Provably Secure Concurrent Error Detection Against Differential Fault Analysis

Xiaofei Guo, Debdeep Mukhopadhyay and Ramesh Karri

21 Sept, 2012

Abstract

Differential fault analysis (DFA) poses a significant threat to Advanced Encryption Standard (AES). It has been demonstrated that DFA can use only a single faulty ciphertext to reveal the secret key of AES in an average of 2^{30} computation. Traditionally, concurrent error detection (CED) is used to protect AES against DFA. However, we emphasize that conventional CED assumes a uniform distribution of faults, which is not a valid assumption in the context of DFA. In contrast, we show practical examples which highlight that an attacker can inject specific and exploitable faults, thus threatening existing CED. This paper brings to the surface a new CED approach for cryptography, aimed at providing provable security by detecting all possible DFA-exploitable faults, which is a small subset of the entire fault space. We analyze the fault coverage of conventional CED against DFA-exploitable faults, and we find that the fault coverage of most of these techniques are significantly lower than the one they claimed. We stress that for security, it is imperative that CED should provide 100% fault coverage for DFA-exploitable faults. We further propose an invariance-based CED which provides 100% provable security against all known DFA of AES.

Keywords: Differential Fault Analysis, Concurrent Error Detection

1 Introduction

In addressing the security requirements of various information disciplines, e.g., networking, telecommunications, database systems, and mobile applications, applied cryptography has recently gained immense importance. To satisfy the high throughput requirements of such applications, the cryptographic systems are implemented either as cryptographic accelerators (ASIC and FPGA implementations), or as cryptographic libraries (optimized software routines). The complexity of these hardware and software implementations is raising concerns regarding their security and reliability.

Advanced Encryption Standard (AES) is the standard secret key algorithm [32]. To provide high security features, AES implementations have been employed in an increasing number of consumer products with dedicated hardware, e.g., smart cards, cell phones, servers, FPGAs, and TV set-top boxes. Although AES is difficult to break mathematically, these hardware circuits, unless carefully designed, may result in security vulnerabilities.

Because the AES algorithm is public, it is subject to continuous, vigilant, expert cryptanalysis. To obtain the secret key, which allows one to recover the encrypted information, an attacker must perform a brute force analysis that requires a prohibitively large number of experiments. Some of the attacks includes linear cryptanalysis differential cryptanalysis truncated differentials square attack interpolation attack algebraic attack and hybrid attack [17]. Although these purely mathematical attacks reduce the key search space, they cannot break AES [17].

It is imperative that hardware and software implementations of cryptographic algorithms should prevent not only purely mathematical cryptanalysis, but also leakage of secret keys due to deliberately injected faults. An attacker can inject malicious faults into a cryptographic device. By building correlations between the non-faulty and corresponding faulty outputs, an attacker is able to drastically reduces the key space needed to brute force the secret key. This is known as Differential fault analysis (DFA). Radiation, heat, incorrect voltages, and atypical clock rates all cause cryptographic devices to malfunction [5]. DFA of the Data Encryption Standard (DES) and other symmetric block ciphers are demonstrated in [9]. Later, DFA of AES has been studied extensively [35, 34, 13, 10, 26, 31, 36, 40]. DFA has evolved into an effective attack and needs to be carefully analyzed and suitably countered.

In recent years, DFA has been demonstrated to be practical, inexpensive, and dangerous [12, 5, 6]. Optical fault injection attack employed a $30 camera flashgun and a microscope to demonstrate its effectiveness on widely used smart cards [15]. Other inexpensive, controlled fault injection methods include varying the supply voltage, the clock frequency, and the operating temperature, and introducing glitches in the clock signal and the supply voltage [6]. Glitches can be introduced in the clock signal and supply voltage at the desired clock cycle, therefore enabling fault injections to disrupt the correct function of cryptographic systems [38]. The success of several DFA of AES has been achieved by injecting clock glitches [36, 4]; such shortening causes multiple errors corrupting a single byte or multiple bytes. An attacker is able to inject transient faults by lowering the supply voltage, also known as underpowering. Because this attack does not require precise timing, the faults tend to occur uniformly throughout the computation, thus, requiring the attacker to discard those faulty ciphertexts that are useless. This methodology is reported to be effective on ASIC implementations of AES [37, 7], as well as FPGA implementation [21].

Once a DFA attack has been developed and made public, its application does not always require high technical skills and/or expensive equipment. Therefore, incorporating countermeasures against DFA into cryptographic devices is necessary for security purposes. National Institute of Standards and Technology (NIST) formulates security requirements for cryptographic modules in FIPS 140 [33]. It defines four levels of security. At security level 4, the highest, the protection circuitry shall either (1) shut down the module to prevent further operation or (2) immediately zeroize all plaintexts and secret keys. Because faults can be detected using concurrent error detection (CED) [22], CED is used as a major countermeasure for DFA attacks. Traditionally, fault coverage indicates the number of random faults being detected. It is also used to show the security provided by CED against DFA.

Until now, CED for AES has been extension of conventional techniques [41, 8, 20, 27, 18, 11, 14, 16, 19]. Several researchers have compared the effectiveness of CED [25, 24]; however, they still focus more on reliability, and there is little discussion on the fault models the attacker actually uses. **Whereas conventional CED targets randomly injected single bit transient, intermittent, and permanent faults, DFA uses a small class of single bit, deliberately injected multiple bit and byte transient faults at specific locations in the design and**

in specific instances during its operation. At first glance, conventional CED may appear to target DFA-exploitable faults as well. However, this is not the case. At best, they provide approximate fault coverage against naturally occurring multiple bit and byte faults[1]. These techniques typically miss the carefully constructed and deliberately injected multiple bit and byte faults. For example, while parity-based CEDs detect all random single bit faults that affect the parity, they can miss all carefully crafted multiple bit and multiple byte faults that are vulnerable to DFA.

1.1 Contributions

We analyze all DFA of AES and the security of four different types of CED. Our contributions are as follows:

- We propose the need for special CED for DFA of AES, in contrast to conventional CED.
- We analyze all DFA of AES and develop the inter-relationships of the DFA-exploitable faults.
- We analyze various CEDs for their strengths against DFA of AES and present counter-examples of DFA-exploitable faults which are missed by these CEDs.
- We present an invariance-based CED which is capable to detect provably all DFA-exploitable faults in 100% of the cases.

The rest of the paper is organized as follows: In Section 2, we introduce the AES algorithm, DFA attack procedure, and conventional CEDs. In Section 3, we summarize the fault models in previous DFA of AES and find out their internal relationships. In Section 4.1, we analyze the security of conventional CEDs against DFA. In Section 5, we propose the invariance-based CED and analyze its strength against DFA. In Section 6, we conclude the paper.

2 Preliminaries

2.1 AES Algorithm

AES is an iterative block cipher with key lengths of 128, 192, and 256. We consider 128-bit key for AES, but the conclusions apply to the other key sizes. AES encrypts a 128-bit input plaintext into a 128-bit output ciphertext with a 128-bit 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 $GF(2^8)$. Each byte is denoted by $\boldsymbol{s_{r,c}}$ $(0 \leq r, c \leq 3)$ indicating that this byte is in row $\boldsymbol{r}$ and column $\boldsymbol{c}$ in the state matrix.

[1]The fault coverage is estimated by simulating only a small subset of randomly injected multiple bit and byte faults.

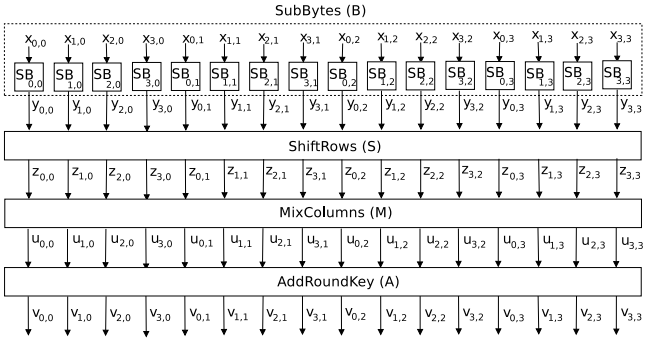


Figure 1: One AES encryption round.

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

In SubBytes, all bytes are processed separately by 16 S-boxes (SBs in Figure 1). Each SB performs a nonlinear transformation of the input byte. If X is the input, the output is:

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

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 one, two, and three bytes respectively. The resulting output is:

$$Z = S(Y) = \begin{bmatrix} y_{0,0} & y_{0,1} & y_{0,2} & y_{0,3} \\ y_{1,1} & y_{1,2} & y_{1,3} & y_{1,0} \\ y_{2,2} & y_{2,3} & y_{2,0} & y_{2,1} \\ y_{3,3} & y_{3,0} & y_{3,1} & y_{3,2} \end{bmatrix} = [y_{r,(r+c) \ mod \ 4}]_{r,c=0}^{3} = [z_{r,c}]_{r,c=0}^{3} \tag{3}$$

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

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

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

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

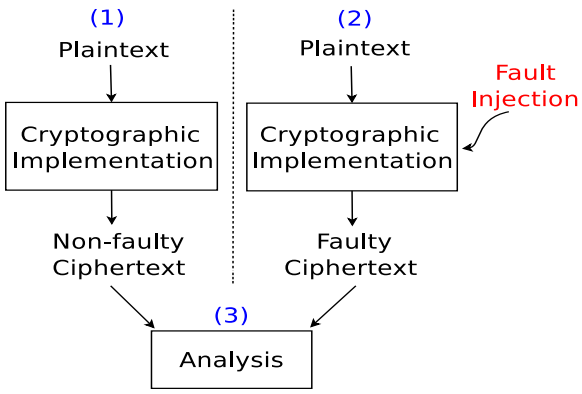


Figure 2: Three steps of DFA.

2.2 Differential Fault Analysis

DFA is applicable to almost any secret key cryptosystem proposed so far in the open literature. DFA has been used to attack many secret key cryptosystems, including DES, IDEA, and RC5 [9].

There has been considerable number of work about DFA of AES. Some of the DFA proposals are based on theoretical model [10, 35, 13, 34, 26, 31, 40, 36], while others launched successful attacks on ASIC and FPGA devices using previously proposed theoretical models [37, 21, 7, 2, 36]. The key idea of DFA is composed of three steps as shown in Figure 2. (1) Run the cryptographic algorithm and obtain non-faulty ciphertexts. (2) Inject faults, i.e., unexpected environmental conditions into cryptographic implementations, rerun the algorithm with the same input, and obtain faulty ciphertexts (3) Analyze relationship between the non-faulty and faulty ciphertexts to significantly reduce the key search space.

Practicality of DFA depends on the underlying fault model and the number of faulty ciphertext pairs needed. In Section 3, we will analyze all the fault models DFA uses and point out their relationships.

2.3 Concurrent Error Detection

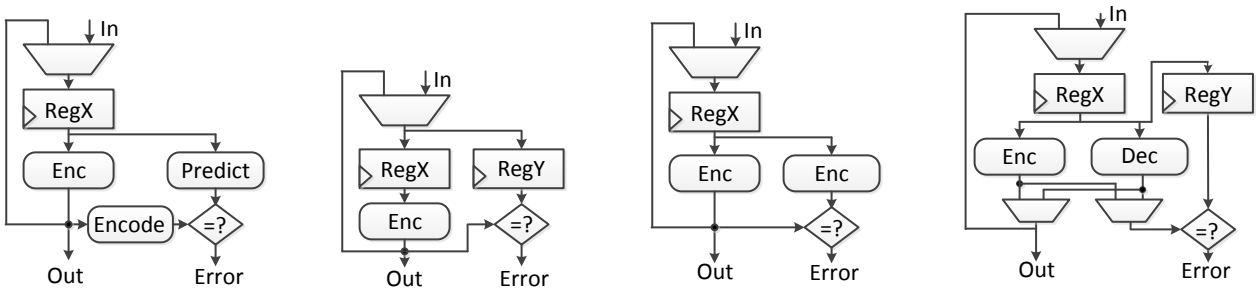
Previous work on CEDs can be classified into four types of redundancy: information, time, hardware, and hybrid redundancy.

2.3.1 Information Redundancy

Many CEDs are based on error detecting codes. In these techniques, the input message is encoded to generate a few check bits, and these bits are propagated along with the input message. The information is validated when the output message is generated. Three information redundancy techniques are discussed below:

Parity-1: [41] uses only single bit parity for the entire 128-bit output, and the parity bit is checked once for the entire round.

Parity-16: Another category of parity technique is proposed in [8, 29, 28]. In these techniques, one parity bit is generated for each byte of the input. While [8] and [29] apply to S-box implementation using look-up table (LUT) and finite field arithmetic, respectively. [28] gives a



(a) (b) (c) (d)

Figure 3: Four different CED techniques. (a) Information redundancy. (b) Time redundancy. (c) Hardware redundancy. (d) Hybrid redundancy.

general parity formation for all S-box implementations. While gaining higher fault coverage, the area overhead of Parity-16 is more than Parity-1.

Robust Code: The parity code suffers from nonuniform fault coverage [18], e.g., parity-1 cannot detect even number of faulty bits, and parity-16 cannot detect even number of faulty bits in each byte. To address the limitation of parity code, A systematic robust code is proposed [18]. It provides uniform fault coverage for all types of faults. The key idea is to construct a prediction circuit at the input of the round to predict nonlinear property of the output of the round. There are three components at the round output to extract the nonlinear property of the output, the compressor, linear compressor, and the cubic function. Each byte of the compressor output is $L(j)$ is equivalent to the componentwise XOR of four bytes of the same column. The prediction circuit is composed of a linear predictor, linear compressor, and a cubic function, each of which is the next stage of the previous one. The linear predictor will take the round key and the round input and generate a 32-bit output. Linear compressor and cubic function will reduce the 32-bit data into 28 bits. The output of the linear predictor $L_l(j)$ is the same as the output of the compressor.

2.3.2 Time Redundancy

In time redundancy, the function is computed twice on the same input, and the results are compared with each other as shown in Figure 3(b). One redundant encryption cycle is required to check each round, and this technique cannot detect permanent faults. A variation of the time redundancy is proposed in [23]. In this method, the function is computed on both clock edges to speed up the computation.

2.3.3 Hardware Redundancy

In hardware redundancy methods, the circuit is duplicated. As shown in Figure 3(c), both original and duplicated circuits are fed with the same inputs and the outputs are compared with each other [16].

Table 1: A summary of DFA of AES. ⋆ CT = ciphertext. † Only one byte in a word is faulty. ‡ Two or three bytes in a word are faulty. ◇ All four bytes in a word are faulty.

Fault Model		**No. of Faulty CTs** ⋆	**Key Search Space**	**Experiment**
3.1.1 Faults are injected in any location and any round				
Random		2^{128}	2^{128}	N/A
3.1.2 Faults are injected in AddRoundKey in round 0				
Single bit	[10]	128	1	No
3.1.3 Faults are injected between the output of 7^{th} and the input of 8^{th} round MixColumns				
Single byte	[35]	2	2^{40}	underpowering [37, 21]
	[31]	2	2^{32}	No
	[40]	1	2^{8}	No
Multiple byte DM0	[36]	1	2^{32}	overclocking [36]
Multiple byte DM1	[36]	1	2^{64}	
Multiple byte DM2	[36]	1	2^{96}	
Multiple byte DM3	[36]	2^{128}	2^{128}	
3.1.4 Faults are injected between the output of 8^{th} and the input of 9^{th} round MixColumns				
Single bit	[13]	≈ 50	1	overclocking [2]
Single byte	[34]	≈ 40	1	underpowering [7]
	[26]†	6	1	No
Multiple byte DM0	[26]‡	6	1	No
Multiple byte DM0	[26]◇	1500	1	No

2.3.4 Hybrid Redundancy

In [19], the authors consider a hybrid CED method in which at the operation, round, and algorithm levels for AES. A single operation, a round, or the entire encryption blocks are followed by their inverses. The detail is shown in Figure 3(d). In this technique, the results are compared with the original input. Although most of the faults are detected by this technique, both encryption and decryption blocks have to be added to the chip. Therefore, this technique suffers from more than 100% overhead.

3 DFA and Associated Fault Models

In this section, we classify the fault models used in DFA and analyze their relationship. We also conduct a practical fault attack to demonstrate the practicality of the most general fault model in [36].

3.1 DFA of AES: Fault Models

DFA exploits a small subspace of all possible faults. Moreover, DFA-exploitable faults are transient and mostly multiple bit and byte faults. Transient faults can leak the information of the key in a

stealthy way, because their presence is temporary. This implies that fault models such as stuck at faults, are not relevant for DFA. Further, the fault injection techniques are not random, rather, they are biased depending on the region in which the DFA works. Hence, it is important to develop suitable CED with this perspective and to prove formally that all DFA-exploitable faults are detectable. Table 1 is a summary of the published DFA of AES. We classify the DFA fault models in four scenarios by the location and round in which faults are injected. Faults can be injected either (I) in any location and any round, (II) in AddRoundKey in round 0, (III) between the output of 7^{th} and the input of 8^{th} round MixColumns, or (IV) between the output of 8^{th} and the input of 9^{th} round MixColumns. In each scenario, we analyze the (A) fault models, (B) number of faulty ciphertexts needed, (C) the key search space for brute force after obtaining the faulty outputs to recover the secret, and (D) the experimental validation of the attack. The considered transient faults are categorized into single bit, single byte, and multiple byte transient faults.

3.1.1 Faults Are Injected in Any Location and Any Round

In the first fault model in Table 1, the attacker injects faults in any random location and any random round. These faults are equivalent to naturally occurring faults. In this case, the attacker will not gain any useful information. Even if he gets all possible 2^{128} faulty ciphertexts, he cannot reduce the key search space. Because the key search space is 2^{128}, this fault model is impractical for the attacker.

3.1.2 Faults Are Injected in AddRoundKey in Round 0

The only fault model an attacker uses in this scenario is single bit transient fault.

Single bit transient fault: In [10], the attacker is able to set or reset every bit of the first round key one bit at a time. This attack recovers the entire key using 128 faulty ciphertexts with each faulty ciphertext uniquely revealing one key bit. Hence, the key search space required to reveal the key is one. However, as transistor size scales, this attack becomes impractical even with expensive equipments such as lasers to inject the faults, because it requires precise control of the fault location [1].

3.1.3 Faults Are Injected between the Output of 7^{th} and the Input of 8^{th} MixColumns

The attacker uses various fault models and analysis in this scenario including single and multiple byte fault.

Single byte transient fault: The three different attacks using this fault model are shown in Table 1. In the first DFA [35], two faulty ciphertexts are needed to obtain the key. This fault model is experimentally verified in [37, 21]. In [37], underpowering is used to inject faults into a smart card with AES ASIC implementation. Although no more than 16% of the injected faults fall into the single byte fault category, only 13 faulty ciphertexts are needed to obtain the key. In [21], the authors underpower an AES FPGA implementation to inject faults with a probability of 40% for single byte fault injection.

In the second DFA [31], two faulty ciphertexts are also needed to reveal the key. Because this attack exploits the faults in a different way, the key search space is 2^{32}.

The attack in [40] is similar to [31], but further improved. For the same fault model, the key search space is reduced to only 2^8 with a single faulty ciphertext.

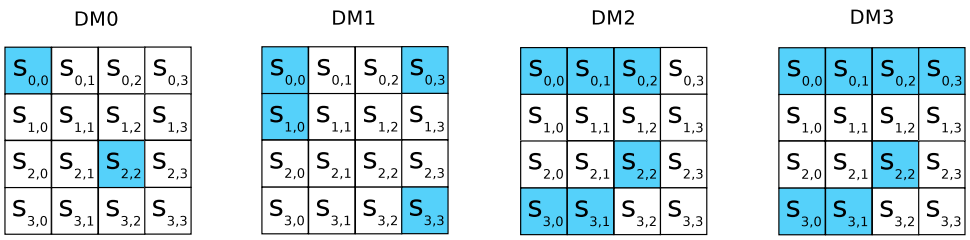


Figure 4: DFA and diagonal fault models. The first state matrix is an example of DM0. Only diagonal D_0 is affected by a fault. The second state matrix is an example of DM1. Both D_0 and D_3 are corrupted by a fault. The third state matrix is an example of DM2. Three diagonals D_0, D_1, and D_2 are corrupted by a fault. The last state matrix is an example of DM3, all four diagonals are corrupted in the fourth state matrix.

Multiple byte transient fault: [36] proposes a general byte fault model called **diagonal fault model**. The authors divide the AES state matrix into four different diagonals and each diagonal has four bytes. **A diagonal** is a set of four bytes of the AES state matrix, where the i^{th} diagonal is defined as follows:

$$D_i = \{s_{j,(j+i)mod4} \ ; \ 0 \leq j < 4\} \tag{6}$$

We obtain the following four diagonals.

$$D_0 = (s_{0,0}, s_{1,1}, s_{2,2}, s_{3,3}), \ D_1 = (s_{0,1}, s_{1,2}, s_{2,3}, s_{3,0}),$$

$$D_2 = (s_{0,2}, s_{1,3}, s_{2,0}, s_{3,1}), \ D_3 = (s_{0,3}, s_{1,0}, s_{2,1}, s_{3,2})$$

The diagonal fault model is classified into four different cases, diagonal fault models 0, 1, 2, and 3, denoted as DM0, DM1, DM2, and DM3. As shown in Figure 4, for DM0, faults can be injected in one of the diagonals; D_0, D_1, D_2, or D_3. For DM1, faults can be injected in at most two diagonals. For DM2, faults can be injected in at most three diagonals. Finally, for DM3, faults can be injected in at most four diagonals. For each of the four different diagonals affected, faults propagate to different columns as shown in Figure 5. Therefore, if faults are injected into one, two, or three diagonals, the key search space is reduced to 2^{32}, 2^{64}, or 2^{96}, respectively. The authors also validate the diagonal fault model with a practical fault attack on AES FPGA implementation using overclocking. In Section 3.3, we also perform a fault injection experiment to validate this general fault model.

3.1.4 Faults Are Injected between the Output of 8_{th} and the Input of 9_{th} MixColumns

Single bit transient fault: In [13], the attacker needs only three faulty ciphertexts to succeed with a probability of 97%. The key search space is trivial. [2] validates this single bit attack on a Xilinx 3AN FPGA using overclocking. It is reported that the success rate of injecting this kind of fault is 90%.

Single byte transient fault: In [34], the authors use a byte level fault model. They are able to obtain the key with 40 faulty ciphertexts, and the key is uniquely revealed. This model is used in a successful attack by underpowering a 65nm ASIC chip [7]. In this attack, 39881 faulty

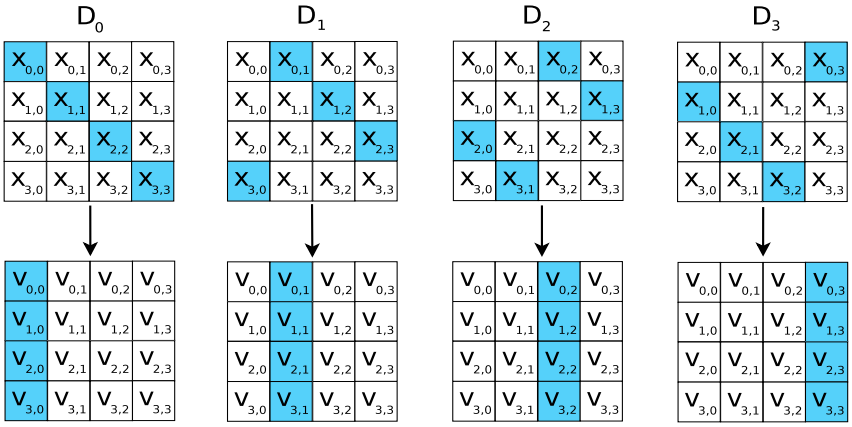


Figure 5: Fault propagation of diagonal faults. The upper row shows the diagonals that faults are injected in. The lower row shows the corresponding columns being affected.

ciphertexts are collected during the 10 experiments; 30386 of them were actually the outcome of a single byte fault. Thus, it has a successful injection rate of 76%.

Multiple byte transient fault: [26] presents a DFA of AES when the faults are injected in a 32-bit word. The authors propose two fault models. In the first model, they assume that at least one of the bytes among the four targeted bytes is non-faulty. This means the number of faulty bytes can be one, two, or three bytes. So this fault model includes the single byte fault model. If only one single byte fault is injected, six faulty ciphertexts are required to reveal the secret key. Whereas the second fault model requires around 1500 faulty ciphertexts. These faulty ciphertexts derive the entire key at constant time. Though the second fault model is much more general, the amount of faulty ciphertexts it requires is very large, it is difficult for the attacker to get all the ciphertexts without triggering the CED alarm.

In summary, the attacker can obtain the secret key with one or two faulty ciphertexts when single or multiple byte transient faults are injected. **Therefore, the CED should have 100% fault coverage, because even a single missed fault can be sufficient to leak the key, rendering the CED useless**.

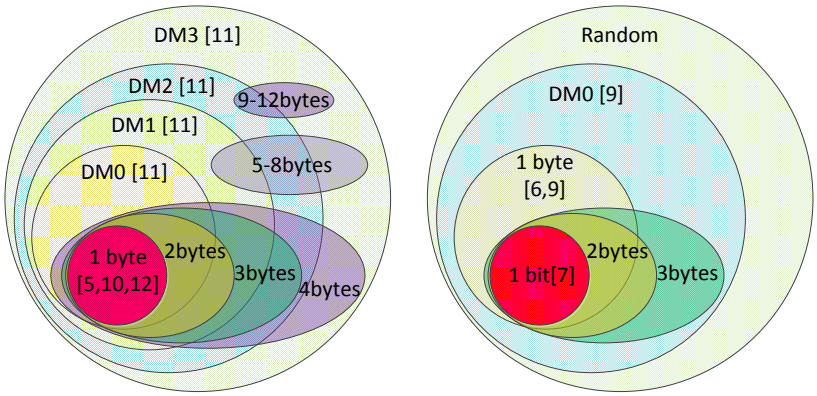
3.2 Relationships between the Discussed Fault Models

As previously mentioned, DFA of AES does not exploit all possible faults. Rather, it exploits a subset of faults, namely single bit, single byte, and multiple byte transient faults injected in selected locations and rounds. Therefore, understanding the relationships between various fault models is the basis for analyzing the security of conventional and new CED as well as for designing new DFA-specific CED. Because DFA of AES targets the last few rounds[2], we synthesize the relationships between different fault models based on the locations and rounds they are injected in. The goal is to identify the fault space for which 100% fault coverage is necessary.

3.2.1 Faults Are Injected in Any Location and Any Round

In this fault model, the attacker cannot derive any useful information from the faults.

[2]In general, the practical faults used in DFA target the 7^{th}, 8^{th}, and 9^{th} rounds.



(a) (b)

Figure 6: Relationships between DFA fault models when faults are injected between (a) the output of 7^{th} and the input of 8^{th} round MixColumns, (b) output of 8^{th} and the input of 9^{th} round MixColumns.

3.2.2 Faults Are Injected in AddRoundKey in Round 0

As we mentioned previously, this attack uses a very restricted fault model, and it is not practical. Thus, this fault model is also not useful for the attacker.

3.2.3 Faults Are Injected between the Output of 7^{th} and the Input of 8^{th} MixColumns

Figure 6a summarizes the relationships between the DFA-exploitable fault models by injecting faults in the output of 7^{th} round MixColumns and the input of 8^{th} round MixColumns.

Single byte faults are, in turn, a subset of the DM0 faults which, in turn, are a subset of the DM1 faults, and so on. The relationship is summarized in (7).

$$\text{Single Byte} \subset DM0 \subset DM1 \subset DM2 \subset DM3 \tag{7}$$

A more careful look reveals that two byte faults can be either DM0 or DM1 but not DM0. Moreover, three byte faults can be either DM0, DM1, or DM2. Four byte faults can be either DM0, DM1, DM2, or DM3. Similarly, the relationship between faulty bytes from 5 to 12 and diagonal fault models are summarized in Figure 6(a).

Such an analysis of the fault classes will enable one to clearly determine the capabilities of a given CED technique. As shown in Figure 6(a), DM3 includes all possible byte transient faults. The attacks proposed in [36] show that DFA based on DM0, DM1, and DM2 leads to the successful retrieval of the key. Remember that DM3 faults are the universe of all possible transient faults injected in the selected AES round. These faults spread across all four diagonals of the AES state and hence, are not vulnerable to DFA as mentioned in Section 3.1.3. These fault models are multiple byte transient faults and thus, attacks based on these models are more feasible than those based on single byte transient faults, which are a subset of the model DM0. The considered fault models are vulnerable to DFA in the following order: (i) the DM0 type faults reduce the key space of AES to 2^{32}, (ii) the DM1 type faults reduce the key space to 2^{64}, and (iii) the DM2

type faults reduce the key space to 2^{96} after a single fault induction. The more encompassing the fault model is, the more realistic the attacks based on it are. Consider the cardinalities of the identified fault classes, the number of possible DM0, DM1, and DM2 faults are 2^{34}, 3×2^{65}, and 2^{98}, respectively. The number of possible DM3 faults is 2^{128} in the state matrix[3]. If all faults are equiprobable during injection (this is the perspective of conventional fault injection and detection studies), the probability of injecting DM0, DM1, and DM2 faults is negligible. The probability that a randomly injected fault is a DM0, DM1, or DM2 type fault is 2^{-94}, $\frac{1}{3} \times 2^{-63}$, and 2^{-30}, respectively. However, we stress that a DFA attacker does not use uniformly distributed fault injection. Rather, he characterizes the device and uses specific fault injections which results in high success rates. We prove this concept in our experiment in Section 3.3. Thus, it is important to develop error detection techniques that consider this bias towards DFA-exploitable faults. This is not the case with conventional CED techniques. Conventional CED techniques, such as parity, do not provide a 100% fault coverage for the DFA-exploitable faults injected between the $7^{th} - 9^{th}$ rounds. This will be described in Section 4.

3.2.4 Faults Are Injected between the Output of 8^{th} and the Input of 9^{th} MixColumns

Figure 6(b) summarizes the relationships between the DFA-exploitable fault models by injecting faults in the output of 8^{th} and the input of 9^{th} round MixColumns. Single bit transient faults are a subset of single byte faults. Single byte faults are again a subset of DM0 faults. Two and three byte faults are a subset of DM0 faults. Again, attacks based on multiple byte faults are more feasible than those based on single bit and single byte faults.

Until now, we highlighted the importance of understanding the fault models for AES. As a designer, we must develop countermeasures for these fault models which, thus, are actually exploitable in practical DFA. We point out that unlike conventional error detection architectures, DFA countermeasures must focus on the smaller class of faults and leading to opportunities for 100% provable coverage with respect to the targeted DFA. In the following section, we present a case study of fault injection through a simple laboratory set-up.

3.3 Practicality of DFA

Some fault attacks can be realized by using simple laboratory set-ups such as the one shown in Figure 8a. The set-up is comprised of a function generator hooked up to a Xilinx Spartan-3E FPGA device on which the AES runs. The design operates correctly when using the ***Slow Clock*** with a clock frequency of 72MHz. When the clock is switched to the ***Fast Clock***, which has a clock frequency of 85-120MHz, at the beginning of the 8^{th} round encryption, it creates critical path violations inside the circuit.

We injected several faults into the AES by overclocking the system in increments of 1MHz. For each overclocking step, we ran 512 encryptions and collected the samples through Xilinx ChipScope Pro [42]. The distribution of the observed faults is shown in Figure 8(a). When the system is only slightly overclocked, i.e., when the Fast Clock is in the range of 85-92MHz, we see that DM0 faults are injected. When the frequency of the Fast Clock is gradually increased in the

[3]The number of faults is calculated based on a simple assumption that the faults are injected at the input to the round. If the faults can be injected anywhere in the AES round, all of these numbers can be proportionally scaled. Further, this is ignoring all permanent and intermittent faults as they are not exploitable from a DFA perspective.

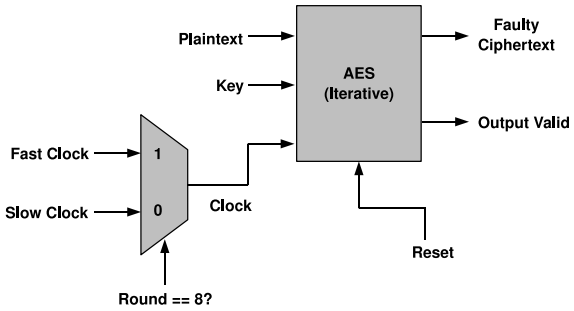
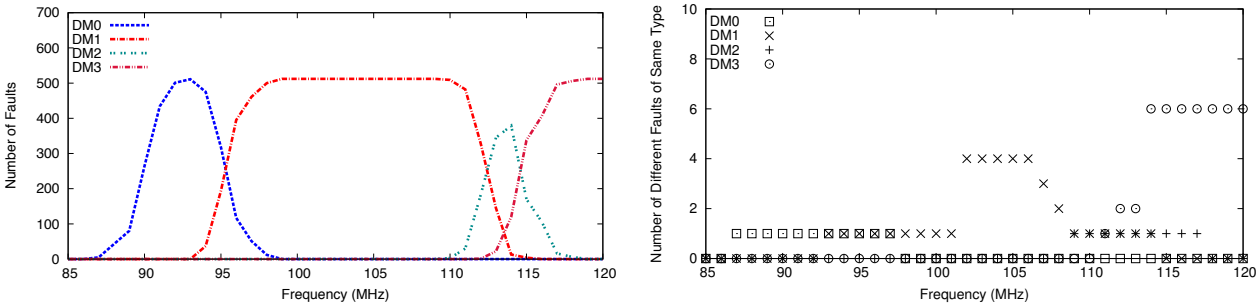


Figure 7: Fault injection set-up to launch faults according to the diagonal fault models.



(a) (b)

Figure 8: Fault injection results. (a) Number of DM0, DM1, DM2, and DM3 faults as a function of the overclocking frequency. (b) Number of different DM0, DM1, DM2, and DM3 faults as a function of the overclocking frequency.

range of 98-108MHz, DM1 faults are injected. In the range of 109-110MHz, we observe DM2 faults being injected. We can also see that there are clearly identifiable regions where DM0, DM1, or DM2 faults can be injected. If the clock frequency is beyond 111MHz, the probability of injecting the DM3 faults increases, and all four diagonals will be affected.

The number of different types of faults is shown in Figure 8(b). This figure shows fault injections succeed within the 512 fault injections and the type of faults injected. and it shows that the fault injections have a deterministic nature. When the Fast Clock is in the range of 85-92MHz, the injected DM0 faults are the same. When the frequency of the Fast Clock is gradually increased in the range of 93-97MHz, DM0 and DM1 faults are injected. Similarly, the DM0 faults in this frequency range are all the same. A similar observation also applies to DM1 faults. In the range of 98-101MHz, we observe DM1 faults being injected. In the range of 102-108MHz, different DM1 faults are injected. In the range of 109-114MHz, DM1, DM2, and DM3 faults are different. Between 115-120MHz, most of those faults injected are DM3 faults.

This experiment was repeatable on different AES implementations with different plaintexts and keys. The general observations are quite similar and are summarized as follows:

- At the beginning of the fault injection, we can inject single byte faults that can be exploited by the attacks proposed in [35, 31, 36].
- When the system is highly overclocked, DM3 faults dominate. These faults and the resulting

faulty ciphertexts are not vulnerable to DFA.

- Between the two extremes discussed above, DM0, DM1, and DM2 faults are injected. DM0 faults reduce the key space of AES to 2^{32}, DM1 to 2^{64}, and DM2 to 2^{96}.

These observations show that the fault distribution is not uniform and is biased depending on the operating region of the device, which the attacker can control. Because conventional CED assumes that all faults are equally probable, there is a significant gap between conventional CED and CED for DFA. Thus, the attacker can accordingly characterize a device and perform directed variations in the operating conditions, e.g., overclocking, to inject exploitable faults and exploit them for DFA with a very high chance of success. From a designer's perspective, we would like to develop countermeasures that prevent such reductions in key search space or detect such faults so that no exploitable information about the key is leaked.

From this summary, we can see that the attacker can obtain the secret key easily with only two or even one faulty output with the respective practical fault model. Therefore, 100% fault coverage for these fault models is necessary for CEDs.

As shown in Figure 6a, we can see that there are random, single bit, single byte, and multiple byte fault models, each of which is a subset of the previous one. From the summary of DFA, it is obvious that the attacker cannot exploit all kinds of faults. Only single bit, single byte, and multiple byte faults as mentioned above are exploitable. Because only a small subset of the random faults are vulnerable to attack, the CED should classify different fault models and give provable 100% fault coverage for the fault models that are vulnerable to the attacker. Otherwise, even very high fault coverage for random faults may compromise the security of AES. We look into characterizing the various CED techniques in their ability to detect the faults in these different DFA-exploitable classes of faults.

4 Security Analysis and Fault Coverage

In this section, we analyze the security of the conventional CEDs and point out their fault coverage decrease significantly when DFA-exploitable faults are injected.

4.1 Conventional CED VS CED for DFA

Table 2 compares the fault coverage of different CEDs. One of the shortcomings of conventional CEDs is that they are mostly tailored to detect randomly injected faults, i.e., they generally do not prioritize DFA-exploitable faults. On the other hand, DFA attacks do not benefit from all possible faults, i.e., the secret key can be broken by injecting specific faults. Accordingly, to equip a cryptographic circuit with a CED technique aiming at detecting the deliberately inserted faults and preventing secret key leakage, the overall fault coverage of the circuit should not be considered; i.e., only the coverage of a specific set of faults from which the attacker can break the secret key, should be considered. Security is more important than reliability in cryptographic hardware. Even by applying a CED technique that detects 99% of faults, the secret key can still be leaked by repeating an experiment 100 times. Accordingly, the fault coverage should be 100% to defend against DFAs.

Table 2: Fault coverage of different CED techniques against natural and DFA faults ¶ Information redundancy. § Time redundancy. ⋆ Hardware redundancy. ♯ Hybrid redundancy. ◇ Invariance-based CED. † Only one byte in a word is faulty. ‡ Two or three bytes in a word are faulty. ℵ All four bytes in a word are faulty. ♭ This technique provides close to 100% fault coverage ($1 - 2^{-56}$).

CED		Random	Single bit		Single byte					Multiple byte				
			[10]	[13]	[35]	[31]	[34]	[40]	[26]†	[26]‡	[26]ℵ	[36]		
												DM0	DM1	DM2
Info. Red. ¶	[41]	48-53	100	100	50	50	50	50	50	50	50	50	50	50
	[8]	99.997	100	100	50	50	50	50	50	75–87.5	93.75	50–93.75	75–99.61	87.5–99.98
	[29]	97.035	100	100	50	50	50	50	50	75–87.5	93.75	50–93.75	75–99.61	87.5–99.98
	[28]	99.996	100	100	50	50	50	50	50	75–87.5	93.75	50–93.75	75–99.61	87.5–99.98
	[18]♭	<100	<100	<100	<100	<100	<100	<100	<100	89.99	99.6	99.6	99.94	99.936
Time Red. §	[25]	100	100	100	100	100	100	100	100	100	100	100	100	100
H.W. Red. ⋆	[16]	100	100	100	100	100	100	100	100	100	100	100	100	100
Hybrid Red. ♯	[19]	100	100	100	100	100	100	100	100	100	100	100	100	100
Invar. CED ◇	[14]	99.999	100	100	100	100	100	100	100	100	100	100	100	100

We analyze the fault coverage of three various CEDs against DFA. Fault coverage (FC) is calculated as:

$$FC = 1 - \frac{T_{undetected}}{T_{total} - T_{correct}}$$

where T_{total} is the total number of experiments, $T_{undetected}$ is the number of experiments in which faults are excited but not detected, and $T_{correct}$ represents the number of experiments in which the faults are not excited.

4.1.1 Security Analysis of Parity-1

Parity-1 considers one parity bit for the 128-bit state matrix. This parity bit provides 50% fault coverage for the data it protects. Apparently, it provides 100% fault coverage against single bit faults. However, the technique cannot detect any even number of faults. The fault coverage is $1 - 50\% = 50\%$. For parity-1, the value of the parity bit is not affected by the location and the type of faults; fault coverage for the single byte and multiple byte faults are equal.

As we discussed in section 3, an attacker needs only one or two faulty outputs to break the secret key. Therefore, by using parity-1 technique with 50% fault coverage, the attacker can access the secret key by at most four experiments. Accordingly, parity-1 does not provide enough security against DFA.

DM0 fault counterexamples for parity-1: Assume the input of the 8^{th} round is X. The parity of the input is $Parity(X)$. Therefore, the parities after SubBytes, ShiftRows, and MixColumns are $Parity(Y)$, $Parity(Z)$, and $Parity(U)$, respectively. Because ShiftRows and MixColumns do not change the parity of the state matrix [41], $Parity(Y) = Parity(Z) = Parity(U)$. The key of this AES round is K. The parity of the key is $Parity(K)$. The predicted output parity is $Parity(U) \oplus Parity(K)$. Because the output of this round is V, the actual output parity is $Parity(V)$. When there is no fault, $Parity(V) = Parity(U) \oplus Parity(K)$. When there is an

attacker injects faults into the chip, let the faults affect any even number of bits in one diagonal of the input to MixColumns. The faulty input of MixColumns becomes Z^f. For this case, the outputs of the MixColumns and AddRoundKey become U^f and V^f. Because changing an even number of bits does not affect the parity, the parity of the faulty input equals the parity of the correct input $Parity(Z^f) = Parity(Z)$. Recall that MixColumns does not change the parity of the state matrix [41], thus, $Parity(U^f) = Parity(Z^f)$. The output parity of the AddRoundKey then becomes $Parity(V^f) = Parity(U^f) \oplus Parity(K)$. Because $Parity(Z^f) = Parity(Z)$, we know $Parity(V^f) = Parity(V)$. Therefore, parity-1 cannot detect an even number of DM0 faults. Similar to the previous counterexample, if any even number of faults affect multiple diagonals at the input of the MixColumns, the faults will be detected because $Parity(Z^f) = Parity(Z)$ still holds true.

4.1.2 Security Analysis of Parity-16

In this technique, a parity bit is generated for every eight bit data in the state matrix. Similar to parity-1, parity-16 provides 100% fault coverage for single bit faults. For single byte faults, the probability of detecting a fault is 50%, while for multiple byte faults, the probability of detecting a fault equals to one minus the probability of detecting none of the faults in each byte. Therefore, the fault coverage is:

$$FC = 1 - (1 - 50\%)^n \tag{8}$$

where n is the number of faulty bytes. Obviously this technique is also insecure against the DFA described in [26, 36, 40, 31].

Using (8), when exactly three bytes are faulty, the fault coverage is 87.5%. Similarly, for a four byte fault, the fault coverage is 93.75%. Both of these cases show security provided by parity-16 is insufficient. In the former case, an attacker can succeed with eight experiments, while in the latter, the key can be obtained by 16 experiments.

For the diagonal fault model DM0, a fault can affect from one to four bytes of data in a diagonal. When four bytes are affected, parity-16 has the highest fault coverage (93.75%), while the fault of this technique is 50% when only one byte is affected. On average, fault coverage is

$$\frac{\textit{Detectable faults}}{\textit{All possible faults}} = \frac{(C_4^4 \times 2^{32} - C_4^3 \times 2^{24} - C_4^2 \times 2^{16} - C_4^1 \times 2^8) \times 93.75\%}{2^{32}} +$$

$$+ \frac{C_4^3 \times 2^{24} \times 87.5\% + C_4^2 \times 2^{16} \times 75\% + 4 \times 2^8 \times 50\%}{2^{32}} \approx 93.74\%$$

It is close to the best case fault coverage. But as we have pointed out, the attacker will aim for the fault that is more likely to escape from the CED. For DM1 and DM2 faults, n changes between two to eight and three to 12, respectively. Accordingly, the highest fault coverage for detecting DM1 and DM2 faults using parity-16 is 99.61% and 99.98%, respectively. However, the lowest fault coverage while using parity-16 to detect DM1 and DM2 faults is 75% and 87.5%, respectively. Accordingly, an attacker only requires four times to break the secret key if the DM1 faults are carefully crafted. The number of experiments are eight while DM2 faults are injected.

DM0 fault counterexample for parity-16: Assume the input of the 8^{th} round is $[x_{r,c}]_{r,c=0}^3$. The parity bits of the input are $[p_{r,c}^{in}]_{r,c=0}^3$. The key input of this AES round is $[k_{r,c}]_{r,c=0}^3$. The parity bits of the key are $[p_{r,c}^{k}]_{r,c=0}^3$. The parity bits of the output of the S-box are $[p_{r,c}^{B}]_{r,c=0}^3$.

ShiftRows only cyclically shifts the parity bits. Therefore, the output parity bits of the ShiftRows becomes $[p^S_{r,c}]^3_{r,c=0} = [p^B_{r,(r+c)\ mod\ 4}]^3_{r,c=0}$. After MixColumns, the parity becomes $[p^M_{r,c}]^3_{r,c=0}$. Finally, the predicted parity bits are $[p^{pre}_{r,c}]^3_{r,c=0} = [p^M_{r,c}]^3_{r,c=0} \oplus [p^k_{r,c}]^3_{r,c=0}$, which matches the actual parity $[p^K_{r,c}]^3_{r,c=0}$. An attacker can flip an even number of bits in any number of bytes in the same diagonal. Then the parity bit of that faulty byte is the same as the nonfaulty one. Therefore, the countermeasure is successfully defeated.

Similarly, we can easily find out that if any even number of bits in arbitrary number of output bytes are flipped by the attacker, parity-16 will not detect it.

4.1.3 Security Analysis of Robust Code

Robust code is designed to address the nonuniform fault coverage problem of linear code such as parity. By using the nonlinear robust code with r check bits, the percentage of undetectable faults is reduced from 2^{-r} to 2^{-2r} compared to the parity code, and it provides uniform fault coverage regardless of the fault distribution [3]. In [18], r is 28 because of the hardware overhead is low for the cubic network implementation. The fault coverage of this technique is

$$1 - 2^{-56} < 100\%$$

However, in the following we will show that a bias fault injection will also significantly reduce the fault coverage of this technique.

DM0 fault counterexamples for robust code: Let the input of the round be X. Then, the output of ShiftRows is Z, and the output of the MixColumns is U. Therefore $L(j) = [u_{i,j}]^3_{i=0} \oplus [k_{i,j}]^3_{i=0}$. Assume e_0, e_1, e_2, and e_3 are the DM0 fault in each bytes in the diagonal at the input of ShiftRows which is equivalent to a column at the input of MixColumns. If $e_0 \oplus e_1 \oplus e_2 \oplus e_3 = 0$, then the robust code fails. The reason is explained as below. The faulty bytes occur at the input of MixColumns. They are $z'_{0,j} = z_{0,j} \oplus e_0$, $z'_{1,j} = z_{1,j} \oplus e_1$, $z'_{2,j} = z_{2,j} \oplus e_2$, and $z'_{3,j} = z_{3,j} \oplus e_3$, respectively. From (4), we know that at the input

$$\begin{aligned}
[u'_{i,j}]^3_{i=0} = {} & 02 \cdot z'_{0,j} \oplus 03 \cdot z'_{1,j} \oplus z'_{2,j} \oplus z'_{3,j} \oplus z'_{0,j} \oplus 02 \cdot z'_{1,j} \oplus 03 \cdot z'_{2,j} \oplus z'_{3,j} \oplus \\
& z'_{0,j} \oplus z'_{1,j} \oplus 02 \cdot z'_{2,j} \oplus 03 \cdot z'_{3,j} \oplus 03 \cdot z'_{0,j} \oplus z'_{1,j} \oplus z'_{2,j} \oplus 02 \cdot z'_{3,j} \\
= {} & 02 \cdot z_{0,j} \oplus 02e_0 \oplus 03 \cdot z_{1,j} \oplus 03e_1 \oplus z_{2,j} \oplus e_2 \oplus z_{3,j} \oplus e_3 \\
& \oplus z_{0,j} \oplus e_0 \oplus 02 \cdot z_{1,j} \oplus 02e_1 \oplus 03 \cdot z_{2,j} \oplus 03e_2 \oplus z_{3,j} \oplus e_3 \\
& \oplus z_{0,j} \oplus e_0 \oplus z_{1,j} \oplus e_1 \oplus 02 \cdot z_{2,j} \oplus 02e_2 \oplus 03 \cdot z_{3,j} \oplus 03e_3 \\
& \oplus 03 \cdot z_{0,j} \oplus 03e_0 \oplus z_{1,j} \oplus e_1 \oplus z_{2,j} \oplus e_2 \oplus 02 \cdot z_{3,j} \oplus 02e_3 \\
= {} & z_{0,j} \oplus z_{1,j} \oplus z_{2,j} \oplus z_{3,j} = [u_{i,j}]^3_{i=0}
\end{aligned}$$

Therefore, $L'(j) = [u'_{i,j}]^3_{i=0} \oplus [k_{i,j}]^3_{i=0} = L(j)$. The faults will not be detected. Hence, the probability of detecting such kind of fault is $1 - \frac{1}{256} = 99.6\%$.

Similarly, we can easily find out that if the fault affect the same bit position in even number of byte quantities, and these byte quantities move to the same column before MixColumns, then the robust code technique will not detect it.

4.2 Detecting All DFA-exploitable Faults is Important

As we observe, though most of the CEDs provide high fault coverage, the fault coverage is not 100%. Further, the above experimental result shows that an attacker has directed fault injection capabilities as opposed to the uniform fault distribution assumed in conventional CEDs. While aiming to detect all faults in other VLSI designs may be an objective, for cryptographic circuits, however, security is more important than reliability. This is because an attacker can characterize the device and operate it at a region where the faults are missed by the CED with a much higher probability than if the faults were random. This obviates the necessity of CEDs which have provable 100% security, w.r.t., all the DFA-exploitable faults. The previous discussions in Section 3, also shows that the DFA exploitable faults are a small subspace of the entire fault space. Hence a CED architecture for DFA should provide provable security for all the DFA-exploitable faults. In the next section, we propose such a scheme based on invariance-based CED, with provable security against all the DFA-exploitable faults.

5 Invariance-based CED

In this section, we elaborate an invariance-based CED technique, which was proposed in [14]. We prove that this technique detects all faults which are exploitable by DFA.

In this technique, the authors discuss that for each AES round i represented by $A(K, (M(S(B(S)))))$, at least one byte permutation for the input S exists such that

$$A(K, (M(S(B(S))))) = P^{-1}(A(P(K), (M(S(B(P(S))))))) \tag{9}$$

where S is the 128-bit state input of round i, P is a permutation, and P^{-1} denotes the inverse function of P. The authors show that one of the byte permutations is as below:

$$P_1(S) = P_1([s_{r,c}]^3_{r,c=0}) = \begin{bmatrix} s_{0,3} & s_{0,0} & s_{0,1} & s_{0,2} \\ s_{1,3} & s_{1,0} & s_{1,1} & s_{1,2} \\ s_{2,3} & s_{2,0} & s_{2,1} & s_{2,2} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{bmatrix} = [s_{r,(c+3)\ mod\ 4}]^3_{r,c=0} \tag{10}$$

$$P_1^{-1}([s_{r,(c+3)\ mod\ 4}]^3_{r,c=0}) = [s_{r,c}]^3_{r,c=0} \tag{11}$$

5.1 CED Architecture

The architecture of the invariance-based CED is shown in Figure 9, Similar to the regular time redundancy technique, one encryption cycle is required to check each round. Let us called the normal round and the check round C1 and C2. During C1, a regular encryption is performed with input state matrix X and key matrix K, and the encryption result V is stored in RegY. During C2, the permuted input $X^{'}$ and $K^{'}$ are used as for encryption. At the end of this cycle, the output $V^{'}$ is inverse permuted and compared with the value stored in the data register ($V1$). If the results are equal, no fault is detected. Otherwise, the error detection signal is raised. In this method, C1 can be any normal encryption cycle, and C2 is the corresponding extra cycle, during which the permuted inputs are used.

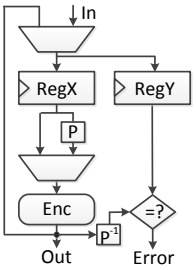


Figure 9: Invariance-based CED.

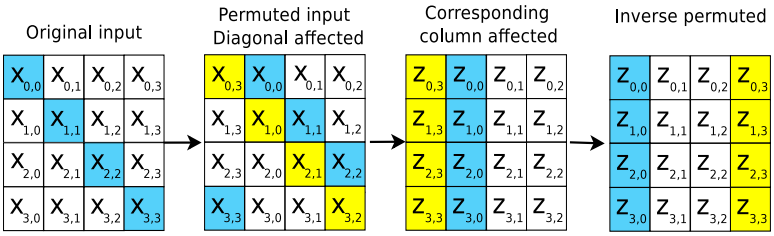


Figure 10: Diagonal fault propagation in the invariance-based CED when diagonal D0 is faulty.

5.2 Security Analysis of the Invariance-based CED

In [14], the authors prove that the invariance-based CED provides 100% fault coverage for single bit and single byte faults. we prove that this technique also provides 100% fault coverage for the diagonal faults of DM0, DM1, and DM2 models. An attacker cannot benefit from injecting DM3 faults to break the secret key. Accordingly, we did not include this fault model in our discussion.

Theorem 1 *For the diagonal fault model, at least one column differs between the original computation and the permuted computation.*

Proof We prove this theorem for DM0, DM1, and DM2 fault models. The theorem holds true for DM3 faults too but since injecting DM3 faults are not useful in breaking the secret key, we do not discuss those faults.

Case 1: DM0 faults. Assume that a fault affects only the diagonal of D_j in C1 (the first encryption cycle used for CED), where $D_j = x_{i,(i+j)\ mod\ 4}$ $(0 \leq i \leq 3,\ 0 \leq j \leq 3)$. Accordingly, $S_{i,(i+j)\ mod\ 4}$ generates faulty output(s) of $y_{i,(i+j)\ mod\ 4}$. After performing ShiftRows, the outputs are $[z_{r,c}]^3_{r,c=0} = [y_{r,(r+c)\ mod\ 4}]^3_{r,c=0}$, and the faulty state elements are $[z_{r,j}]^3_{r=0} = y_{i,(i+j)\ mod\ 4}$. In MixColumns, a fault is propagated from a single faulty input to all the state elements residing in the same column, i.e., $[u_{r,j}]^3_{r=0}$. After AddRoundKey, $[v_{r,j}]^3_{r=0}$ are the faulty state elements. However, in C2 (the second encryption cycle used for CED), we use X' and K' as the permuted inputs. Using the same steps shown above, faulty state elements are represented as $[v'_{r,j}]^3_{r=0}$. From equation (11), we know that

$$[v'_{r,j}]^3_{r=0} = [v_{r,(j+3)\ mod\ 4}]^3_{r=0} \tag{12}$$

Assume that that the faulty column in C1 is the column j. Considering the above equation, the faulty column in C2 is the column $(j+3)\ mod\ 4$, where $0 \leq i, j \leq 3$. Therefore, $j \neq (j+3)\ mod\ 4$.

As an example, consider Figure 5. Assume that in the normal round, diagonal D_0 is faulty. In C1, after performing SubBytes, ShiftRows, MixColumns, and AddRoundKey, column 0 holds the faulty bytes. Figure 10 shows the effect of the faults on the permuted round, using the proposed CED. As shown in this figure, the proposed CED changes the location of the diagonal affected by the fault. Accordingly, in the normal operation diagonal 0 is affected, while in the CED-related operation (performed in C2) column 3 is affected, and column 0 is non-faulty.

Case 2: DM1 Faults. Assume that multiple faults affect two diagonals in C1; diagonals $D_{j1} = x_{i,(i+j1) \ mod \ 4}$ and $D_{j2} = x_{i,(i+j2) \ mod \ 4}$ where $0 \leq i \leq 3$, and $j1 \neq j2$.

In C1, while performing SubBytes, $SB_{i,(i+j1) \ mod \ 4}$ and $SB_{i,(i+j2) \ mod \ 4}$ generate faulty output $y_{i,(i+j1) \ mod \ 4}$ and $y_{i,(i+j2) \ mod \ 4}$, respectively. After performing ShiftRows, MixColumns, and AddRoundKey, similar to Case 1, $[v_{r,j1}]_{r=0}^{3}$ and $[v_{r,j2}]_{r=0}^{3}$ are the faulty state elements. On the other hand, we apply the permuted inputs in C2. Using the same steps shown above, faulty state elements are represented as $[v'_{r,j1}]_{r=0}^{3}$ and $[v'_{r,j2}]_{r=0}^{3}$. Therefore, the faulty columns in C1 are the columns of $j1$ and $j2$, while the faulty columns in C2 correspond to column $(j1+3)$ *mod* 4 and $(j2+3)$ *mod* 4 in C1. Note that $j1 \neq (j1+3)$ *mod* 4 and $j2 \neq (j2+3)$ *mod* 4. Therefore, the location of the faulty columns in normal and CED-related rounds are different.

As an example, let $s_{0,0}$, $s_{1,1}$, $s_{2,2}$, and $s_{3,3}$ be the state elements of first faulty diagonal. Similarly, assume that $s_{0,1}$, $s_{1,2}$, $s_{2,3}$, and $s_{3,0}$ are the state elements residing in another faulty diagonal. In C1, after performing AddRoundKey, the faulty columns are shown as $[v_{r,0}]_{r=0}^{3}$ and $[v_{r,1}]_{r=0}^{3}$. However, in C2, the faulty columns are represented as $[v'_{r,0}]_{r=0}^{3}$ and $[v'_{r,1}]_{r=0}^{3}$, because $[v'_{r,0}]_{r=0}^{3} = [v_{r,3}]_{r=0}^{3}$ and $[v'_{r,1}]_{r=0}^{3} = [v_{r,0}]_{r=0}^{3}$. Similar to the case of DM0, the faulty columns in C1 and C2 are different.

Case 3: DM2 faults. The same proof used for Case 2 can be used for this case.

As discussed above, by using invariance-based CED to detect DM0, DM1, and DM2 fault models, at least one column of the original and permutated data states are different. Therefore, using the discussed invariance-based CED, all diagonal faults can be detected. Accordingly, invariance-based CED has 100% fault coverage for diagonal fault models of DM0, DM1, and DM2.

6 Conclusion

DFA is proven to be practical and low-cost. CEDs are used to detect the deliberately injected faults, However, conventional CEDs do not differentiate random and malicious faults. In conventional CEDs, their claims of security are based on the fault coverage of random fault model. But from an attacker's perspective, only a subset of the fault space is enough for successful attacks. Consequently, conventional CEDs may in fact miss the carefully constructed and deliberately injected faults by DFA. Further, they entail a large overhead. Moreover, since an exponentially large number of random faults are possible, fault simulation of randomly injected faults is used to estimate the fault coverage of the developed countermeasures. While this is reasonable for random faults (from single bit to a very limited extent multiple bit faults), it is unacceptable for carefully crafted and deliberately injected faults in the context of DFA. Simulations may miss such faults even though they may be harmful with respect to DFA. For example, parity-based techniques have 50% fault coverage of random number of faults for each of the unit a parity bit protects.

We show that such CEDs still allow DFA to succeed. Consequently, focusing on countermeasures useful in detecting DFA-exploitable faults and leveraging those properties that a DFA attack exploits is an important focus of this research. Our hypothesis is that such an investigation helps

us develop CED techniques that have a provably 100% fault coverage for DFA-exploitable faults, and also have high fault coverage for natural faults (assessed using simulation of randomly injected faults), but have very low overhead.

Therefore, we analyze the fault coverage of DFA-exploitable faults for conventional CEDs. We discover that most of the claims of security in previous work are not always true, because their fault coverage are based on the wrong attack model. Some CEDs such as parity provide only 50% fault coverage against DFA. Other CEDs which provide 100% fault coverage against DFA can incur more than 100% hardware overhead, e.g., hardware redundancy, hybrid redundancy, and robust code. The proposed invariance-based CED provides provably 100% fault coverage while only a minimum amount of overhead.

Note that a majority of the DFA attacks assume faults are injected in the datapath of AES. Our future research will explore the protection of key schedule unit. DFA of the key schedule of AES are proposed in [35, 39, 30]. State machine validation and duplication techniques can be use to protect the key schedule unit with small overhead. [23]. Nonlinear robust code can also be applied to protect the state machine [3].

Although the main discussion focuses on AES-128, it is note that the same conclusion also apply to AES-192 and AES-256 due to the similar datapath. We hope that this work will bring a new era to the CED for cryptographic devices.

References

[1] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. How to Flip a Bit? pages 235–239. IOLTS, Jul 2010.

[2] Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria. When Clocks Fail: On Critical Paths and Clock Faults. pages 182–193. CARDIS, 2010.

[3] Kahraman D. Akdemir, Zhen Wang, Mark Karpovsky, and Berk Sunar. Design of Cryptographic Devices Resilient to Fault Injection Attacks Using Nonlinear Robust Codes. *Fault Analysis in Cryptography*, pages 171–199, 2012.

[4] Frederic Amiel, Christophe Clavier, and Michael Tunstall. Fault Analysis of DPA-Resistant Algorithms. In *FDTC*, pages 223–236, 2006.

[5] Hagai Bar-el, Hamid Choukri, David Naccahe, Michael Tunstall, and Claire Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. In *IACR e-print archive 2004/100*. http://eprint.iacr.org, 2004.

[6] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache. Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures. *Proceedings of the IEEE*, PP(99):1, 2012.

[7] Alessandro Barenghi, Cédric Hocquet, David Bol, François-Xaiver Standaert, Francesco Regazzoni, and Israel Koren. Exploring the Feasibility of Low Cost Fault Injection Attacks on Sub-Threshold Devices through An Example of A 65nm AES Implementation. pages 48–60. in Proc. Workshop RFID Security Privacy, 2011.

[8] 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.

[9] Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. *Proceedings of Eurocrypt, Lecture Notes in Computer Science*, 1233:37–51, 1997.

[10] Johannes Blömer and Jean-Pierre Seifert. Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). In *Financial Cryptography*, pages 162–181, 2003.

[11] Y. Chih-Hsu and W. Bing-Fei. Simple Error Detection Methods for Hardware Implementation of Advanced Encryption Standard. *IEEE Trans. on Computers*, 55(6):730–731, 2006.

[12] Chistophe Giraud and Hugues Thiebeauld. A Survey on Fault Attacks. In *CARDIS*, page 159176, August 2004.

[13] Christophe Giraud. DFA on AES. In *IACR e-print archive 2003/008*, page 008. http://eprint.iacr.org/2003/008, 2003.

[14] Xiaofei Guo and R. Karri. Invariance-based Concurrent Error Detection for Advanced Encryption Standard. In *DAC*, pages 573–578, Jun 2012.

[15] J.Markoff. Vulnerability Is Discovered In Security for Smart Cards. New York Times, 2002.

[16] M. Joye, P. Manet, and JB. Rigaud. Strengthening Hardware AES Implementations against Fault Attack. *IET Information Security*, 1:106–110, 2007.

[17] A. Kaminsky, M. Kurdziel, and S. Radziszowski. An Overview of Cryptanalysis Research for the Advanced Encryption Standard. In *MILCOM*, pages 1310–1316, Nov 2010.

[18] Mark Karpovsky, Konrad J. Kulikowski, Er Taubin, and Senior Member. Robust Protection Against Fault-Injection Attacks of Smart Cards Implementing the Advanced Encryption Standard. In *DNS*, pages 93–101, 2004.

[19] Ramesh Karri, Kaijie Wu, P. Mishra, and Y. Kim. Concurrent Error Detection Schemes of Fault Based Side-Channel Cryptanalysis of Symmetric Block Ciphers. *IEEE Trans. on Computer-Aided Design*, 21(12):1509–1517, Dec 2002.

[20] Mehran Mozaffari Kermani and Arash Reyhani-Masoleh. A Low-Power High-Performance Concurrent Fault Detection Approach for the Composite Field S-Box and Inverse S-Box. *IEEE Trans. Computers*, 60(9):1327–1340, 2011.

[21] Farouk Khelil, Mohamed Hamdi, Sylvain Guilley, Jean Luc Danger, and Nidhal Selmane. Fault Analysis Attack on an AES FPGA Implementation. pages 1–5. ESRGroups, 2008.

[22] Israel Koren and C. Mani Krishna. *Fault-Tolerant Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[23] P. Maistri and R. Leveugle. Double-Data-Rate Computation as a Countermeasure against Fault Analysis. *IEEE Transactions on Computers*, 57(11):1528–1539, Nov 2008.

[24] Paolo Maistri. Countermeasures against Fault Attacks: the Good, the Bad, and the Ugly. In *IOLTS*, pages 134–137, Jul 2011.

[25] T.G. Malkin, F.-X. Standaert, and Moti Yung. A Comparative Cost/Security Analysis of Fault Attack Countermeasures. In *FDTC*, pages 109–123, Sept 2005.

[26] Amir Moradi, Mohammad T. Manzuri Shalmani, and Mahmoud Salmasizadeh. A Generalized Method of Differential Fault Attack against AES Cryptosystem. In *CHES*, pages 91–100, 2006.

[27] Mehran Mozaffari-Kermani and Arash Reyhani-Masoleh. Parity-Based Fault Detection Architecture of S-box for Advanced Encryption Standard. In *DFT*, pages 572–580, Oct 2006.

[28] Mehran Mozaffari-Kermani and Arash Reyhani-Masoleh. Concurrent Structure-Independent Fault Detection Schemes for the Advanced Encryption Standard. *IEEE Trans. Computers*, 59(5):608–622, 2010.

[29] 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.

[30] Debdeep Mukhopadhyay. New Differential Fault Analysis on AES Key Schedule: Two Faults Are Enough. In *CARDIS*, pages 48–60, 2008.

[31] Debdeep Mukhopadhyay. An Improved Fault Based Attack of the Advanced Encryption Standard. In *AFRICACRYPT*, pages 421–434, 2009.

[32] National Institute of Stardards and Technology (NIST). Advanced Encryption Standard (AES). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, Nov 2001.

[33] National Institute of Stardards and Technology (NIST) Federal Information Processing Standards (FIPS) publication 140-2. Security requirements for cryptographic modules. http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf, Mar 2001.

[34] G. Letourneux P. Dusart and O. Vivolo. Differential Fault Analysis on AES. In *Cryptology ePrint Archive*, pages 293–306, Oct 2003.

[35] 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.

[36] Dhiman Saha, Debdeep Mukhopadhyay, and Dipanwita Roy Chowdhury. A Diagonal Fault Attack on the Advanced Encryption Standard. *IACR Cryptology ePrint Archive*, page 581, 2009.

[37] Nidhal Selmane, Sylvain Guilley, and Jean-Luc Danger. Practical Setup Time Violation Attacks on AES. pages 91–96. European Dependable Computing Conference, 2008.

[38] Sergei P. Skorobogatov and Ross J. Anderson. Optical Fault Induction Attacks. In *proceedings of CHES*, pages 2–12, Aug 2002.

[39] Junko Takahashi, Toshinori Fukunaga, and Kimihiro Yamakoshi. DFA Mechanism on the AES Key Schedule. In *FDTC*, pages 62–74, 2007.

[40] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault. In *WISTP*, pages 224–233, 2011.

[41] 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.

[42] Xilinx. ChipScope Pro. http://www.xilinx.com/support/ documentation/dt_chipscopepro.htm.