scheme are jointly depicted in Fig. 8, where the decision blocks are removed from Fig. 1 for a simple description of error detection, as the conditions only determine where the error detection is applied, not how it is designed.

In this key expansion, with error detection, one word contains five bytes and the symbol of a word is denoted by $\texttt{W}'[\texttt{i}] = [\texttt{W}[\texttt{i}] \parallel \text{parity}]$, where $\parallel$ is a catenation symbol. At first, the parities of the first `Nk` words, where $\texttt{Nk} \in \{4, 6, 8\}$, are obtained by the generator $1 + x$, i.e., the parity p_i of $\texttt{W}[\texttt{i}] = [w_{i,0}\ w_{i,1}\ w_{i,2}\ w_{i,3}]$ is

$$p_i = w_{i,0} + w_{i,1} + w_{i,2} + w_{i,3}. \tag{23}$$

Then, the `Nk`-pair parities and messages form new `Nk` words, $\texttt{W}'[0], \texttt{W}'[1], \dots,$ and $\texttt{W}'[\texttt{Nk} - 1]$. The new words are successively put into the `Nk` shift blocks, from $\texttt{W}'[\texttt{i} - \texttt{Nk}]$ to $\texttt{W}'[\texttt{i} - 1]$, at the top of Fig. 8, after which, the key expansion starts. A 128-bit round key and its one-byte parities are collected after each period of four shifts. If $(17, 16)$ CRC is chosen for AES, the one-byte parity of a round key is obtained by summing the four parities of output words. If $(5, 4)$ CRC is chosen, then the four parities are kept.

In the key expansion, the `RotWord` rotates the byte order of $\texttt{W}[\texttt{i} - 1]$; hence, the parity is the same as that of $\texttt{W}'[\texttt{i} - 1]$. For the `SubWord` operation because it is a function which executes `SubBytes` on each byte of input, the error detection scheme is the same as that in `SubBytes`, described in Section 3.1. However, in the case of *united SubBytes* being used, the parity must be calculated separately.

For the EXOR operation with `Rcon[i/Nk]`, the error detection is achieved by EXORing the parity of `temp` and that of `Rcon[i/Nk]`, where $\texttt{Rcon[i/Nk]} = \{\texttt{02}^{\lfloor \texttt{i/Nk} \rfloor}, \texttt{00}, \texttt{00}, \texttt{00}\}$. The parity of `Rcon[i/Nk]` equals $\texttt{02}^{\lfloor \texttt{i/Nk} \rfloor}$ due to the three bytes of zero value in `Rcon[i/Nk]`. At the end of the key expansion, the parity t_0 is the EXOR of the parity of current data and the parity of $\texttt{W}'[\texttt{Nk}-1]$.

3.6 More Details for $(5,4)$ CRC

Although the $(5,4)$ CRC has four parities, it is possible for only one parity to be used in realization of this scheme. AES can be implemented in a 32-bit structure, i.e., one column of a state is processed once in every round. In this structure, the position of `ShiftRows` must be shifted above the `SubBytes` operation. After `ShiftRows`, each column passes through the identical calculations, `SubBytes`, `MixColumns`, and `AddRoundKey`; the parity generation, or the syndrome calculation for each column, are also identical, so only one circuit is required.

4 Undetectable Errors

Even though the AES algorithm propagates the errors during encryption, the error coverage can be also analyzed mathematically. Actually, only the `MixColumns` and `SubBytes` operations cause numerous erroneous bits when a single-bit error is injected, when `ShiftRows` or `AddRoundKey` do not change the bit number of the errors. Several assumptions are made, as follows:

1. The error model is considered as Fig. 2.
2. All nonzero error block over $GF(2^{8(n+1)})$ have the same probability, where $n \in \{4, 8, 16\}$.
3. Each operation has the same error injection probability.

4.1 The Undetectable Errors in `SubBytes`

Because `SubBytes` is invertible, all errors injected into input can be detected by `InvSubBytes` and vice versa. Therefore, the *united SubBytes*, has 100 percent fault coverage. In *separated SubBytes*, both operations, the $GF(2^8)$ inversion and the affine transformation, have their own error detection. The $GF(2^8)$ inversion is also invertible, so it has 100 percent fault coverage in *hybrid SubBytes*.

In *parity-based SubBytes*, the error detection capability of the $GF(2^8)$ inversion is analyzed. According to (14), the scheme only uses XOR operations, so all the codewords are the undetectable errors in *parity-based SubBytes*. Therefore, while applying the $(17,16)$ CRC to a 128-bit data block, the number of undetectable nonzero errors is $(2^8)^{16}-1$ and the percentage of undetectable errors is $\frac{(2^8)^{16}-1}{(2^8)^{17}} \cong 0.4\%$. When the $(5,4)$ CRC is applied to a 128-bit data block, the total number of undetectable nonzero errors is $((2^8)^4-1)^4$ and the percentage is $(\frac{(2^8)^4-1}{(2^8)^5})^4 \times 100\% \cong 2.56 \times 10^{-8}\%$. Similarly, the percentage of undetectable errors for the $(9,8)$ CRC is $0.16 \times 10^{-2}\%$.

The affine transformation is detected by $(n+1,n)$ CRC. Although five erroneous bits were caused, while injecting a single-bit error, the error coverage can still be analyzed.

Theorem 1. *Given an input state $S = \{p, s_0, s_1, \ldots, s_{n-1}\}$, where parity p is $\sum_{i=0}^{n-1} s_i$, and $n \in \{4, 8, 16\}$, the output state is $T = \{t_0, t_1, \ldots, t_n\}$, where t_0 is Ap from (16), and t_{i+1}, $0 \le i \le n-1$, is obtained from (2). Introducing an error $E = \{e_0, e_1, \ldots, e_n\}$ into the state $S = \{p, s_0, s_1, \ldots, s_{n-1}\}$, the summation of the output T' will equal to zero if and only if $\sum_{i=0}^{n} e_i = 0$.*

Proof. Because n is even, the value `63` will be cancelled. Therefore, the summation of the erroneous output T' is

$$\begin{aligned}\sum_{i=0}^{n} t'_i &= Ap + e_0 + A\sum_{i=0}^{n-1}(s_i + e_{i+1}),\\ &= \underbrace{Ap + A\sum_{i=0}^{n-1} s_i}_{0} + A\sum_{i=0}^{n} e_i,\\ &= A\sum_{i=0}^{n} e_i.\end{aligned}$$

Therefore, $\sum_{i=0}^{n} t'_i$ equals to zero if and only if $A\sum_{i=0}^{n} e_i = 0$ is held. Because the matrix A is nonsingular over $GF(2)$, $A\sum_{i=0}^{n} e_i$ is zero if and only if $\sum_{i=0}^{n} e_i$ is zero. □

In the $(n+1,n)$ CRC, the nonzero errors are undetected, when the equation $\sum_{i=0}^{n} e_i = 0$ is held, i.e., errors are also the codewords. According to Theorem 1, all undetectable errors are also undetected after the affine transformation. Therefore, while applying the $(n+1,n)$ CRC to a 128-bit data block, the percentages of the undetectable errors are 0.4 percent, $0.16 \times 10^{-2}\%$, and $2.56 \times 10^{-8}\%$, respectively, for $n = 16$, $n = 8$, and $n = 4$.

4.2 The Undetectable Errors in `MixColumns`

`MixColumns` also has a diffusion property. It causes five or 11 erroneous bits while injecting a single-bit error in one column vector of the input state. However, the coefficients eliminate the diffusion of errors after summing the erroneous column vector of the output state. The `MixColumns` is shown again below, and it is supposed that each byte of the input vector is polluted by an error.

$$\begin{bmatrix} t_{i+1} \\ t_{i+2} \\ t_{i+3} \\ t_{i+4} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_i + e_i \\ s_{i+1} + e_{i+1} \\ s_{i+2} + e_{i+2} \\ s_{i+3} + e_{i+3} \end{bmatrix}. \qquad (24)$$

Then, the summation of the column vector t_{i+1} is

$$\begin{aligned}\sum_{k=0}^{3} t_{i+k+1} = {} & (02+01+01+03)(s_i + e_i) + \\ & (03+02+01+01)(s_{i+1} + e_{i+1}) + \\ & (01+03+02+01)(s_{i+2} + e_{i+2}) + \\ & (01+01+03+02)(s_{i+3} + e_{i+3}), \\ = {} & \sum_{k=0}^{3}(s_{i+k} + e_{i+k}).\end{aligned} \qquad (25)$$

The equation also holds for two or four columns vectors.

Theorem 2. *Giving an input state $S = \{p, s_0, s_1, \ldots, s_{n-1}\}$, where $p = \sum_{i=0}^{n-1} s_i$ is the checksum of the input state and $n \in \{4, 8, 16\}$. After* `MixColumns` *and the parity prediction (20), the output state is $T = \{t_0, t_1, \ldots, t_n\}$, where $t_0 = p$, and the rest is the output of* `MixColumns`*. Introducing an error $E = \{e_0, e_1, \ldots, e_n\}$ into the state $S = \{p, s_0, s_1, \ldots, s_{n-1}\}$, then the errors of the $(n+1, n)$ CRC in* `MixColumns` *are undetectable if and only if the summation $\sum_{i=0}^{n} e_i$ is zero.*

Proof. The syndrome $\sum_{i=0}^{n} t_i$ is used to check whether errors occurred or not. It is assumed that no errors occurred, if and only if the syndrome is zero. The summation of the erroneous output state is

$$\sum_{i=0}^{n} t'_i = (t_0 + e_0) + \sum_{i=1}^{n} t'_i.$$

From (25), because n is the multiple of four, the above equation is represented as

$$\begin{aligned}\sum_{i=0}^{n} t'_i &= (t_0 + e_0) + \sum_{i=1}^{n} (s_{i-1} + e_i), \\ &= \underbrace{t_0 + \sum_{i=0}^{n-1} s_i}_{0} + \sum_{i=0}^{n} e_i, \\ &= \sum_{i=0}^{n} e_i.\end{aligned}$$

Therefore, the error is undetectable if and only if $\sum_{i=0}^{n} e_i$ is zero. □

From Theorem 2, there are $((2^8)^{16} - 1)$ nonzero errors that are undetectable, when the $(17, 16)$ CRC is applied to a 128-bit data block. This result is the same as those in the affine transformation described above. Similarly, the total number of the undetectable errors for the $(9, 8)$ or $(5, 4)$ CRC is $((2^8)^4 - 1)^4$ or $((2^8)^8 - 1)^2$, respectively.

4.3 The Undetectable Errors in `ShiftRows` or `AddRoundKey`

`ShiftRows` does not change the value of the input state, and `AddRoundKey` only EXORs the input state with a round key. Therefore, the undetectable errors are the same as those analyzed in the affine transformation or `MixColumns`.

5 Detection Levels

The proposed scheme may be used in operation-level, round-level, or algorithm-level error detection. In operation-level detection, the syndrome is checked at the end of each operation. Similarly, if the syndrome is obtained at the end of each round, it is round-level detection. The implementation of operation-level error detection is easy to figure out. The syndrome is calculated at the end of each operation according to the equations derived in Section 3. However, the implementation of a round-level detection needs more ingenuity, when the `SubBytes` is protected by *united SubBytes*. The parity is generated at the end of the `SubBytes` or the beginning of the `ShiftRows`. Then, the parity directly passes through `ShiftRows`, and `MixColumns` because its value will not be changed after the two operations. Finally,

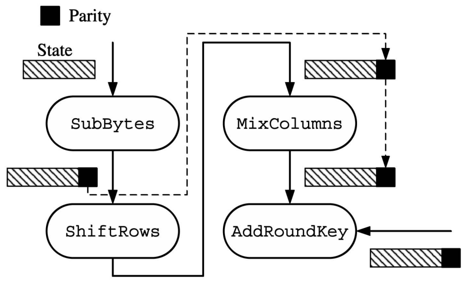

Fig. 9. The proposed scheme under round-level error detection.

the parity is EXORed with the key parity. The total path is shown in Fig. 9. Obviously, the syndrome could then be checked at the end of the round. In *hybrid SubBytes*, the structure for round-level error detection is similar to Fig. 9, but the parity is generated after the $GF(2^8)$ inversion. Because the parity of the state, in the ith round, cannot pass through the inversion of $GF(2^8)$ in $i+1$ round, the parity must be regenerated in each round. Therefore, *united-SubBytes detection* or *hybrid-SubBytes detection* cannot be implemented as algorithm-level detection.

However, each operation of *parity-based SubBytes* is protected by $(n+1, n)$ CRC, hence the parity could pass through a round. Therefore, *parity-based SubBytes* could be applied as an operation-level, round-level, or algorithm-level error detection.

6 Features and Costs

6.1 Scalability

In Section 3, it was found that the three error detections, $(n+1, n)$ CRC, where $n \in \{4, 8, 16\}$, had similar structures. The calculations of parities or syndromes were all based on Byte-EXOR (B-EXOR) operation and the length of the message was a multiple of four bytes. Therefore, the proposed approach is scalable with practical hardware design; in other words, the three CRCs can be applied to an AES implementation of an 8-bit, 32-bit, or 128-bit structure. In general, the portable devices are more probable to encounter DFA than a nonportable device. Therefore, the scalability of error scheme is good for practical purposes because 8-bit and 32-bit architectures are most commonly used in portable applications, such as cell phones, SmartCard, or RFID tag.

The approach proposed by Bertoni et al. [1] cannot be easily scaled down into the 8-bit architecture because the parity of s_i requires the information from s_{i+1} and s_{i+2}. However, this work can easily be applied to an 8-bit, 32-bit, or 128-bit AES architecture. The syndrome generation is similar to parity generation. Fig. 10 shows a block diagram of (17) and (16) for 8-bit AES architecture. While 16 bytes t_i are obtained, the syndrome u is obtained immediately, where the initial value of parity registers as a zero byte. The `ShiftRows`, `MixColumns`, or `AddRoundKey` have similar structures to Fig. 10, but the matrix transformation, A, is not required. The 32-bit or 128-bit AES can also be implemented, based on the concept in Fig. 10.

The 32-bit architecture is the most flexible structure from the point of error detection because it could use $(17, 16)$,

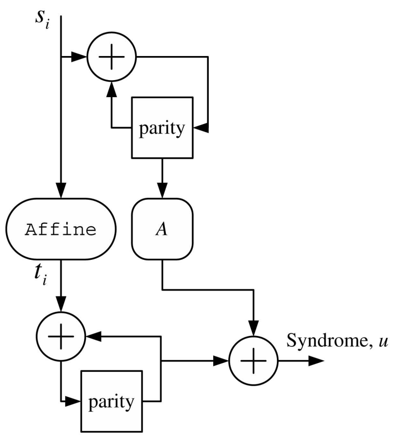

Fig. 10. The block diagram of error detection for 8-bit AES architecture.

$(9,8)$, or $(5,4)$ CRC to achieve the error detection objective. No matter which one is selected, it is possible that only a one-byte register is required to store the parities. However, the input must be a one-column vector, defined in AES; thus, (20) may be used to detect faults for a one-column calculation.

6.2 Symmetry

From Fig. 10, it can be seen that the proposed scheme is symmetric in both encryption and decryption. This has the advantage of the encryption and decryption being integrated into one chip. However, the scheme proposed by Bertoni et al. [1] is asymmetrical in `MixColumns` and `InvMixColumns`. As shown in Table 1, the output parity prediction of `InvMixColumns` is more complex than that of `MixColumns`.

6.3 Costs

While introducing proposed error detection schemes into AES, the hardware cost required by those schemes is evaluated through their computational complexity. Error detection consists of two parts—the parity and syndrome generation. Discussing the cost in parity generation first, in our proposed schemes, the parity requires only the EXOR operation. A total of $(n-1) \times \frac{16}{n}$ Byte-XORs (B-EXOR) is required to calculate the parity of the input for the proposed approach. Taking the $(5,4)$ CRC for a 128-bit data block as an example, one checksum of an input message is generated by three B-EXORs and a total of 12 B-EXORs for four parities. However, *united SubBytes* uses `InvSubBytes` to check error, so no parity generation is required. In *hybrid SubBytes*, the $(n+1,n)$ CRC is applied to the affine transformation; 15, 14, or 12 B-XORs are required to produce the parities for n of 16, 8, or 4, respectively. In the method proposed by Bertoni et al. [1], 16×7 bit-EXORs (b-EXOR) were required to obtain 16 one-bit parities for an AES state. In [7], they used the inversion operation to detect the errors; hence, no parities were paid for. However, the hardware of parity generation is minor because the parity generation is required to perform at the beginning of the parity-based detection is applied. In PbSBD, because the parity can pass through each operation along with predicting the parity, the parity generation only performs once. In USBD and HSBD, the parity must be regenerated in `SubBytes` of each round; nevertheless, only one circuit of parity generation is required when one round is implemented to achieve AES computing. In the approach of Bertoni et al. [1], the parity can also pass through the round; hence, one circuit of parity generation is required.

As regards the cost of the syndrome generation and parity prediction, it varies from operation to operation. *United SubBytes* uses the *InvSubBytes* to detect errors. In *hybrid SubBytes*, the $GF(2^8)$ inversion is used to self-check errors; the $(n+1,n)$ CRC is used to detect errors of affine transformation. According to (17), 16 B-EXORs are required to obtain the syndrome for every $(n+1,n)$ CRCs. However, the execution number of affine multiplication to predict parity, (16), depends on n; the number is one, two, or four when n is 16, 8, or 4, respectively. For *parity-based SubBytes*, the cost in affine transformation is the same as that in *hybrid SubBytes*. However, the $GF(2^8)$ inversion also uses $(n+1,n)$ CRC; according to (14), 32 B-EXORs are required (note that the $(t_{i+1}+t_{i+1}^{-1})$ in (14) is obtained from a table, not requiring EXOR calculation). In `ShiftRows` and `MixColumns`, no prediction functions are necessary and the syndrome is obtained by summing all output byte and the parity. Therefore, in the two operations, 16 B-EXORs are required. In `AddRoundKey`, the one, two, or four one-byte parities of a round key are involved in the parity prediction, requiring extra B-EXORs to be paid for. The results summarized in Table 1 are the cost of the operation-level detection, i.e., the error detection is at the end of every operation. If round-level or algorithm-level are chose, only

TABLE 1
The Cost of Syndrome Generation and Parity Prediction in Each AES Operation in the Operation-Level Detection

		Ours (n =16, 8 or 4)	Bertoni[1], [3]	Karri[7]
Bit number of parity		8/16/32 bits	16 bits	0 bit
`SubBytes`	USB	`InvSubBytes`	—	InvSubBytes
	HSB	the $GF(2^8)$ inversion, 16×8 b-EXORs, and 1/2/4 AMs	—	
	PbSB	32×8 b-EXORs, 16×8 b-EXORs, and 1/2/4 AMs	$m \times 256$ bits memory, $m \times 9$ b-EXORs, and comparison circuits.	
`ShiftRows`		16×8 b-EXORs	bit shift+16×8 b-EXORs	`InvShiftRows`
`MixColumns`	Cost in EN	16×8 b-EXORs	$16 \times 8 + 16 \times 4$ b-EXORs	`InvMixColumns`
	Cost in DE	16×8 b-EXORs	More complicated than in EN	`MixColumns`
`AddRoundKey`		$(16 + 1/2/4) \times 8$ b-EXORs	$16 \times 8 + 16$ b-EXORs	`AddRoundKey`

B-EXOR = 8 b-EXORs, b-EXOR = bit EXOR operation, EN = encryption, DE = decryption, and AM = affine multiplication.

TABLE 2
The Possible Combinations of Our Proposed Schemes

USBD	HSBD	PbSBD
(17,16)	(17,16)	(17,16)
(9,8)	(9,8)	(9,8)
(5,4)	(5,4)	(5,4)

the cost of parity prediction is required in every operation and the cost of syndrome generation is only paid at the end of each round or of the AES algorithm, respectively.

The costs of Bertoni et al.'s [1] approach are also varied in each operation. The SubBytes requires extra m 256-byte memory spaces to predict the parity, where m is dependent on the implementation of the AES. Taking an AES implemented in a 32-bit structure as an example, four bytes are calculated in parallel, thus four tables are required. The size of a table with error detection, in [1], is a double of that in AES, so a total of 512 bytes is for one table, i.e., 256 extra bytes are caused for one table. The 256 extra bytes are constants with odd parity, e.g., 00000000 1; therefore, one comparison circuit or syndrome generation circuit is required to detect the error. This detection method has been modified by Bertoni et al. [3] and the extra memory size is reduced from $m \times 256$ bytes to $m \times 256$ bits. Additionally, $m \times 9$ b-EXORs are introduced. The error detection of one byte, appended with one-bit parity, requires eight b-EXORs (bit EXOR operation) or a total of 16×8 b-EXORs for a 128-bit data block. However, Bertoni et al.'s scheme must predict the output parity in MixColumns, therefore, the extra calculations of 16×4 b-EXORs are required in the encryption process. In decryption, the error-detection hardware for InvMixColumns is more complicated than in encryption. Because the prediction of InvMixColumn is not derived in [1], the cost is not specified in Table 1. The costs of Karri et al.'s scheme required the inversion of each operation and it was also time-consuming. The operations in the key expansion are similar to the four major operations of AES; thus, the detailed comparisons of the key expansion are not discussed. Although most operations require 16 B-EXORs to compute the syndrome, it is possible to achieve the computation with less B-EXORs.

7 Error Detection Capability

In Karri et al. [7], because the four operations of AES are bijective, their error detection capability is very high. If it is assumed that only one 128-bit error occurs during encryption or decryption, then all nonzero error patterns can be detected in the operation-level, round-level, or algorithm-level detection. In Bertoni et al. [1], they used the parity-based technique and the undetectable errors do exist. Bertoni et al. [1] did a lot of tests to obtain the results about error detection capability and the results will be compared to ours in Fig. 14.

All simulations and statements of our proposed schemes, addressed here, are also under the three assumptions given in Section 4. Three architectures, USBD, HSBD, and PbSBD, were proposed herein; each architecture has three types of CRC, $(17,16)$, $(9,8)$, and $(5,4)$ CRCs, as shown in Table 2.

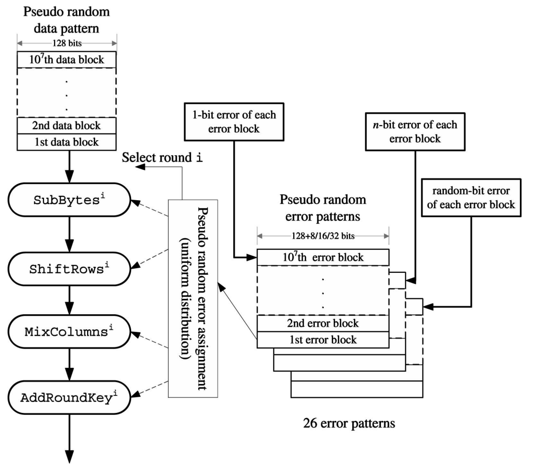
Pseudo random data pattern
128 bits
10^7th data block
2nd data block
1st data block
Select round i
SubBytes^i
ShiftRows^i
MixColumns^i
AddRoundKey^i
Pseudo random error assignment (uniform distribution)
1-bit error of each error block
n-bit error of each error block
random-bit error of each error block
Pseudo random error patterns
128+8/16/32 bits
10^7th error block
2nd error block
1st error block
26 error patterns

Fig. 11. The simulation model. Each data block has 64 ones and the position of ones uniformly distributed in a data block. The error bits uniformly distribute in an error block. The assignment of error blocks uniform distributes in both rounds and operations.

Thus, nine methods were simulated. In PbBSD, the data procedure is thoroughly protected by the $(n+1, n)$ CRC; thus, each operation has undetectable errors. However, in USBD, the fault coverage in SubBytes is 100 percent, so the amount of overall undetectable errors is 80 percent of that in USBD. Similarly, in HSBD, the amount is reduced to 75 percent of that in USBD.

The simulation model is shown in Fig. 11. Each method is simulated by 26 tests distinguished by the bit number of the injected errors. The last test in Fig. 12, Fig. 13, and Fig. 14, labeled as random, used error patterns with random erroneous bit number. Each error pattern has 10^7 blocks and the bit length of every block is $136(128+8)$, $144(2 \times (64+8))$, or $160(4 \times (32+8))$, respectively, for the $(17,16)$, $(9,8)$, or $(5,4)$ CRC. The all-one error block was considered as a totally different state; hence, the maximum number of erroneous bits was 135, 143, or 159 in a random test. Each test used one data pattern of 10^7 data blocks, and every

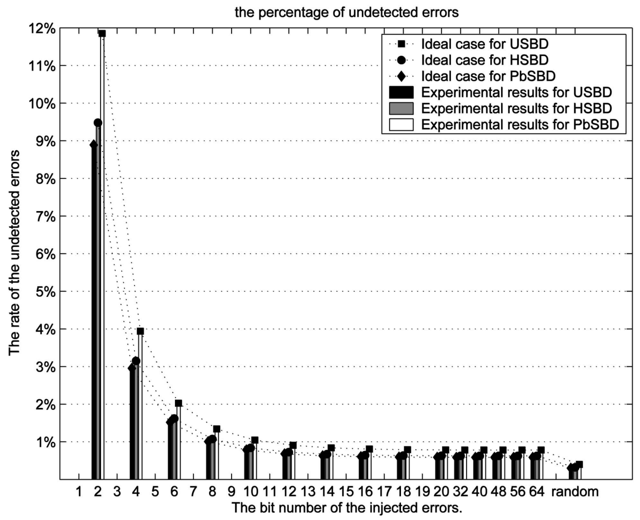

Fig. 12. Percentage of undetectable errors of the $(17,16)$ CRC over $GF(2^8)$.

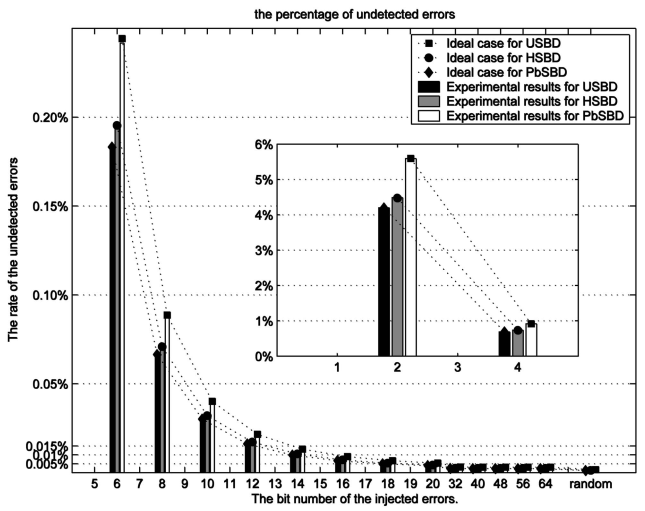

Fig. 13. Percentage of undetectable errors of the $(9,8)$ CRC over $GF(2^8)$. Their percentage is 4.14 percent for 2-bit errors and 0.067 percent for 4-bit errors.

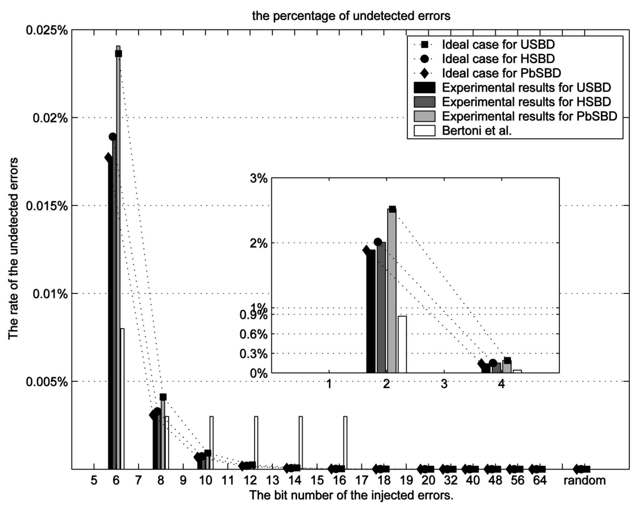

Fig. 14. Percentage of undectable errors of the $(5,4)$ CRC over $GF(2^8)$. The percentage is 1.8 percent for 2-bit errors and 0.13 percent for 4-bit errors.

block has 64-bit ones of normal distribution. The erroneous rounds and erroneous operation were also randomly chosen.

As seen in Fig. 12, all the simulated odd-bit errors were detected. The percentage of the undetectable errors dropped dramatically as the erroneous bit number increased. When the number of erroneous bits was greater than eight, the percentage was below 1 percent and stable. The test using random erroneous bits is about 0.3 percent and it was close to the theoretic value obtained in Section 4, 0.4 percent. Obviously, all the experimental results followed the curves of ideal values.

The same data patterns used in the above tests were also used for the $(9,8)$ CRC and the $(5,4)$ CRC; all test conditions, except for the error patterns, were identical to those used to test the $(17,16)$ CRC. The $(9,8)$ CRC generated two parities for a 128-bit data block. Because the values in the two tests, 2-bit and 4-bit erroneous bits, are too large, they were dependently shown in Fig. 13. All odd-bit errors were also detected. The percentage also dropped dramatically when the erroneous bits increased, as shown in Fig. 13. For the random test, the percentage is about $0.14 \times 10^{-2}\%$, very close to the theoretical value of $0.16 \times 10^{-2}\%$.

In Fig. 14, the results of the $(5,4)$ CRC and Bertoni et al. [1] are shown. Obviously, this percentage is very small in contrast to the $(17,16)$ CRC or the $(9,8)$ CRC. When the number of erroneous bits was larger than 16, the percentages of undetectable errors dropped to zero. The percentage in the random test was 0 percent, very close to the theoretic value of $2.56 \times 10^{-8}\%$. Of course, all odd-bit errors could be detected.

Fig. 14 also shows the results in Bertoni et al. [1]. The test models of Bertoni et al. [1] are different from ours. They have injected multiple bit errors (between 2 to 16) at the beginning of the round. From Fig. 14, their scheme has better error detection than ours, when the errors are between 2 and 6, and the cases of 8-bit errors are close. When the number of erroneous bits is above 10, the performance of the proposed scheme is better than that of Bertoni et al. [1].

8 Conclusions

This work has proposed a simple, symmetric, and high-fault-coverage error detection scheme for AES. Although the erroneous bits are diffused in AES, this work used the linear behavior of each operation in AES to design a detection scheme. This scheme only uses an $(n+1,n)$ CRC to detect the errors, where $n \in \{4,8,16\}$, and the parity of the output of each operation is predicted in a simple fashion. Even though the number of parities is two or four, respectively, for $n = 8$ or $n = 4$, it is possible to use only one 8-bit register for storing the parities during hardware implementation. This error detection may also be used in encryption-only or decryption-only designs. Because of the symmetry of the proposed detection scheme, the encryption and decryption circuit can share the same error detection hardware. The proposed schemes can be applied in the implementation of AES against differential fault attacks and can be easily implemented in a variety of structures, such as 8-bit, 32-bit, or 128-bit structures.

Acknowledgments

This work was financially supported by the Program for Promoting University Academic Excellence under Grant no. EX-91-E-FA06-4-4.

References

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

[3] G. Bertoni, L. Brevegelieri, I. Koren, and P. Maistri, "An Efficient Hardware-based Fault Diagnosis Scheme for AES: Performances and Cost," *Proc. 19th IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems,* pp. 130-138, Oct. 2004.

[4] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," *Advances in Cryptology—Proc. CRYPTO '97,* pp. 513-525, 1997.

[5] P. Dusart, G. Letourneux, and O. Vivolo, "Differential Fault Analysis on A.E.S," *Applied Cryptography and Network Security,* pp. 293-306, 2003.

[6] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong Authentication for RFID Systems Using the AES Algorithm," *Proc. Cryptographic Hardware and Embedded Systems (CHES '04),* pp. 357-370, 2004.

[7] R. Karri, K. Wu, P. Mishra, and 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, pp. 1509-1517, Dec. 2002.

[8] R. Karri, G. Kuznetsov, and M. Goessel, "Parity-Based Concurrent Error Detection of Subsititution-Permutation Network Block Ciphers," *Proc. Cryptographic Hardware and Embedded Systems (CHES '03),* pp. 113-124. 2003.

[9] S. Mangard, M. Aigner, and S. Dominikus, "A Highly Regular and Scalable AES Hardware Architecture," *IEEE Trans. Computers,* vol. 52, no. 4, pp. 483-491, Apr. 2003.

[10] US Nat'l Inst. of Standards and Technology, "Federal Information Processing Standards Publication 197—Announcing the ADVANCED ENCRYPTION STANDARD (AES)," 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[11] G. Piret and J.J. Quisquater, "A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD," *Proc. Cryptographic Hardware and Embedded Systems (CHES '03),* pp. 77-88, 2003.

[12] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," *AES Algorithm Submission,* Sept. 1999.

[13] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," *Proc. Advances in Cryptology (ASIACRYPT '01),* pp. 171-184, 2001.

[14] K. Wu, R. Karri, G. Kuznetsov, and M. Goessel, "Low Cost Concurrent Error Detection for the Advanced Encryption Standard," *Proc. Int'l Test Conf. (ITC '04),* pp. 1242-1248, 2004.

Chi-Hsu Yen received the BS degree in electrical engineering from National Central University, Jhongli, Taiwan, in 1995 and the MS degree in electrical engineering from Tamkang University, Tamsui, Taiwan, in 1997. He received the PhD degree from the Department of Electrical and Control Engineering of National Chiao-Tung University, Hsinchu, Taiwan, in 2005. His research interests include cryptographic algorithms and error-control coding.

Bing-Fei Wu (S'89-M'92-SM'02) received the BS and MS degrees in control engineering from National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 1981 and 1983, respectively, and the PhD degree in electrical engineering from the University of Southern California, Los Angeles, in 1992. Since 1992, he has been with the Department of Electrical Engineering and Control Engineering, where he is currently a professor. He has been involved in the research of intelligent transportation systems for many years and is leading a team to develop the first Smart Car with autonomous driving and active safety system in Taiwan. His current research interests include vision-based intelligent vehicle control, multimedia signal analysis, embedded systems, and chip design. He is a senior member of the IEEE. He founded and served as the chair of the IEEE Systems, Man, and Cybernetics Society Taipei Chapter in Taiwan, 2003. He was the director of the Research Group of Control Technology of Consumer Electronics in the Automatic Control Section of the National Science Council (NSC), Taiwan, from 1999 to 2000. As an active industry consultant, he was also involved in the chip design and applications of the flash memory controller and 3C consumer electronics in multimedia systems. The research has been honored by the Ministry of Education as the Best Industry-Academics Cooperation Research Award in 2003. He received the Distinguished Engineering Professor Award from the Chinese Institute of Engineers in 2002, the Outstanding Information Technology Elite Award from the Taiwan Government in 2003, the Golden Linux Award in 2004, the Outstanding Research Award in 2004 from NCTU, the Research Awards from NSC in the years of 1992, 1994, 1996-2000, the Golden Acer Dragon Thesis Award sponsored by the Acer Foundation in 1998 and 2003, respectively, the First Prize Award of the We Win (Win by Entrepreneurship and Work with Innovation & Networking) Competition hosted by Industrial Bank of Taiwan in 2003, and the Silver Award of Technology Innovation Competition sponsored by the Advantech Foundation in 2003.