A Hybrid Fault Tolerant Approach for AES

Chang N. Zhang[1], Qian Yu[1], and Xiao Wei Liu[2]
(Corresponding author: Qian Yu)

Department of Computer Science, Univeristy of Regina, Regina, SK, Canada[1]
IBM China Development Lab, Shanghai, China[2]
(Email: yu209@cs.uregina.ca)

(Received Apr. 29, 2011; revised and accepted Feb. 22, 2012)

Abstract

In this paper, a lightweight hybrid fault tolerant approach for AES, which is based on the integration of the algorithm based fault tolerant (ABFT) technique and the fault tolerant technique for s-box byte substitution operation is proposed. Two versions of scheme are presented to satisfy different application requirements. The first general version scheme can detect single error for the whole AES process with high efficiency. Another run-time version scheme is used to immediately terminate the error round with no time delay and no computation wasted on the rest rounds for propagating errors. Utilizing the ready-made arithmetic units in AES, single error can be detected by the sender and prevent the misdirected information from sending out. The results of the hardware FPGA implementation and simulation show that the proposed scheme can be integrated both on software and hardware without making many changes to the original AES implementation.

Keywords: ABFT, advanced encryption standard, algorithm based fault tolerant, error detection, fault tolerance

1 Introduction

Advanced Encryption Standard (AES) is a replacement for triple DES (3DES which not only has comparable security strength, but also achieve significant efficiency improvement for implementation on software or hardware.

With the wide-spread of the AES applications, differential types of faults may occur from time to time. Several efforts were devoted into fault tolerance of the transformations and rounds in AES algorithm. Guido Bertoni *et al.* presented a fault model for AES and analyzed the behavior of the AES algorithm in the presence of faults [1]. They also proposed a fault detection technique for a hardware implementation of the AES algorithm which is based on the parity codes [2]. Moreover, they developed an analytical error model for the parity-based EDC for the AES encryption algorithm and are capable of locating single-bit transient and permanent faults [3, 4]. Later, the same group further described the complete error model extended to include the Key Schedule (KS) part and presented the results of the software simulations of the model [5]. L. Berveglieri *et al.* proposed an extension to an existing AES architecture to provide error detection and fault tolerance [7]. Kaijie Wu *et al.* presented a low-cost concurrent checking method for the AES encryption algorithm by using parity checking which can detect faults during normal operation and deliberately injected faults [19]. Mark Karpovsky *et al.* presented a method of protecting a hardware implementation of the AES against a side-channel attack known as Differential Fault Analysis attack [13]. Chih-Hsu Yen *et al.* proposed several error-detection schemes for AES which are based on the (n+1, n) cyclic redundancy check over GF (2^8) [20]. Luca Breveglieri *et al.* presented an operation-centered approach to the incorporation of fault detection into cryptographic device implementation through the use of Error Detection Codes [8]. Ramesh Karri *et al.* presented a fault-tolerant architecture for symmetric block ciphers which is based on a hardware pipeline for encryption and decryption [12]. P. Maistri *et al.* presented the results of a validation campaign on an AES core protected with some error detection mechanisms [15]. Mojtaba Valinataj *et al.* combined and reinforced the parity prediction scheme with a partially distributed TMR to achieve more reliability against multiple simultaneous errors [18].

Other efforts focus on relevant fault detection field. L. Berveglieri *et al*. presented suggestions for providing fault detection capabilities in recent block ciphers and came to the conclusion that the detection capability of any code depends on the type of the code, the frequency of checkpoints and the level of redundancy [6]. Ramesh Karri *et al*. presented a technique to concurrently detect errors in block ciphers as well as a new encoding strategy [9].

As a summary, the parity bit check coding technique has been introduced and widely applied to the basic operations of AES. For this technique, the parity bit needs to be generated and checked for every individual AES operation which brings in considerable time and hardware overhead. The algorithm based tolerant (ABFT) technique is a general concept for designing efficient fault tolerant schemes based on structures of the algorithms [10, 14, 16]. The ABFT makes use of the computational nature of the targeted algorithm and poses a conceptual way to better create a fault tolerant version by altering the algorithm computation so that its output contains extra information for error detection and correction. It has relatively low overhead and no additional arithmetical logic unit is required.

However, there is a limitation that the ABFT approach cannot be applied to AES directly. The limitation is that it applies to certain type of algorithm usually only involved linear and/or logical XOR operations. Due to the security nature of the AES algorithm, some non-linear operation such as s-box byte substitution operation is required. For this reason, the ABFT can not apply to AES directly. In this paper, we proposed a hybrid approach which integrated the ABFT approach with other fault tolerant approach such that it can apply to AES with lighter overheads.

This paper is organized as follows. Section 2 briefly reviews the AES algorithm and introduces the notations for further discussion. Section 3 presents the proposed hybrid fault tolerant approach for AES. Section 4 shows the implementation, simulation results and comparison. The last section concludes this paper.

2 The AES Algorithm and Relevant Notations

For the sake of simplicity, we adopt the AES parameters of 128-bit key size, 128-bit plaintext block size, 10 rounds, 128-bit round key size, and 44-word expanded key size, which are the most commonly-used parameter set. Due to the similarity between encryption and decryption parts, just encryption is introduced as an illustration of our schemes. There is one initial round followed by 9 four-step rounds and ended by a tenth final round.

2.1 The Initial Round

The first initial round only performs the AddRoundKey operation. "m" stands for the total ordinal number of each round. "e" is the results for every round. "P" and "K" are defined as plaintext and round key respectively. For the initial round 0, we have $m=0$, the plaintext state P and round 0th key state K^0 are represented as,

$$P=\begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$$

$$K^0=\begin{bmatrix} K^0_{00} & K^0_{01} & K^0_{02} & K^0_{03} \\ K^0_{10} & K^0_{11} & K^0_{12} & K^0_{13} \\ K^0_{20} & K^0_{21} & K^0_{22} & K^0_{23} \\ K^0_{30} & K^0_{31} & K^0_{32} & K^0_{33} \end{bmatrix}$$

According to the operation, the equation for round 0 can be derived as follows:

$$\begin{bmatrix} e^0_{0j} \\ e^0_{1j} \\ e^0_{2j} \\ e^0_{3j} \end{bmatrix} = \begin{bmatrix} P_{0j} \\ P_{1j} \\ P_{2j} \\ P_{3j} \end{bmatrix} \oplus \begin{bmatrix} K^0_{0j} \\ K^0_{1j} \\ K^0_{2j} \\ K^0_{3j} \end{bmatrix}$$; where $0 \le j \le 3$ and e^0_{ij} is the result for round 0, $0 \le i \le 3$.

2.2 The 9 Rounds

The 9 rounds have the same sub-operations for each round, which are SubBytes, ShiftRows, MixColumns and AddRoundKey according to the order. For these 9 rounds, $1 \le m \le 9$, "a" is the input of round m, "b" "r" and "c" stands for the intermediate results for SubBytes, ShiftRows and MixColumns operations respectively, and e^m also represents the results of round m.

Input of Round m:

$a^m_{ij} = e^{m-1}_{ij}$, where $1 \le m \le 9$, $0 \le i \le 3$, $0 \le j \le 3$;

SubBytes Operation:

$b^m_{ij} = S[a^m_{ij}]$, where S stands for the substitution by the S-box and $1 \le m \le 9$, $0 \le i \le 3$, $0 \le j \le 3$.

ShiftRows Operation:

$$\begin{bmatrix} r^m_{0j} \\ r^0_{1j} \\ r^0_{2j} \\ r^0_{3j} \end{bmatrix} = \begin{bmatrix} b^m_{0j} \\ b^m_{1,j+1} \\ b^m_{2,j+2} \\ b^m_{3,j+3} \end{bmatrix}$$, where $0 \le j \le 3$ and $1 \le m \le 9$.

MixColumns Operation:

$$\begin{bmatrix} c^m_{0j} \\ c^m_{1j} \\ c^m_{2j} \\ c^m_{3j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} r^m_{0j} \\ r^0_{1j} \\ r^0_{2j} \\ r^0_{3j} \end{bmatrix}$$ in GF (2^8), where $0 \le j \le 3$ and $1 \le m \le 9$.

AddRoundKey Operation:

$$\begin{bmatrix} e^m_{0j} \\ e^m_{1j} \\ e^m_{2j} \\ e^m_{3j} \end{bmatrix} = \begin{bmatrix} c^m_{0j} \\ c^m_{1j} \\ c^m_{2j} \\ c^m_{3j} \end{bmatrix} \oplus \begin{bmatrix} K^m_{0j} \\ K^m_{1j} \\ K^m_{2j} \\ K^m_{3j} \end{bmatrix}$$, where $0 \le j \le 3$ and $1 \le m \le 9$.

Note that $c^m_{ij} \oplus K^m_{ij} = c^m_{ij} + K^m_{ij}$ in GF (2^8).

As a conclusion, the equation for each round can be represented as:

$$E_j^m = \begin{bmatrix} e_{0j}^m \\ e_{1j}^m \\ e_{2j}^m \\ e_{3j}^m \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} S[e_{0j}^m] \\ S[e_{1,j+1}^m] \\ S[^m_{2,j+2}] \\ S[e_{3,j+3}^m] \end{bmatrix} + \begin{bmatrix} K_{0j}^m \\ K_{1j}^m \\ K_{2j}^m \\ K_{3j}^m \end{bmatrix}$$

in GF (2^8), (1)

where $0 \le j \le 3$ and $1 \le m \le 9$.

2.3 The Final Round

The final round has no MixColumns operation, and the three operations and their execution order is the same as the previous 9 rounds. In this 10th round, we have $m = 10$ and the following equation:

$$\begin{bmatrix} e_{0j}^{10} \\ e_{1j}^{10} \\ e_{2j}^{10} \\ e_{3j}^{10} \end{bmatrix} = \begin{bmatrix} S[e_{0j}^9] \\ S[e_{1,j+1}^9] \\ S[^9_{2,j+2}] \\ S[e_{3,j+3}^9] \end{bmatrix} + \begin{bmatrix} K_{0j}^{10} \\ K_{1j}^{10} \\ K_{2j}^{10} \\ K_{3j}^{10} \end{bmatrix}$$ in GF (2^8), where $0 \le j \le 3$.

3 The Hybrid Fault Tolerant Approach for AES

For round 0 to 10, let $K = \left[K_{ij}\right] = [\sum_{m=0}^{10} K_{ij}^m]$, and $E = \left[e_{ij}\right] = \left[\sum_{m=0}^{10} e_{ij}^m\right]$, both in GF ($2^8$), where $0 \le i, j \le 3$, we have:

$$E_j = \begin{bmatrix} e_{0j} \\ e_{1j} \\ e_{2j} \\ e_{3j} \end{bmatrix} = \begin{bmatrix} P_{0j} \\ P_{1j} \\ P_{2j} \\ P_{3j} \end{bmatrix} + \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} \sum_{m=o}^{8} S\left[e_{0j}^m\right] \\ \sum_{m=0}^{8} S\left[e_{1,j+1}^m\right] \\ \sum_{m=0}^{8} S\left[e_{2,j+2}^m\right] \\ \sum_{m=0}^{8} S\left[e_{3,j+3}^m\right] \end{bmatrix} + \begin{bmatrix} S[e_{0j}^9] \\ S[e_{1,j+1}^9] \\ S[^9_{2,j+2}] \\ S[e_{3,j+3}^9] \end{bmatrix} + \begin{bmatrix} K_{0j} \\ K_{1j} \\ K_{2j} \\ K_{3j} \end{bmatrix}$$

in GF (2^8), where $0 \leqq j \leqq 3$. (2)

Suppose that only a single error may occur during the whole AES process. First we assume that the s-box SubByte operation $b_{ij}^m = S[a_{ij}^m]$ is implemented by look-up table (ROM). By applying Hamming Code to the tables any single error can be detected and corrected. Second, we assume that the SubByte operation is implemented by a circuit. By applying majority voting technique or more efficient fault tolerant technique is GF (2^8) [18], any single error can be detected or corrected.

As a summary, if s-box SubByte operation can be correctly performed and stored then due to the linear computational nature of the Equation (2), the algorithm based fault tolerant (ABFT) technique can be applied to pre-compute some known parameters as check-sums. By storing certain intermediate results, this equation can be used to detect error. We propose two versions of the error detection scheme. The general version can detect an error for the whole total 11 rounds AES process. Another run-time version can detect error and immediately stop the round in which the error exists so that it can prevent the error from propagating to following rounds.

3.1 General Version of the Scheme

Firstly, the plaintext state and the round key state have to be XORed together,

$$PK = \left[PK_{ij}\right], \tag{3}$$

where $PK_{ij} = \sum_{m=0}^{10} K_{ij}^m \oplus P_{ij}$, and $0 \le i, j \le 3$;

Secondly, while the rounds are going, two intermediate results of each rounds need to be stored, which are the final results of each round and the substitution results after SubBytes operation of each round. And then they are pre-computed as the following.

$E = \left[e_{ij}\right]$, where $e_{ij} = \sum_{m=0}^{10} e_{ij}^m$, and $0 \le i, j \le 3$;

$$S_{0j} = \sum_{m=0}^{8} S\left[e_{0j}^m\right],\ S_{1j} = \sum_{m=0}^{8} S\left[e_{1,j+1}^m\right],$$

$$S_{2j} = \sum_{m=0}^{8} S\left[e_{2,j+2}^m\right],\ S_{3j} = \sum_{m=0}^{8} S\left[e_{3,j+3}^m\right],$$

where $0 \le j \le 3$.

Thirdly, according to above notations and the Equation (2), we can obtain the following equation:

$$E' = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} S_{0j} \\ S_{1j} \\ S_{2j} \\ S_{3j} \end{bmatrix} + \begin{bmatrix} S[e_{0j}^9] \\ S[e_{1,j+1}^9] \\ S[^9_{2,j+2}] \\ S[e_{3,j+3}^9] \end{bmatrix} + \begin{bmatrix} PK_{0j} \\ PK_{1j} \\ PK_{2j} \\ PK_{3j} \end{bmatrix}$$

in GF (2^8) where $0 \le j \le 3$. (4)

At last, in order to know whether error occurs, the only work is to compare the results of above equation E' with the stored values E. If they equal then there is no error, otherwise, single error occurs in some round. By knowing the existence of errors, the sender can be informed of the false encryption result at once. In that case, error can be blocked. The general version of concurrent error detection scheme is shown in Figure 1.

If we count one round with four basic operations as a time unit, the proposed general version ABFT scheme requires only the final two basic operations, MixColumns and AddRoundKey, to perform multiplication and addition. In that case, the overhead is about 1/20 of the total AES processing time. If the AES is implemented by pipeline, and each pipeline unit performs one of the four basic operations, then the proposed ABFT scheme only affects the pipeline latency and needs some extra storage for the intermediate results, but still maintain the same throughput.

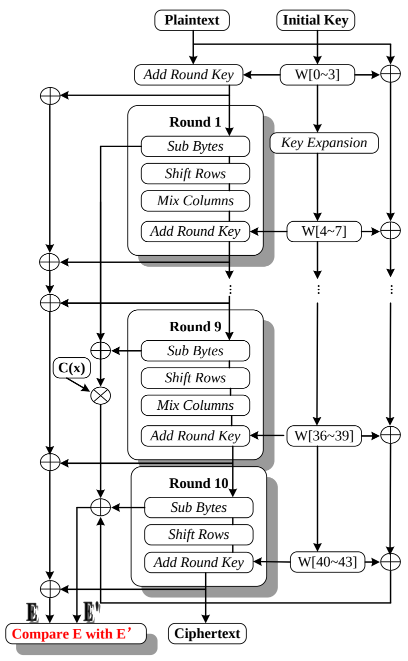

Figure 1: The general version of the scheme for AES

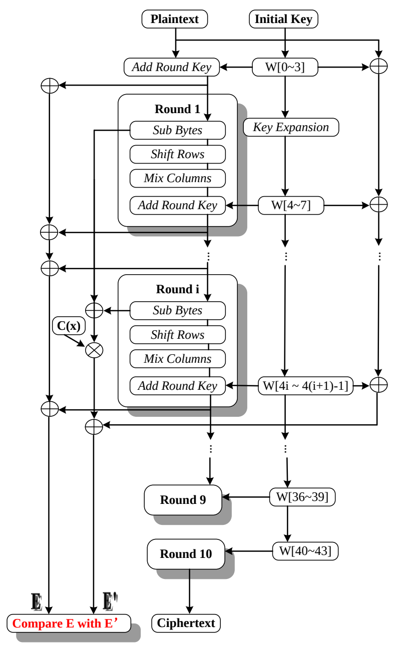

Figure 2: The run-time version of the scheme for AES

3.2 The Run-Time Concurrent Error Detection Scheme

The goal of the run-time error detection version is to find an error and immediately stop the process. Similar to the previous discussion, we can pre-compute the PK according to Equation (3). Then the intermediate results of SubBytes and the results of each round are stored and organized in the same way as the general version, however, the computational method varies. The flow chart of the run-time error detection version is shown in Figure 2.

Let “h” represent the number of round which has just finished and use the following notation:

$$E^h = \left[e_{ij}^h\right], \text{ where } e_{ij}^h = \sum_{m=0}^{h} e_{ij}^m \text{ , and } 0 \le i, j \le 3; \qquad (5)$$

$$S_{0j}^h = \sum_{m=0}^{h} S\left[e_{0j}^m\right], S_{1j}^h = \sum_{m=0}^{h} S\left[e_{1,j+1}^m\right], S_{2j}^h = \sum_{m=0}^{h} S\left[e_{2,j+2}^m\right]$$

$$, S_{3j}^h = \sum_{m=0}^{h} S\left[e_{3,j+3}^m\right], \text{ where } 0 \le j \le 3;$$

$$PK_{ij}^h = \sum_{m=0}^{h} K_{ij}^m \oplus P_{ij}, \text{ where } 0 \le i, j \le 3$$

Also, there is some modification based on Equation (1), that is,

$$E_h^{'} = \begin{bmatrix} e_{0j}^h \\ e_{1j}^h \\ e_{2j}^h \\ e_{3j}^h \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} S_{0j}^h \\ S_{1j}^h \\ S_{2j}^h \\ S_{3j}^h \end{bmatrix} + \begin{bmatrix} PK_{0j}^h \\ PK_{1j}^h \\ PK_{2j}^h \\ PK_{3j}^h \end{bmatrix} \text{ in}$$

GF (2^8), where $0 \le j \le 3$. (6)

By using this equation, we can detect the errors in each round by comparing the value E^h given by Equation (5) with the value $E_h^{'}$ of the Equation (6), $h = 1, 2, \cdots, 9$. An error exists when they do not equal. If the error happened in round 10 (h=10), we still can use the check Equation (4) indicated in the general version to compute the check value. In order to find errors in real-time during the round, check scheme has to be performed at the end of every round.

Using the run-time concurrent error detection scheme, error can be found before the whole AES process finishes. Once error occurs in certain round, there is no need to do the rest rounds. By performing the run-time check, the work and time spent on computing useless information can be saved.

As defined above, one round time with four basic operations is considered as a time unit. So the proposed run-time version ABFT scheme requires the last two basic operations to do the multiplication and addition in GF (2^8) for error detection. Hence, the overhead is about 1/2 of the total AES processing time for each round. In this way, if sub-operations are implemented by pipeline units, only pipeline latency is affected.

4 Implementation, Simulation and Comparison

As a symmetric encryption algorithm standard, AES has become one of the most important crypto-algorithms implemented on a variety of platforms, such as Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC). Among them, there are two most basic and commonly adopted architectures, which are rolling architecture and unrolling architecture [17]. The rolling architecture uses a feedback structure where the data are iteratively transformed by the round functions. This approach has the advantage of small area but the disadvantage of a low throughput. The unrolling architecture pipelines the eleven rounds and inserts registers between every two rounds. This kind of architecture achieves a high throughput, but compromises the area that is approximately 10 times larger than the rolling architecture.

For each round of the AES encryption and decryption operations, a different round key is required. In general, two methods exist to generate the round keys for the eleven rounds [11]. The first way is to pre-compute the round keys and store them into a register or memory for all the incoming plain texts in one session. However, this register/ memory based method requires a large memory or register for key storage. Another way is to generate the round keys is in an on-the-fly fashion, which allows the key expansion scheduling running concurrently with the data encryption/ decryption rounds even if the initial key is changed.

The proposed general version of the algorithm based concurrent error detection AES scheme has been implemented according to the rolling architecture and shared memory is chosen as the way to generate round keys.

The Xilinx SpartanIII XC3S400 FPGA device is used to prototype the proposed scheme. Simulation is done by Modelsim PE version. Xilinx ISE synthesizes and implements the design. Very-High-Speed Integrated Circuit Hardware Description Language (VHDL) is chosen as the description language and top-level source type in Xilinx ISE.

The ABFT AES architecture is partitioned into four different modules performing distinct functions, each of which synchronously cooperates with other modules by using linked signals. Assembling these components together, a ciphertext state and an error detection signal are obtained after each encryption process, which feed back the encrypted ciphertext and error detection result.

The program control module takes charge of the procedure and sends out time-sequential commands to the AES core module. The AES core is responsible for all the sub-operations in the rolling architecture. These sub-operations include add round key, substitute bytes, shift rows, mix columns, error detection computation and result state comparisons. This module acts as an intermediary between the program control module, S-box module, and key-ram module. The S-box module is used for S-box table lookup for SubByte operation. It is a 16*16 ROM and each element is an 8-bit data. The key-ram module stores the 44 word round keys for the 11 rounds. In this implementation, round keys are pre-generated before the encryption and are stored in the key RAM in advance. A 16-bit RAM bus is used to transfer the two bytes round keys to AES core module.

Table 1 gives the specification of the proposed scheme generated by the Xilinx ISE. The comparison is performed between the original AES encryption and the proposed scheme.

Table 1: Comparison between the original AES and the proposed scheme

Logic Utilization	**Original**	**ABFT AES**	**Overhead**
# of Slices	943	1254	32.9%
# of Slice Flip Flops	289	433	49.8%
# of 4 Input LUTs	1802	2376	31.9%
Clock Period (ns) (Clock Frequency MHz)	17.494 (57.162)	17.760 (56.306)	1.52%

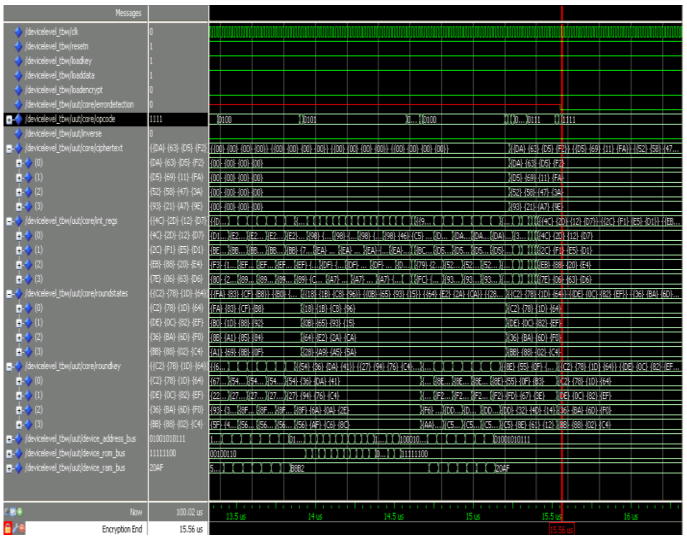
Figure 3: Simulation result of ABFT AES scheme

We use Modelsim PE edition to simulate the proposed scheme. Figure 3 shows the simulated wave graph of our proposed scheme. The implementation details and input parameters are set in the following way: the clock frequency is 50MHz with no particular constraint specification. The plaintext is chosen to be “00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff” (in hexadecimal), and the initial key is “f6 cc 34 cd c5 55 c5 41 82 54 26 02 03 ad 3e cd”. In this graph, the left column lists out the input, output and intermediate parameters in our implementation. Next to this column is the detailed data of each parameter. And in the right most view, you can find the variation of every signal in the process of ABFT AES. The output signal (encrypted ciphertext) turns out to be what we expected, which is “da d5 52 93 63 69 58 21 d5 11 47 a7 f2 fa 3a 9e”. Essentially the point we want to show in this figure is that the logic of our design has been verified on FPGA and accordingly, the waves prove the proposed scheme can achieve AES encryption with fault tolerance in a parallel way.

Since there is no united way to evaluate various implementations of AES algorithm across various platforms, and since every group employs different technology libraries, chooses different tools and even sets up different constrained parameters to test their designs, it is not quite comparable between these schemes merely because of their presented implementation results.

The proposed scheme has a reasonable hardware overhead compared to the existing schemes. Furthermore, the relative overhead will be much less if the unrolling architecture is implemented.

Nowadays, the parity bit check coding technique has been widely applied to the basic operations of AES for error detection. The parity bit needs to be generated and checked for every individual AES operation which brings in considerable time and hardware overhead. The algorithm based tolerant (ABFT) technique is a general concept for designing efficient fault tolerant schemes based on structures of the algorithms. The proposed hybrid fault tolerant approach is based on the integration of the ABFT technique. Compare with other fault tolerant schemes, the proposed scheme has at least four key features which listed below:

- Less hardware implementation overhead

Hardware overhead is a considerable factor, as cryptography module has always been an accessorial module in practical scenarios.

Some fault tolerant schemes require additional computational arithmetic unit to be prepared for calculating the checksums or some needs modification of the original crypto algorithm to implement the encoding and decoding logic. Our proposed scheme only involves XOR computational unit which is a basic component across the entire hardware platform. Besides, the checksums needed by our scheme are computed by the original AES rounds, which are generated and compared in an on-going way along with the AES algorithm.

That is to say, the proposed scheme makes use of the computational nature of the targeted algorithm and poses a conceptual way to better create a fault tolerant version by altering the algorithm computation so that its output contains extra information for error detection and correction. It has relatively low overhead and does not require additional arithmetical logic unit.

- Timely intervention to achieve higher fault detection efficiency

According to the nature of AES, multiple rounds of complex computational actions are involved. The mating crypto scheme should be more efficient when it is comes to large-scale data encrypt processing. Our scheme is capable of detecting error as soon as the error exists, and the precision should be within one round of AES algorithm. This feature prevents error data from propagating to the next rounds, which saves computational resources from wrongly processed.

- Organize and process data in matrix-based way

AES is one kind of block ciphers. In order to give full play to the extra fault tolerant scheme, the best way to process data is to treat them as matrix. Our proposed scheme places the checksums as rows and columns attached beside the data matrix and the ABFT algorithm processes the data matrix with checksums in a one-time manner.

- Flexibility of cooperating with multiple versions of AES algorithms

In this paper, we adopted the AES parameters of 128-bit key size, 128-bit plaintext block size, 10 rounds, 128-bit round key size, and 44-word expanded key size as an example. It is flexible to modify our scheme to cooperate with other versions of AES algorithms which might have different key sizes, block sizes or even computation for each round.

5 Conclusion

In this paper, a hybrid fault tolerant approach for AES is proposed. This approach is based on the fault tolerant technique for s-box byte substitution operation and the integration of the algorithm based fault tolerant (ABFT) technique for the AES algorithm. Utilizing the ready-made arithmetic units in the original design, single error can be efficiently detected by the sender. In this way, useless computation and false crypto code can prevent propagation. According to the practical requirements, two versions of the scheme are presented. The general version deals with the whole AES process and the error detection procedure occurs at the end of all rounds. The run-time version performs error detection for every round. Hence, it is capable of terminating the error round immediately. Compared to other fault tolerant schemes for AES, the proposed scheme only brings in overheads of computational time spent on calculating the detection equations, as well as additional memory or register for storing intermediate results. Moreover, without doing much modification to the AES architecture, this scheme can be integrated both on software and hardware in an easy way. The rolling architecture is chosen to implement the general version on Xilinx FPGA board. The simulation result shows that our scheme has a reasonable hardware overhead compared to the existing schemes and the relative overhead will be much less if the unrolling architecture is to be implemented.

References

[1] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri. "On the propagation of faults and their detection in a hardware implementation of the advanced encryption standard," in *Proceedings of 13th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP 2002)*, pp. 303-312, 2002.

[2] 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," in *Proceedings of 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT*

2002), pp. 51-59, 2002.

[3] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "Detecting and locating faults in VLSI implementations of the advanced encryption standard," in *Proceedings of 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2003)*, pp. 105-113, 2003.

[4] 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 Transactions on Computers*, vol. 52, no. 4, pp. 492-505, 2003.

[5] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "An efficient hardware-based fault diagnosis scheme for AES: performances and cost," in *Proceedings of 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2004)*, pp. 130-138, 2004.

[6] L. Breveglieri, I. Koren, and Paolo Maistri, "Detecting faults in four symmetric key block ciphers," in *Proceedings of 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 258-268, 2004.

[7] L. Breveglieri, I. Koren, and P. Maistri, "Incorporating error detection and online reconfiguration into a regular architecture for the advanced encryption standard," in *Proceedings of 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2005)*, pp. 72-80, 2005.

[8] L. Breveglieri, I. Koren, and P. Maistri, "An operation-centered approach to fault detection in symmetric cryptography ciphers," I*EEE Transactions on Computers*, vol. 56, no. 5, pp. 635-649, 2007.

[9] S. Fernandez-Gomez, J. J. Rodriguez-Andina, E. Mandado, "Concurrent error detection in block ciphers," in *Proceedings of International Test Conference 2000 (ITC 2000)*, pp. 979-984, 2000.

[10] R. K. Gulati and S. M. Reddy, "Concurrent error detection in VLSI array structures," in *Proceedings of 1986 IEEE International Conference on Computer Design*, pp. 488-491, 1986.

[11] F. K. Guürkaynak, A. Burg, N. Felber, W. Fichtner, D. Gasser, F. Hug, and H. Kaeslin, "A 2 Gb/s balanced AES crypto-chip implementation," in *Proceedings of 14th ACM Great Lakes symposium on VLSI*, pp. 39-44, 2004.

[12] M. K. Joo, J. H. Kim, and Y. H. Choi, "A fault tolerant architecture for symmetric block ciphers," in *Proceedings of 11th Asian Test Symposium, 2002. (ATS 2002)*, pp. 212-217, 2002.

[13] M. Karpovsky, K. J. Kulikowski, and A. Taubin, "Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard," in *Proceedings of 2004 International Conference on Dependable Systems and Networks*, pp. 93-101, 2004.

[14] R. H. Kuhn, *Yield Enhancement by Fault-tolerant Systolic Arrays in VLSI and Modern Signal Processing*", Prentice-Hall, 1985.

[15] P. Maistri, P. Vanhauwaert, and R. Leveugle, "Evaluation of register-level protection techniques for the advanced encryption standard by multi-level fault injections," in *Proceedings of 22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2007)*, pp. 499-507, 2007.

[16] J. H. Patel and L.Y. Fung, "Concurrent error detection in ALU's by recomputing with shifted operands," I*EEE Transactions on Computers*, vol. C-31, no. 7, pp. 589-595, 1982.

[17] H. Qin, T. Sasao, and Y. Iguchi, "An FPGA design of AES encryption circuit with 128-bit keys," in *Proceedings of 15th ACM Great Lakes symposium on VLSI*, pp. 147-151, 2005.

[18] M. Valinataj, and S. Safari, "Fault tolerant arithmetic operations with multiple error detection and correction," in *Proceedings of 22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2007)*, pp. 188-196, 2007.

[19] K. Wu, K. Ramesh, G. Kuznetsov, and M. Goessel, "Low cost concurrent error detection for the advanced encryption standard," in *Proceedings of International Test Conference 2004 (ITC 2004)*, pp. 1242-1248, 2004.

[20] C. H. Yen and B. F. Wu, "Simple error detection methods for hardware implementation of advanced encryption standard," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 720-731, 2006.

Chang N. Zhang has been at the University of Regina since 1990 where he is currently a Professor of the Department of Computer Science, and Adjunct Scientist with Telecommunication Research Labs (TRLabs). He received his B.S. in Applied Math from Shanghai University, and his Ph.D. in Computer Science and Engineering from Southern Methodist University.

Qian Yu is currently a Ph.D. candidate in the Department of Computer Science at the University of Regina. He received his Master of Science degree in Computer Science from University of Regina in 2007 and his Bachelor of Engineering degree in Computer Science and Technology from National University of Defense Technology, China in 2003.

Xiao Wei Liu was graduated from University of Regina, Computer Science Department. During her post-graduate period, she was supervised by Dr. Chang N. Zhang, funded by Telecommunication Research Labs (TRLabs), and her major research interest was fault tolerance in Cryptography. She is currently working for IBM China Development Lab and mainly focuses on Cloud platform and Application Server middleware related area.