Invariance-Based Concurrent Error Detection for Advanced Encryption Standard

Xiaofei Guo and Ramesh Karri

Polytechnic Institute of New York University
xg243@nyu.edu, rkarri@poly.edu

Abstract. Naturally occurring and maliciously injected faults reduce the reliability of Advanced Encryption Standard (AES) and may leak confidential information. We developed an invariance-based concurrent error detection (CED) scheme which is independent of the implementation of AES encryption/decryption. Additionally, we improve the security of our scheme with Randomized CED Round Insertion and adaptive checking. Experimental results show that the invariance-based CED scheme detects all single-bit, all single-byte fault, and 99.99999997% of burst faults. The area and delay overheads of this scheme are compared with those of previously reported CED schemes on two Xilinx Virtex FPGAs. The hardware overhead is in the 13.2-27.3% range and the throughput is between 1.8-42.2Gbps depending on the AES architecture, FPGA family, and the detection latency. One can implement our scheme in many ways; designers can trade off performance, reliability, and security according to the available resources.

1 Introduction

The Advanced Encryption Standard (AES) is used in a variety of applications, including smart cards, mobile phones, WWW servers, automated teller machines, and digital recorders. The decreasing cost of VLSI chips and increasing user throughput requirements make hardware implementation of AES necessary.

Faults that occur in VLSI chips are classified into two categories: transient faults that eventually die away and permanent faults. The origin of these faults could be internal phenomena in the system, such as threshold changes, shorts, opens, etc., or external influences, such as electromagnetic radiation. These faults affect the memory as well as the combinational parts of a circuit and are detected using concurrent error detection (CED) [13]. Cryptographic chips are sensitive to natural faults in the hardware [2]. A small number of excited faults can cause a large number of output bits of AES to be faulty [1]. Recently, attackers have injected faults into cryptographic circuits to steal secret information as well [9, 10].

1.1 Related Work

Previous work on CED can be classified into four types of redundancy: hardware, time, information, and combined redundancy.

Hardware redundancy duplicates the function and detects faults by comparing the outputs of two copies.

Time redundancy: The function is computed twice on the same input and the results are compared. A variation of the time redundancy is in [7]. The function is computed on both clock edges. This speeds up the computation. Under some conditions, this scheme allows the encryption to be computed twice without affecting the global throughput. This scheme is complex and delicate to implement as technology scales.

Information redundancy: Many fault detection schemes are based on error detecting codes. A few check bits are generated from the input message; then they propagate along with the input message and are finally validated when the output message is generated. The basic parity scheme is proposed in [1]. In this scheme, each predicted parity bit is generated from an input byte. Then, the predicted parity bits, and actual parity bits of output are compared to detect the faults. This scheme incurs large hardware overhead. [15] proposes a scheme in which only single-bit parity is used for the entire 128-bit output, and the parity bit is checked once for the entire round. However, the above two schemes only apply to lookup table-based (LUTs) substitution-box (S-box) implementation. In [8], parity is obtained for S-box implementation that uses logic gates. In [6], a general parity-based scheme is proposed to protect the S-box regardless of its implementation. All these parity schemes share the same limitation. If an even number of faults occur in the same byte, none of these schemes can detect them.

Combined redundancy: In [4], the authors consider CED at the operation, round, and algorithm levels for AES. In these schemes, an operation, a round, or the entire encryption and decryption are followed by their inverses, and the results are compared with the original input to detect faults. Although these schemes detect most of the faults, they require both encryption/decryption to be on chip and can suffer from more than 100% throughput overhead.

1.2 Contributions

We propose a low overhead, implementation independent, and invariance-based CED scheme, which uses combined redundancy for obtaining a reliable AES implementation. The scheme achieves a fault detection capability that is close to [4] and hardware redundancy, the state-of-the-art countermeasures, but with much lower cost. Our contributions are as follows:

- We present the invariance-based[1] CED with Randomized CED Round Insertion and adaptive checking for AES to improve security.
- We prove that the invariance-based CED scheme detects all single-bit and single-byte faults.
- The invariance-based CED achieves an order of 10^5 lower fault miss rate than the best parity schemes for multiple burst and multiple random faults.

This paper is organized as follows: In Section 2, we introduce the AES algorithm. In Section 3, we propose the key idea of our invariance-based CED scheme, and we show the fault coverage, hardware overhead, and detailed analyses of the scheme.

2 Advanced Encryption Standard

In this paper, we consider 128-bit AES as specified by the National Institute of Standards and Technology (NIST). AES encrypts a 128-bit input plaintext into a 128-bit output ciphertext with a user key using 10 nearly identical rounds plus an initial special round (round 0). One AES encryption round consists of SubBytes, ShiftRows, MixColumns, and AddRoundKey denoted by B, S, M, and A, respectively, as shown in Figure 1. In round 0, only AddRoundKey is performed and in round 10, MixColumns is not performed. Each operation in every round acts on a 128-bit input `state`, where each state element is a byte in $GF(2^8)$. In this paper, each byte is denoted by $\boldsymbol{s_{r,c}}$ $(0 \leq r, c \leq 3)$, and it indicates that this byte is in row ***r*** and column ***c*** in state matrix.

$$S = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = [s_{r,c}]_{r,c=0}^{3} \tag{1}$$

In SubBytes, all the bytes are processed separately by 16 S-boxes (SBs). Each S-box performs a nonlinear transformation of the input byte. Let X be the input to the SBs. The resulting output is:

$$Y = B(X) = [y_{r,c}]_{r,c=0}^{3} \tag{2}$$

In ShiftRows, the rows of the state are shifted cyclically byte-wise using a different offset for each row. Row 0 is not shifted, while rows 1, 2, and 3 are cyclically shifted to the left by 1 byte, 2 bytes, and 3 bytes, respectively. The resulting output is:

$$Z = S(Y) = \begin{bmatrix} y_{0,0} & y_{0,1} & y_{0,2} & y_{0,3} \\ y_{1,1} & y_{1,2} & y_{1,3} & y_{1,0} \\ y_{2,2} & y_{2,3} & y_{2,0} & y_{2,1} \\ y_{3,3} & y_{3,0} & y_{3,1} & y_{3,2} \end{bmatrix}$$

[1] Let $f : \{0,1\}^{128} \rightarrow \{0,1\}^{128}$ denotes an operation on the state space of Rijndael which operates completely on the Galois field $GF(2^8)$. A property $P \subseteq \{0,1\}^{128}$, $P \neq 0$, is called an invariance of f, if P is preserved by f , i.e., for every $x \in P$ it follows that $f(x) \in P$. [5]

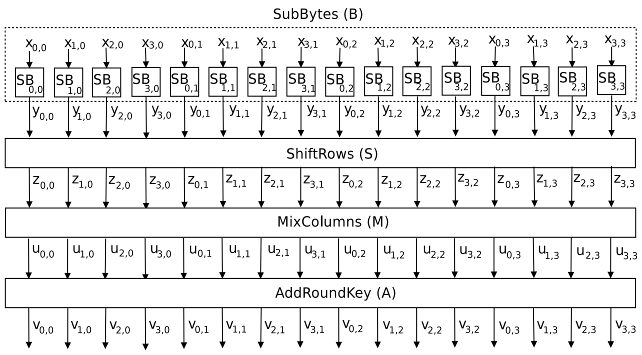


Fig. 1: One AES encryption round

$$= [y_{r,(r+c)\ mod\ 4}]_{r,c=0}^{3} = [z_{r,c}]_{r,c=0}^{3} \tag{3}$$

In MixColumns, the output state is obtained by multiplying a constant matrix with the output of ShiftRows. The resulting output is:

$$U = M(Z) =$$

$$\begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} z_{0,0} & z_{0,1} & z_{0,2} & z_{0,3} \\ z_{1,0} & z_{1,1} & z_{1,2} & z_{1,3} \\ z_{2,0} & z_{2,1} & z_{2,2} & z_{2,3} \\ z_{3,0} & z_{3,1} & z_{3,2} & z_{3,3} \end{bmatrix} = [u_{r,c}]_{r,c=0}^{3} \tag{4}$$

In AddRoundKey, the input state is added (modulo-2) to the round key, i.e., the 128-bit state U with matrix $K = [k_{r,c}]_{r,c=0}^{3}$. The resulting output is:

$$V = A(K)U = [u_{r,c}]_{r,c=0}^{3} + [k_{r,c}]_{r,c=0}^{3} = [v_{r,c}]_{r,c=0}^{3} \tag{5}$$

3 Invariances of AES

In [5], the authors have shown that AES exhibits various mapping invariance properties. They were investigating these as possible source of weakness in AES. In contrast, we use these invariances for CED to protect the AES against random faults and malicious attacks. We analyze three round level invariances of AES. We give a formal proof of the invariance property P_1, which is the most effective invariance according to our experimental results. We analyze the effectiveness of the remaining invariances in Section 4.

Theorem 1. *An AES round can be represented as*

$$A(K)(M(S(B(X))))$$

where X is the 128-bit input to the round. Exist byte permutation P, so that the following hold true:

$$A(K)(M(S(B(X)))) = P^{-1}(A(P(K))(M(S(B(P(X)))))) \tag{6}$$

where P^{-1} denotes the inverse function of P.

One of the byte permutation is:

$$P_1(X) = P_1([x_{r,c}]_{r,c=0}^{3}) = \begin{bmatrix} x_{0,3} & x_{0,0} & x_{0,1} & x_{0,2} \\ x_{1,3} & x_{1,0} & x_{1,1} & x_{1,2} \\ x_{2,3} & x_{2,0} & x_{2,1} & x_{2,2} \\ x_{3,3} & x_{3,0} & x_{3,1} & x_{3,2} \end{bmatrix}$$

$$= [x_{r,(c+3)\ mod\ 4}]_{r,c=0}^{3} \tag{7}$$

$$P_1^{-1}([x_{r,(c+3)\ mod\ 4}]_{r,c=0}^{3}) = [x_{r,c}]_{r,c=0}^{3} \tag{8}$$

Proof. Let us start from the right-hand side of the equation. First, we apply permuted input $X^{'} = P_1(X)$ to SubBytes, and from (2) and (7), we get:

$$Y^{'} = [y^{'}_{r,c}]^3_{r,c=0} = [y_{r,(c+3)\ mod\ 4}]^3_{r,c=0} = P_1(Y) \tag{9}$$

Given $Y^{'}$ as the input to ShiftRows, we get:

$$Z^{'} = S(Y^{'}) = \begin{bmatrix} y^{'}_{0,0} & y^{'}_{0,1} & y^{'}_{0,2} & y^{'}_{0,3} \\ y^{'}_{1,1} & y^{'}_{1,2} & y^{'}_{1,3} & y^{'}_{1,0} \\ y^{'}_{2,2} & y^{'}_{2,3} & y^{'}_{2,0} & y^{'}_{2,1} \\ y^{'}_{3,3} & y^{'}_{3,0} & y^{'}_{3,1} & y^{'}_{3,2} \end{bmatrix}$$

$$= \begin{bmatrix} y_{0,3} & y_{0,0} & y_{0,1} & y_{0,2} \\ y_{1,0} & y_{1,1} & y_{1,2} & y_{1,3} \\ y_{2,1} & y_{2,2} & y_{2,3} & y_{2,0} \\ y_{3,2} & y_{3,3} & y_{3,0} & y_{3,1} \end{bmatrix} = [z^{'}_{r,c}]^3_{r,c=0} \tag{10}$$

From (3) and (10), we find that:

$$Z^{'} = [z_{r,(c+3)\ mod\ 4}]^3_{r,c=0} = P_1(Z) \tag{11}$$

Applying $Z^{'}$ to MixColumns, we get:

$$U^{'} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} z^{'}_{0,3} & z^{'}_{0,0} & z^{'}_{0,1} & z^{'}_{0,2} \\ z^{'}_{1,3} & z^{'}_{1,0} & z^{'}_{1,1} & z^{'}_{1,2} \\ z^{'}_{2,3} & z^{'}_{2,0} & z^{'}_{2,1} & z^{'}_{2,2} \\ z^{'}_{3,3} & z^{'}_{3,0} & z^{'}_{3,1} & z^{'}_{3,2} \end{bmatrix}$$

$$= [u_{r,(c+3)\ mod\ 4}]^3_{r,c=0} = P_1(U) \tag{12}$$

Then we apply the permuted round key matrix $K^{'} = P_1(K)$ resulting in:

$$V^{'} = [u^{'}_{r,c}]^3_{r,c=0} + [k^{'}_{r,c}]^3_{r,c=0}$$

$$= [u_{r,(c+3)\ mod\ 4}]^3_{r,c=0} + [k_{r,(c+3)\ mod\ 4}]^3_{r,c=0} = P_1(V) \tag{13}$$

Finally we apply inverse permutation P^{-1} to the output. We get:

$$P_1^{-1}(V') = P_1^{-1}(P_1(V)) = V \tag{14}$$

3.1 Invariance-based CED Architecture

We design two invariance-based CED architectures for AES: Fully pipelined and iterative.

Fully pipelined: There are 11 stages in our pipelined architecture. For each pipeline stage in Figure 2(a), we add two muxes (mux_x and mux_k) and a comparator (cmp). P is a permutation of wires based on the invariance property. P^{-1} is the inverse permutation. We need two encryption cycles to detect the faults, and let us call them C1 and C2. In C1, let the input and the key of round 1 be $X1$ and $K1$. We run the encryption with mux_x and mux_k, selecting $X1$ and $K1$ as the inputs. The round result $V1$ is stored in the data register. Then we run C2; we run the encryption with mux_x and mux_k, selecting permuted inputs $X1^{'}$ and $K1^{'}$, respectively. At the end of C2, we inverse permute output $V1^{'}$ and compare it with the value $V1$ stored in the data register. If the results are equal, no fault is detected. Otherwise, the comparator will assert the fault indication flag. The comparator does not add delay to the critical path because the comparison can be performed when the next round input is executed. We see that C1 can be any normal encryption cycle, and C2 is the corresponding extra cycle, which selects the permuted inputs to be performed after every C1. One can add a C2 after R rounds; we call R the checking ratio. R can be changed based on the tradeoff between performance, reliability, and security specified by the designer. For a detailed analysis, please see Section 3.3.

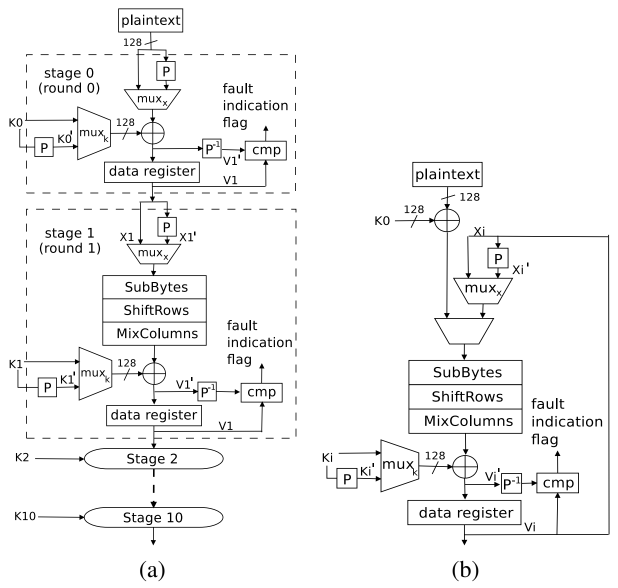


Fig. 2: AES hardware architectures (a) Pipeline. (b) Iterative. cmp stands for comparator. We assume that the comparator is fault tolerant.

Iterative: As shown in Figure 2(b), we add mux_x and mux_k and a comparator. There is one security benefit of iterative implementation. Each ciphertext takes 10 cycles to generate in an iterative architecture. If the designer chooses to add a C2 for every ten cycles, then the faulty ciphertext will not be sent to the output because the fault is detected before it is generated. This will further prevent an attacker from stealing the secret key. In the pipeline architecture, a ciphertext is generated every cycle. Therefore, if the faults are generated before the comparison, faulty outputs will be obtained by the attacker.

3.2 Fault Analysis

Invarianced-based CED detects all single-bit and single-byte faults. Our simulations show that this CED scheme detects 99.99999997% of multiple burst faults and close to 100% of multiple random faults. Fault coverage (FC) is calculated as:

$$FC = 1 - FMR$$

where FMR is the fault miss rate calculated as:

$$FMR = \frac{T_{undetected}}{T_{total} - T_{correct}}$$

where $T_{undetected}$ is the number of tests in which faults are excited but not detected. T_{total} is the total number of tests we applied. $T_{correct}$ represents the tests in which the faults are not excited.

100% Fault Coverage of Single-Bit Faults

Theorem 2. *If a single-bit fault in any of the steps in a round affects the outputs of the final result of that round, our scheme will detect it.*

Proof. **Case 1: A single-bit fault in S-box (SB)**. In Figure 1, let the $SB_{i,j}$ ($0 \leq i, j \leq 3$) have a single-bit stuck-at fault. If the SBs are implemented using ROMs, the considered fault corresponds to an address fault of the ROM, a fault in memory location, or a fault in the output data lines. If the SBs are implemented using combinational logic, the considered fault can appear in any gate of the implementation.

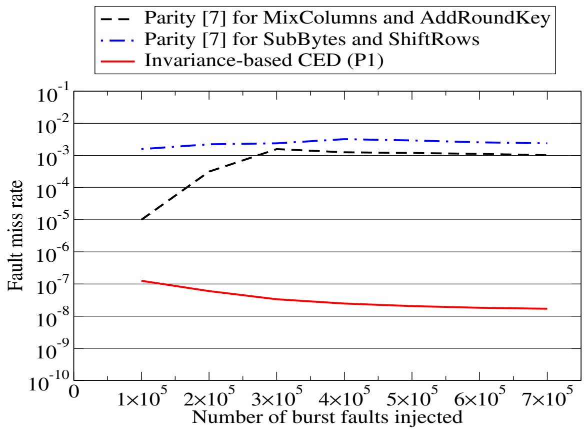


Fig. 3: Simulation results show that the fault miss rate of the invariance-based CED is superior to that of the parity scheme in [6].

In C1, the $SB_{i,j}$ generates faulty output $y_{i,j}$. After ShiftRows, the outputs are $[z_{r,c}]_{r,c=0}^{3} = [y_{r,(r+c)\ mod\ 4}]_{r,c=0}^{3}$, and the faulty state element is $z_{i,(j-i)\ mod\ 4} = y_{i,j}$. In MixColumns, a single faulty input causes four bytes within the same column to be faulty. The faulty state elements are represented as $[u_{r,(j-i)\ mod\ 4}]_{r=0}^{3}$. After AddRoundKey, $[v_{r,(j-i)\ mod\ 4}]_{r=0}^{3}$ are the faulty state elements. In C2, we apply $X^{'}$ and $K^{'}$ as the permuted inputs. Using the same steps shown above, faulty state elements are represented as $[v^{'}_{r,(j-i)\ mod\ 4}]_{r=0}^{3}$. From (8), we know that

$$[v^{'}_{r,(j-i)\ mod\ 4}]_{r=0}^{3} = [v_{r,((j-i)+3\ mod\ 4)\ mod\ 4}]_{r=0}^{3}$$

Therefore, the faulty column in C1 is $(j-i)\ mod\ 4$, but the faulty column in C2 corresponds to column $((j-i)\ mod\ 4)+3)\ mod\ 4$ in C1; note that $0 \leq i, j \leq 3$ and $(j-i)\ mod\ 4 \neq (((j-i)\ mod\ 4)+3)\ mod\ 4$.

Since faulty $SB_{i,j}$ affects different columns in C1 and C2, we always compare a faulty column with a nonfaulty column, and our scheme detects the fault as long as it affects the output. For a concrete example, let $SB_{1,2}$ be faulty, thus, $y_{1,2}$ is the faulty output byte. After ShiftRows, $z_{1,1} = y_{1,2}$. After MixColumns, the faulty state elements are shown as $[u_{r,1}]_{r=0}^{3}$. Then we apply this as the input of AddRoundKey, so faulty state elements are shown as $[v_{r,1}]_{r=0}^{3}$. Then we run C2, and faulty state elements are represented as $[v^{'}_{r,1}]_{r=0}^{3}$. Because $[v^{'}_{r,1}]_{r=0}^{3} = [v_{r,0}]_{r=0}^{3}$, the faulty columns in C1 and C2 are different, and we detect the fault by comparing the outputs.

Case 2: A single-bit fault in ShiftRows. A fault in ShiftRows is equivalent to a fault at the input of MixColumns, and thus, we prove this in case 3.

Case 3: A single-bit fault in MixColumns. Since MixColumns is mainly implemented with XOR and a few other basic gates, we consider a fault in MixColumns in three scenarios: the input, the internal logic gates, and the output. If there is a stuck-at fault in the input, the fault will propagate to all four bytes in the same column. Assuming that column $[u_{r,j}]_{r=0}^{3}$ is faulty in C1, we know that the column $[v_{r,j}]_{r=0}^{3}$ is faulty in the final output of the current round. In C2, we know that the column $[u^{'}_{r,j}]_{r=0}^{3}$ is faulty, and the column $[v^{'}_{r,j}]_{r=0}^{3}$ of the output of the round is also faulty. From (12) and (14), we know that $j^{'} = (j+3)\ mod\ 4$. Because the faulty columns of the two outputs are different, we detect the fault.

Case 4: A single-bit fault in AddRoundKey. AddRoundKey is mainly implemented as bit-wise XOR gates. We consider a fault in AddRoundKey as stuck-at fault in the input or output. The fault at the input is equivalent to the fault in the output of MixColumns. Let us prove the theorem true for a single-bit fault at the output of AddRoundKey. Let the faulty bit affects byte $v_{i,j}$ in the C1 and $v^{'}_{i,j}$ in C2. From (14), we know that $v^{'}_{i,j} = v_{i,(j+3)\ mod\ 4}$. Again, the faulty columns in C1 and C2 are different, and thus we detect the fault.

100% Fault Coverage for Single-Byte Faults Recent experiments have shown that high-energy ions can energize two or more adjacent memory cells in a circuit [11]. Because an attacker can choose light [14] or electromagnetic

Table 1: Comparison of fault coverage. a. burst faults b. random faults

FDS	Fault coverage		
	Single bit	Single byte	Multiple bit
HW red.	100%	100%	100%
Parity 1 [1]	100%	50%	99.997%
Parity 2 [15]	98.7%	50%	48-53%
Parity 3 [6]	100%	50%	99.996%
Alg. level [4]	100%	100%	100%
Invariance	**100%**	**100%**	**99.99999997%** [a] **100%** [b]

radiation [12] to inject faults, this is a realistic attack model. We define single-byte fault as faults that affect at most a single byte, i.e., they can affect from one bit upto eight bit of the byte.

Theorem 3. *If multiple faults in any of the processing steps in a round affect a byte quantity, the invariance scheme will detect it.*

Proof. We prove this theorem for single-byte fault in either S-box, ShiftRows, MixColumns, or AddRoundKey.

Case 1: A single-byte fault in S-box (SB). Let multiple faults in S-box affect one byte output $y_{i,j}$ in C1. From the proof of our theorem 1, after AddRoundKey, faulty state elements are shown as $[v_{r,(j-i)\ mod\ 4}]_{r=0}^{3}$. After C2, the faulty state elements are represented as $[v'_{r,(j-i)\ mod\ 4}]_{r=0}^{3}$. Theorem 1 proves that these faults are detected.

A single-byte fault in ShiftRows, MixColumns, and AddRoundKey can also be similarly detected.

In order to have a 100% single-byte fault detection rate, one need hardware redundancy or combined redundancy [4], both of which have more than 100% hardware overhead. Our fault simulation confirms that the invariance-based CED detects all single-bit and single-byte faults.

Fault Coverage for Multiple Faults We simulated multiple faults for the invariance scheme and compared it with the one proposed in [6]. These models cover both natural faults and fault attacks [3].

Due to technology constraints, an attacker may not be able to inject a single-bit fault [3]. Multiple faults are actually injected in the process. We use burst and random fault models [3]. We use Fibonacci Linear Feedback Shift Register (LFSR) with 128-bit output taps to inject faults. The maximum sequence length polynomial for the LFSR is selected as $L(X) = X^{128} + X^{29} + X^{27} + X^{2} + 1$.

Burst faults occur at the output of only one operation at a time, i.e., the faults are injected at the 128-bit output of one operation in the AES encryption. This includes both stuck-at zero and one faults. The size, location, and type of the burst are randomly generated.

The results of our simulations for the burst faults in the AES encryption are shown in Figure 3. We compare our miss rate with that of [6]. The dot-dash line respresents the fault miss rate of SubBytes and ShiftRows in the AES encryption in [6]. The dash line represents the fault miss rate of MixColumns and AddRoundKey in the AES encryption in [6]. The solid line represents the fault miss rate of the invariance-based CED. We have injected up to 700,000 burst faults at the operation outputs and monitored the faults that are detected by the fault indication flag. The fault miss rate for [6] is between 10^{-2} and 10^{-3}. The miss rate of our scheme is between 10^{-7} and 10^{-8}; a reduction of 10^{5}.

Random faults are injected at random locations, i.e., four 128-bit outputs of the operations. In another simulation, we saw 0% fault miss rate after injecting up to 700,000 random faults.

Comparison of Fault Coverage As shown in Table 1, for single-bit and single-byte faults, the invariance-based CED provides 100% fault coverage, the same as [4] and hardware redundancy. It is note that [4] requires both encryption/decryption to be on chip to achieve such fault coverage. While most of the parity schemes achieves 100% fault coverage for single-bit fault, they can only provide 50% fault coverage for single-byte fault [8]. For multiple faults, [4] and hardware redundancy provide 100% fault coverage. The invariance-based CED provides 99.99999997% fault coverage, and much higher than the parity-based schemes. The tradeoff between performance and detection latency can be

Table 2: Comparisons of implementation of CED schemes on two Xilinx FPGAs. We use the metrics, FPGA platform, and results from [6]. Our pipeline implementation are shown in bold, and we implement the iterative architectures. a. the latency is 2x the original AES encryption/decryption b. using two (256 $\times$ 9) ditributed memories for CED of each S-box or inverse S-box c. using (256 $\times$ 9) distributed memories for CED of each S-box or inverse S-box d. checking ratio is 11 e. checking ratio is 10 f. checking ratio is 1

FPGA (Device)	Arch.	Scheme	Encryption				Decryption			
			Slice (overhead)	Freq. (MHz)	Thro. (Gbps)	Eff(Mbps /slice)	Slice (overhead)	Freq. (MHz)	Thro. (Gbps)	Eff.(Mbps /slice)
$Virtex^{TM}$-4 (xc4vlx160-12)	Pipe.	Original	18335(-)	240.5	30.8	1.7	19322(-)	203.5	26.0	1.3
		HW Red.	36684(100.1%)	240.5	30.8	1.1	38658(100.1%)	203.5	26.0	0.7
		Parity 1 [1][b]	39104(113.3%)	163.5	20.9	0.5	40244(108.3%)	145.4	18.6	0.5
		Parity 2 [15][c]	21211(15.7%)	240.5	30.8	1.4	22280(15.3%)	203.5	26.0	1.1
		Parity 3 [6]	20127(9.8%)	240.5	30.8	1.5	20909(8.2%)	203.5	26.0	1.2
		Alg. level [4]	38273(108.7% [6]) (1.6%)	194.9	24.9[a]	0.6	38273(98.1% [6]) (1.6%)	194.9	24.9[a]	0.6
		Invariance	**21253(15.9%)**	**232.3**	**27.2[d]– 14.9[f]**	**1.28[d]– 0.7[f]**	**22240(15.1%)**	**197.6**	**23.1[d]– 12.6[f]**	**1.0[d]– 0.6[f]**
	Iter.	Original	1905(-)	224.6	2.9	1.5	2002(-)	192.0	2.5	1.2
		Invariance	**2170(13.9%)**	**217.4**	**2.55[e]– 1.4[f]**	**1.2[e]– 0.7[f]**	**2267(13.2%)**	**186.7**	**2.2[e]– 1.2[f]**	**1.0[e]– 0.6[f]**
$Virtex^{TM}$-5 (xc5vlx110-3)	Pipe.	Original	2960(-)	371.7	47.6	16.1	3906(-)	296.3	37.9	9.7
		HW Red.	5934(100.5%)	371.7	47.6	10.2	7826(100.4%)	296.3	37.9	5.5
		Parity 1 [1][b]	5590(88.9%)	282.8	36.2	6.5	6680(71.2%)	260.2	33.3	4.9
		Parity 2 [15][c]	3619(22.3%)	304.0	38.9	10.7	4426(13.3%)	277.0	35.5	8.0
		Parity 3 [6]	3757(26.9%)	371.7	47.6	12.7	4286(9.7%)	296.3	37.9	8.8
		Alg. level [4]	5849(97.6% [6]) (-14.8%)	284.4	36.4[a]	6.2	5849(49.7% [6]) (-14.8%)	284.4	36.4	6.2
		Invariance	**3664(23.9%)**	**358.9**	**42.2[d]– 23.0[f]**	**12.0[d]– 6.6[f]**	**4434(13.5%)**	**288.9**	**33.9[d]– 18.5[f]**	**7.6[d]– 4.2[f]**
	Iter.	Original	344(-)	347.0	4.4	12.8	462(-)	286.4	3.7	7.9
		Invariance	**438(27.3%)**	**335.8**	**3.9[e]– 2.2[f]**	**7.9[e]– 4.4[f]**	**539(16.7%)**	**273.1**	**3.2[e]– 1.8[f]**	**5.9[e]– 3.7[f]**

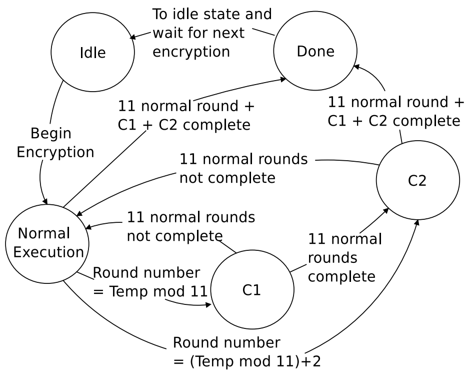
explored by varying the checking ratio R, which is the ratio of the number of results computed without invariance to the number of results computed with invariance-based CED.

3.3 Security Analysis

In this section, we propose two policies that the designers can employ to further secure their design.

Randomized control: The invariance-based CED technique cannot detect the faults that are not excited in C1 and C2 rounds. If an attacker determines the architecture of the AES with the proposed CED implementation, this feature can be used as a weakness to insert faults in such a way that they do not exist during the CED rounds but only during the normal rounds. To prevent this, we proposed using Randomized CED Round Insertion (RCRI). In this method, the positions of the CED rounds C1 and C2 are randomized during the 11-round AES encryption process for pipelined architecture. This can be implemented as shown in the state diagram of Figure 4. A random number can be obtained using the randomness property of the AES algorithm.

For example, a Rand register can be incorporated into the circuit with some random number stored in it at manufacture time and for every subsequent encryption performed, the resulting ciphertext is xored with Rand to get a Temp number. When an encryption is performed, the algorithm enters the normal execution state. Normal encryption rounds are performed until the value of the Temp modulo 11 equals the round number. Once this condition is satisfied, the CED round C1 is performed. Depending on whether 11 normal rounds have been performed, either C2 or the remaining normal rounds are performed. The encryption process is complete when 11 normal rounds and the randomly inserted C1 and C2 CED round are complete. For the mod operation, we can take the last four bits of Temp and apply it to a



Idle
Done
To idle state and wait for next encryption
11 normal round + C1 + C2 complete
11 normal round + C1 + C2 complete
Begin Encryption
11 normal rounds not complete
C2
11 normal rounds not complete
Normal Execution
11 normal rounds complete
Round number = Temp mod 11
C1
Round number = (Temp mod 11)+2

Fig. 4: State machine for Randomized CED Round Insertion (RCRI)

lookup table which contains modulo 11 results from input 0 to 15. For the iterative architecture, since the encryption takes 10 rounds, we use Temp modulo 10.

Adaptive checking: If the designer chooses $R \neq 1$, some of the transient faults injected by the attacker may go undetected. However, an attacker needs to obtain a large number of faulty outputs to steal the secret key. We suggest the designer deploy an adaptive approach. When a fault is detected, the chip changes its checking ratio from the R specified by the designer to 1, and thus almost all the faults will be detected. If faults are detected in a number of consecutive checks, the chip can stop its function or self-destruct.

If $R = 1$, all results will be checked. The fault miss rate remains the same for permanent and transient faults. If $R > 1$, every R^{th} result will be checked. Let us assume the transient faults appear for N cycles. When $R \leq N$, the fault coverage remains the same, because the results of C1 and C2 will be checked before the faults disappear. When $R > N$, the probability of detecting a single-bit and single-byte fault is $\frac{N}{R} \times 100\%$ and that of multiple burst faults is $\frac{N}{R} \times 99.99999997\%$.

3.4 Implementation and Comparison

The implementation results shown in Table 2 are all post place-and-route. We implement fully pipelined and iterative architectures. We use pipelined distributed memories for S-boxes and inverse S-boxes similar to [6]. Hardware redundancy, information redundancy [1, 6, 15], combined redundancy [4], and invariance-based CED are compared. The metrics include (1) slice utilization (the number of occupied slices), (2) slice overhead (ratio of number of slices for CED schemes over the number of slices for AES), (3) maximum clock frequency, (4) throughput, and (5) efficiency (raitio of (4) over (1)).

For pipeline architecture, the hardware overhead of the invariance-based CED is much lower than that of hardware redundancy for both encryption and decryption. The invariance-based CED provides flexible throughput. If the designer checks all rounds, checking ratio R is 1 and the throughput overhead is 100%. If the designer performs one check per ciphertext, the checking ratio R is 11 for pipeline architecture and 10 for iterative. The throughput overhead is $\frac{1}{11}$ for the pipeline architecture and $\frac{1}{10}$ for the iterative architecture. If R is large, then the throughput is unaffected. The invariance-based CED has higher efficiency when the checking ratio is 11 compared to hardware redundancy. Because the scheme in [15] uses 1-bit signatures for the 128-bit block of data, it has lower hardware overhead and higher efficiency compared to invariance-base scheme. However, from Table 1, the fault coverage of this scheme is the lowest. [1] and [6] use 16 bits for each 128-bit block, and this leads to much higher fault coverage. The invariance-based CED has much smaller hardware overhead and higher efficiency than [1], but provides higher fault coverage. Another limitation of [1] and [15] is that they are only applicable to S-box implementation using LUT. On Virtex-5, the invariance-based CED has higher efficiency than most of the CED schemes except [6]. Although the invariance-based CED has approximately the same hardware overhead compared to [6], it detects all single-byte faults and lowers the fault miss rate of multiple burst faults by an order of 10^5. The schemes in [4] are only applicable when both encryption and decryption are both on the same chip. Therefore, if only encryption or decryption is on chip, the hardware overhead

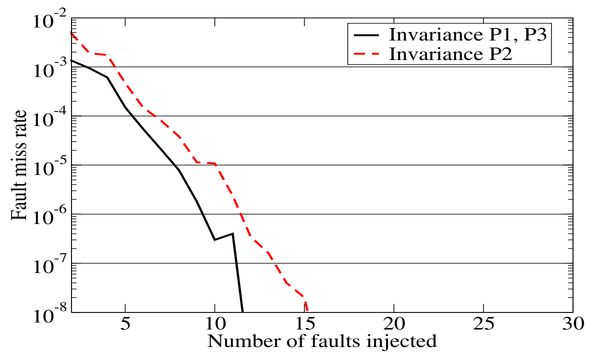


Fig. 5: Fault miss rate for invariance P_1, P_2, and P_3

of [4] is in the 49.7–108.7% range [6], e.g., 108.7% for AES encryption on Virtex-4 FPGA. If both encryption and decryption are on the same chip, the hardware overhead of [4], which is from the comparator, is very low. For Virtex-5, the overhead of this scheme is -14.8%, because the slice utilization of this scheme is smaller than the total slice utilization of encryption and decryption. However, the efficiency of the invariance-based CED is higher than that of [4]. Most importantly, the invariance-based CED and all other CED schemes do not require both encryption and decryption to be on chip.

For iterative architecture, since round 0 is performed in the same clock cycle as round 1, an extra delay is added in the critical path. The hardware overhead of the invariance-based CED as a 16.7-27.3% is slightly higher than that of the pipeline architecture on Virtex-5.

4 Concluding Remarks

There are several other mapping invariances that can be used for CED [5]. Most of them restrict the pattern of the inputs and thus are not effective when realistic random inputs are provided. However, there are two other invariances that allow us to perform CED on any inputs:

$$P_2(X) = \begin{bmatrix} x_{0,2} & x_{0,3} & x_{0,0} & x_{0,1} \\ x_{1,2} & x_{1,3} & x_{1,0} & x_{1,1} \\ x_{2,2} & x_{2,3} & x_{2,0} & x_{2,1} \\ x_{3,2} & x_{3,3} & x_{3,0} & x_{3,1} \end{bmatrix} \tag{15}$$

$$P_3 = P_1(P_{pre}(X)) = P_1(\begin{bmatrix} x_{0,0} & x_{0,3} & x_{0,2} & x_{0,1} \\ x_{1,0} & x_{1,3} & x_{1,2} & x_{1,1} \\ x_{2,0} & x_{2,3} & x_{2,2} & x_{2,1} \\ x_{3,0} & x_{3,3} & x_{3,2} & x_{3,1} \end{bmatrix}) \tag{16}$$

P_2 swaps the first and third columns and also the second and fourth columns in the state matrix. P_3 is the same as P_1 except that the initial input X is permuted by P_{pre} before being applied to the input. Fault miss rate for CED using invariances P_1, P_2, and P_3 are compared in Figure 5. The fault miss rate dropped sharply to below 10^{-7} after we injected 11 faults using P_1 and P_3, and 15 faults for P_2. After we injected 15 faults, the fault miss rate was very low and thus not shown in Figure 5. The fault miss rates of invariance P_1 and P_3 are lower than the fault miss rate of invariance P_2 with any number of faults injected. Compare to P_2, the area and performance overheads of P_1 is the same. For P_3, one first need to apply $P_{pre}(X)$ as input to run C1, and store the result V_{pre}. Then use $P_1(P_{pre}(X))$ as input to run C2, and apply P_1^{-1} to the result $V_{pre}^{'}$ to compare with V_{pre}. Both C1 and C2 are extra overhead for performance. Compare with P_3, C2 is the only performance overhead for P_1. Therefore, P_1 is the most effective invariance.

5 Acknowledgments

This material is based upon work supported by the NSF CNS program under grant 0831349.

References

1. Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard. *IEEE Trans. Computers*, 52(4):492–505, 2003.
2. D. Boneh, R. DeMillo, and R. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Eurocrypt*, pages 37 – 51. Lecture Notes in Computer Science, 1997.
3. Luca Breveglieri, Israel Koren, and Paolo Maistri. An Operation-Centered Approach to Fault Detection in Symmetric Cryptography Ciphers. *IEEE Trans. Computers*, 56:635–649, May 2007.
4. Ramesh Karri, Kaijie Wu, Piyush Mishra, and Yongkook Kim. Concurrent Error Detection of Fault-Based Side-Channel Cryptanalysis of 128-Bit Symmetric Block Ciphers. In *DAC*, pages 579–585, 2001.
5. Tri Van Le, Rudiger Sparr, Ralph Wernsdorf, and Yvo Desmedt. Complementation-Like and Cyclic Properties of AES Round Functions. In *AES*, pages 128–141, 2005.
6. M. Mozaffari-Kermani and A. Reyhani-Masoleh. Concurrent Structure-Independent Fault Detection Schemes for the Advanced Encryption Standard. *IEEE Trans. Computers*, 59(5):608–622, 2010.
7. P. Maistri and R. Leveugle. Double-Data-Rate Computation as a Countermeasure against Fault Analysis. *IEEE Trans. on Computers*, 57(11):1528–1539, Nov 2008.
8. Mehran Mozaffari-Kermani and Arash Reyhani-Masoleh. A Lightweight High-Performance Fault Detection Scheme for the Advanced Encryption Standard Using Composite Field. *IEEE Trans. VLSI Systems*, 19(1):85–91, 2011.
9. G. Letourneux P. Dusart and O. Vivolo. Differential Fault Analysis on AES. In *Cryptology ePrint Archive*, 2003.
10. G. Piret and J.J. Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad. In *CHES*, pages 77–88, Sept 2003.
11. R.A. Reed, M.A. Carts, P.W. Marshall, C.J. Marshall, O. Musseau, P.J. McNulty, D.R. Roth, S. Buchner, J. Melinger, and T. Corbiere. Heavy Ion and Proton Induced Single Event Multiple Upsets. *IEEE Trans. on Nuclear Science*, 44(6):2224–2229, 1997.
12. David Samyde, Sergei Skorobogatov, Ross Anderson, and Jean-Jacques Quisquater. On a New Way to Read Data from Memory. In *proceedings of IEEE Security in Storage Workshop*, pages 65–69, Dec 2002.
13. Daniel P. Siewiorek and Robert S. Swarz. Reliable Computer Systems: Design and Evaluation. *A K Peters/CRC Press; 3 edition*, 1998.
14. Sergei P. Skorobogatov and Ross J. Anderson. Optical Fault Induction Attacks. In *proceedings of CHES*, pages 2–12, Aug 2002.
15. Kaijie Wu, Ramesh Karri, G. Kuznetsov, and M. Goessel. Low Cost Concurrent Error Detection for the Advanced Encryption Standard. In *ITC*, pages 1242–1248, Oct 2004.