Fault Attacks on AES and Their Countermeasures

Subidh Ali, Xiaofei Guo, Ramesh Karri, and Debdeep Mukhopadhyay

Abstract Fault Attacks exploit malicious or accidental faults injected during the computation of a cryptographic algorithm. Combining the seminal idea by Boneh, DeMillo and Lipton with Differential Cryptanalysis, a new field of Differential Fault Attacks (DFA) has emerged. DFA has shown that several ciphers can be compromised if the faults can be suitably controlled. DFA is not restricted to old ciphers, but can be a powerful attack vector even for modern ciphers, like the Advanced Encryption Standard (AES). In this book chapter, we present an overview on the history of fault attacks and their general principle. The chapter subsequently concentrates on the AES algorithm and explains the developed fault attacks. The chapter covers the entire range of attacks finally showing that a single random byte fault can reduce the AES key to 2^8 values, with a time complexity of 2^{30}. Further extensions of the fault attack to multiple byte fault models and attacks targeting the AES key schedule are also presented in the chapter. These attacks emphasize the requirement of counter-measures to detect the underlying faults and accordingly suppress the invalid output. The chapter then presents a survey of existing DFA countermeasures, concluding with the efficient Concurrent Error Detection (CED) schemes which have been developed utilizing the invariance properties in AES. Such a strategy provides near 100 % fault coverage at a less overhead. The combined chapter shows that DFA against AES are practical, and can be prevented using suitable techniques.

The authors "Subidh Ali" and "Xiaofei Guo" have equally contributed to this book chapter.

S. Ali
New York University Abu Dhabi, Abu Dhabi 129188, UAE
e-mail: subidh.ali@nyu.edu

X. Guo • R. Karri
Five MetroTech Center Brooklyn, New York University School of Engineering,
Brooklyn, NY 11201, USA
e-mail: xg243@nyu.edu;rkarri@nyu.edu

D. Mukhopadhyay (✉)
Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur,
Kharagpur, West Bengal 721302, India
e-mail: debdeep@cse.iitkgp.ernet.in

C.-H. Chang, M. Potkonjak (eds.), *Secure System Design and Trustable Computing*, DOI 10.1007/978-3-319-14971-4_5

1 Introduction: Faults and Cryptosystems

The growing complexity of the cryptographic algorithms and the increasing applications of ciphers in real time applications has lead to research in the development of high speed hardware designs or optimized cryptographic libraries for these algorithms. The complex operations performed in these designs and the large state space involved indicates that a complete verification is ruled out. Hence these designs have a chance of being fault prone. Apart from these unintentional faults, faults can also be injected intentionally. Literature shows several ways of fault injection: accidental variation in operating conditions, like voltage, clock frequency, or focussed laser beams in hardware. Software programs can also be subjected to situations, like missing of certain instructions to inflict faults. Apart from the general issue of fault tolerance in any large design, faults bring a complete new aspect when dealing with cryptographic algorithms: security.

The first thing that comes to mind is the relation between faults and secrets. In this section, we first attempt to motivate the impact of faults in the leakage of information.

Motivating Example Consider a pedagogical example comprising of two hardware devices as illustrated in Fig. 1.

The first device has a register storing the values $R1 = (6, 0)$, and computes the product $y_{left} = (6, 0) \times (a, b)^T = 6a \bmod m$. The value of m is fixed as say 8. The second device on the other hand has the register with value $R2 = (0, 2)$ and computes $y_{right} = (0, 2) \times (a, b)^T = 2b \bmod m$. The users can feed in values of (a, b), st. $(a, b) \in \{2, 6\} \times \{2, 6\}$. For the rest of the discussion all the computations are mod 8 and are not explicitly stated.

The user can only input the values (a, b) chosen from the 4 values of $\{2, 6\} \times \{2, 6\}$. On receiving the inputs (a, b) both the hardwares compute the values of y_{left} and y_{right}. However the user is given either y_{left} or y_{right}, chosen on the basis of a random toss of an unbiased coin which is hidden from the user. The challenge of the user is to guess the outcome of the random coin with a probability better than $\frac{1}{2}$.

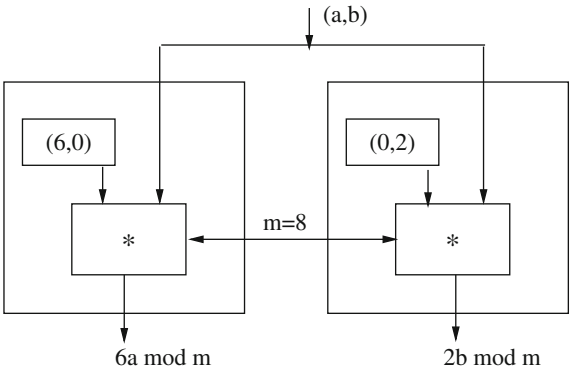


Fig. 1 Effect of faults on secrets

The user is allowed to make multiple queries by providing any of the 4 inputs (a, b). It can be assumed that the random choice is kept constant for all the inputs.

It may be easily observed that $y_{left} = y_{right}$ for all the 4 values of (a, b) which implies that the output y_{left} or y_{right} does not reveal which output is chosen by the random toss. For all the input values of (a, b) the output is 4.

Now consider that one of the hardwares is subjected to a permanent stress, which creates a fault in either the registers $R1$ or $R2$. Hence, either $R1 = r \neq 6$ or $R2 = r \neq 2$. If the fault occurs in $R1$, $y'_{left} = ra$, while $y_{right} = 2b$. Else if the fault occurs in $R2$, $y_{left} = 6a$, while $y'_{right} = rb$. WLOG. assume that the fault is in the first device.

Now the attacker provides two inputs: $(2, 2)$ and $(6, 6)$ to the hardware devices. The attacker observes both of the outputs. If both the outputs are the same then the attacker concludes that the right output is chosen, while if they are different the left output is chosen with probability 1.

Thus this simple example shows that a fault can leak information which seemed to be perfectly hidden in the original design. Thus apart from the malfunction of hardware or software designs, algorithms which hide information (like ciphers) should be analyzed w.r.t. faults. Next, we consider a more non-trivial example of fault based analysis of the popular RSA cryptosystem.

1.1 Fault Analysis of the RSA Cipher

The first fault based attack was mounted on the well-known public key cryptosystem *RSA*. We know that *RSA* works by considering two keys: a *public key* is known to every one, while a *private key* is *secret*. Encryption of a message is performed using the public key, but decryption requires the knowledge of the private key. All the operations are done mod n, where n is the product of two large distinct prime number p and q. The values of p and q are however private and hence not disclosed to all. The encryption key, which is public is a value b, where $1 \leq b \leq \phi(n)$ where $\phi(n)$ is the Euler-Totient function. The decryption key is a private value a, which is selected such that $ab \equiv 1 \, mod \, \phi(n)$. The owner of the private key (p, q, a) publishes the value (b, n) which is the public key.

The encryptor chooses a message x, where $x \in Z_n$. It may be mentioned that $Z_n = \{0, 1, \ldots, n-1\}$. The encryption process is computing the cipher as $y \equiv x^b \, mod \, n$ using the public key b. Since the decryptor knows the value of a, which is the private key, he computes the value of x from y by computing $y^a \equiv (x^b)^a \, mod \, n \equiv x \, mod \, n$. The security of *RSA* is based on the assumption that decryption can be performed only by the knowledge of the private key b. However to obtain the private information from the public value a requires one to compute the modular inverse of a modulo $\phi(n)$. It is believed that to obtain $\phi(n)$ from n requires the knowledge of the prime factors of n, namely p and q. The security of RSA is thus based on the hardness assumption of factorization of large n.

However we explain that under situation of faulty computations the value of the secret exponent a can be retrieved by efficient algorithms. In the attack it is assumed that the attacker is in possession of certain number of plaintext and ciphertext pairs. The attacker has the ability to flip one bit of the value a during computation. Say, the ith bit a_i of a is flipped and modified to $\hat{a}_i$, where $0 \leq i \leq |a|$ and $|a|$ is the bit-length of a. The attacker has access to both fault-free X and faulty plaintexts $\hat{X}$. Therefore, he can compute, $\frac{X}{\hat{X}} = \frac{Y^{\hat{a}}}{Y^a} = \frac{Y^{2^i \hat{a}_i}}{Y^{2^i a_i}} \, mod \, n$. If the ratio is equal to Y^{2^i}, the attacker can be sure that $a_i = 0$. On the other hand if the ratio is $\frac{1}{Y^{2^i}}$, the attacker ascertains that $a_i = 1$. The same technique is repeated for all the values of i, thus a can be retrieved. The attack is also applicable when the fault is induced in Y. It is also possible in cases when the fault flips two or more bits. The details are left to the reader as an exercise.

These attacks show that fault analysis can be a powerful tool for attacking ciphers. Significant research has been performed in the field of fault based cryptanalysis of various ciphers of different types. From the seminal paper of [10] fault attacks have been improved with the ideas of differential analysis to attack block ciphers, like Data Encryption Standard (DES). However, after the acceptance of the 128-bit version of the *Rijndael* block cipher, designed by Vincent Rijmen and Joan Daemen, as the Advanced Encryption Standard (AES) in 2001, the main focus of fault attacks have been AES. In the following section we present an overview on the AES algorithm.

2 Preliminaries

The chapter focuses on AES and its fault analysis. The present section provides a top level description of the block cipher algorithm.

2.1 AES Algorithm

AES is an iterative block cipher, designed by Vincent Rijmen and Joan Daemen. The algorithm, originally designed to support both block and key lengths of 128, 192, and 256 bits, the standardizes AES supports only block length of 128 bits, though the key can be of all the three specifications: 128, 192 and 256 bits.

The AES algorithm is an iterated block cipher, meaning the plaintext is applied over several rounds to obtain the final ciphertext. The three versions of AES, ie. AES-128, AES-192, and AES-256 has 10, 12, and 14 rounds. The 128-bit input plaintext is transformed by the rounds into a 128-bit output ciphertext. All the rounds of AES are identical except the last round with a slight change.

Each round of AES encryption consists of SubBytes, ShiftRows, MixColumns, and AddRoundKey denoted by B, S, M, and A, respectively, as shown in Fig. 2.

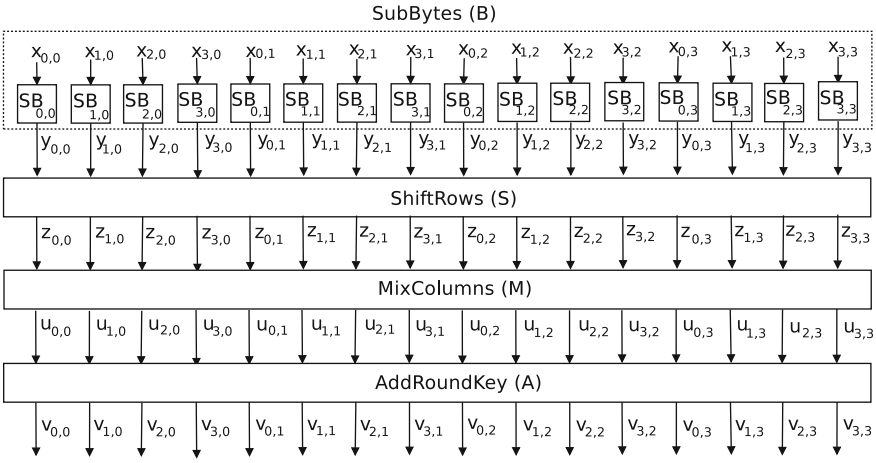


Fig. 2 One AES encryption round

The plaintext is first *mixed* with the input key through a key XORing operation. Subsequently the rounds are applied, which are composed of the above mentioned four operations, B, S, M, and A. The last round is only distinct as MixColumns is not performed.

2.2 Round Transformations of AES

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.

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

In SubBytes, all bytes are processed separately by 16 S-boxes (SBs in Fig. 2). 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 1, 2, and 3 bytes respectively. The resulting output is:

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

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

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

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:

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

2.3 Key Scheduling Algorithm

The round keys are generated by the AES key scheduling algorithm, as shown in Algorithm 1. The master key K is used to derive all the round keys, where N_k, N_r and K^r represent the key length in words (4 bytes), number of rounds and the rth round key respectively. The input key is of size $4N_k$ bytes. The algorithm produces the rth round key, which is denoted by K^r. As $0 \leq r \leq N_r$, the total expanded round keys can be stored in the vector $W[N_b(N_r+1)]$, where N_b is the block length of AES. The algorithm consists of operations: *SubWord* and *RotWord*, which are explained as follows: The operation SubWord consists of SubByte operations applied to each of the 4 bytes separately on every byte of a word. The RotWord operation is a cyclic circular left shift on the bytes of an input word. Finally, the round constant abbreviated as $Rcon[n] = (\{02\}^n, \{00\}, \{00\}, \{00\})$. For more details one can refer to the AES specification [43].

3 Introduction to Differential Fault Analysis

The first fault attack was applied to the *RSA* cryptosystem. Biham and Shamir proposed a new fault based attacking technique which is wildly known as Differential Fault Analysis (DFA)[10]. DFA attack is a very powerful attack model which can threaten a large class of ciphers. However, the actual attack procedure may vary from cipher to cipher, and one has to exploit the fault propagations suitably to

Algorithm 1: AES Key Scheduling Algorithm

Input: K the initial key of length N_k bytes
Output: K^r the round key where $0 \leq r \leq N_r$

for $i = 0$ *to* $N_k - 1$ **do**
  $W[i] \leftarrow \{K[4 * i], K[4 * i + 1], K[4 * i + 2], K[4 * i + 3]\}$;

for $i = N_k$ *to* $N_b * (N_r - 1)$ **do**
  $temp \leftarrow W[i - 1]$;
  if $i \ mod \ N_k = 0$ **then**
    $temp \leftarrow SubWord(RotWord(temp)) \oplus Rcon[i/N_k]$;
  else if $N_k > 6$ *and* $i \ mod \ N_k = 4$ **then**
    $temp \leftarrow SubWord(temp)$;
  $W[i] \leftarrow W[i - N_k] \oplus temp$;

return W

extract the key of a given cipher. The foremost DFA proposed was on the DES cipher, which is essentially a Feistel cipher. Later, DFA has been extensively applied on other ciphers, with greater focus on the AES algorithm. Before discussing on fault based analysis of the AES cipher, we would discuss a general idea on DFA of block ciphers. We would restrict ourselves to the Substitution Permutation Network (SPN), as AES belongs to this family. However, similar observations and results can be obtained for Feistel structures, putting to threat all block ciphers of the modern day.

3.1 General Principle of DFA of Block Ciphers

In this section, we study the basic principle of DFA which shall be subsequently applied for the AES algorithm. As apparent from the name, DFA combines the concepts of differential cryptanalysis with that of fault attacks. DFA is applicable to almost any secret key cryptosystem proposed so far in the open literature such as DES, IDEA, and RC5 [10].

There has been considerable number of work about DFA of AES. Some of the DFA proposals are based on theoretical model [11, 15, 16, 36, 40, 45, 46, 54], while others launched successful attacks on ASIC and FPGA devices using previously proposed theoretical models [2, 8, 28, 46, 49]. The key idea of DFA is composed of three steps as shown in Fig. 3. (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 space.

Practicality of DFA depends on the underlying fault model and the number of faulty ciphertext pairs needed. In the following section we will analyze all the fault models DFA of AES uses and point out their relationships. In this section, we continue the discussion on the working principle of DFA w.r.t. a generalized block cipher model.

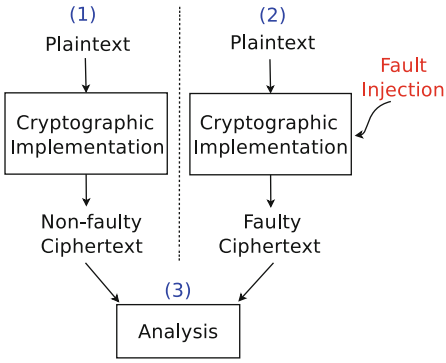


Fig. 3 Three steps of DFA

DFA works under the assumption of underlying faults. These faults are often caused by various mechanisms, like: fluctuation of operating voltage, varying the clock frequency, changing the temperature of a device and with the most accurate injection of laser beams. However, in all of the above techniques the faults are created by sudden variation of the operating conditions. It may be noted that apart from the above means of malicious or intentional fault injections, faults can be also unintentional. With the growing complexity of crypto devices, chances of mistakes in the design also increase.

Faults can be categorized depending on whether they are permanent or transient. From the point of view of cryptography, we would like to point out that transient faults are of high concern as they are hard to detect. These faults can be of such a short duration that most simulation based techniques of fault detection may be unable to detect the advent of the faults. However, as we shall soon observe that few faults are enough to leak the entire key of a standard cipher, like AES.

3.1.1 Fault Models

The faults can be of varying nature but can be categorized as follows:

1. **Bit Model**: This fault model assumes that the faults is localized to one bit. The fault control is crucial here, as there is a high probability that a random fluctuation of the operating conditions can lead to more than one bit getting affected. Hence attacks based on such models are often unrealistic and may not be practically viable.
2. **Single Byte**: A more practical and most common fault model is the single byte model. This fault model assumes that the faults are spread to bytes and the fault model can be any *random* non-zero value. This non-specificity of the fault value makes these types of DFAs very powerful and practical techniques.
3. **Multiple Byte**: In this fault model, it is assumed that the faults propagate to more than 1 byte. More often, these models are more practical, in the sense that the DFAs based on them work even with lesser fault control. In context to DFA of

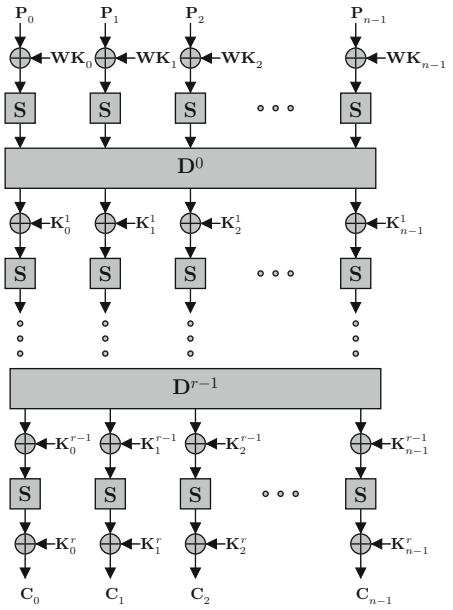


Fig. 4 Basic structure of SPN ciphers

AES, we shall observe a special fault model, namely the *Diagonal Fault Model* which helps to generalize the DFA of AES to a large extent. The fault values are again arbitrary, and hence makes these attacks very powerful.

3.1.2 The Effect of Faults on a Block Cipher

It is expected that the induced fault changes certain bits or bytes during a particular round of the encryption and generates certain differences. Most often the DFAs target the non-linear transformations, namely S-Boxes of the block ciphers. As the faults are induced during the encryption process, the fault propagation patterns give some *relations* between the input and output difference of certain S-boxes. In most of the ciphers like AES, the S-boxes are known and therefore, one can easily deduce the *difference distribution table* of the S-box being used. Generally, the S-boxes have inputs which are combined with part of the keys through some mixing operation. Using the difference distribution table and the relations between the input and output difference one reduces the search space of a part of the key. This divide and conquer mechanism helps to recover the entire key quite efficiently for most ciphers. We explain the working in more details for the generalized SPN cipher, as modeled in Fig. 4.

Figure 4 shows the basic structure of r-round Substitution Permutation Network (SPN) cipher with block length n-bytes. Each round consists of confusion layer S which is realized by non-linear S-box operation, and a linear transformation called diffusion layer D, followed by an addition with the round key. There is an addition

with the whitening key *WK* at the beginning of the encryption called key-whitening phase. The diffusion layer is generally provided by multiplication with *MDS* matrix followed by some rotation operations. The diffusion operation plays a major role in DFA. If a byte is modified at the input of the diffusion operation, the induced difference spreads to multiple bytes at the output depending on the *branch number* of the diffusion layer. The disturbed bytes are often referred to as the active bytes in the literature of differential cryptanalysis. Branch number for a diffusion matrix on bytes is used to observe how a non-zero input differential spreads in a cipher through the diffusion layer. Branch number is defined as the sum of the minimum number of active bytes at the input and output of the diffusion layer.

As the diffusion layer is a linear operation w.r.t. the key mixing operation namely XOR, the output difference can be expressed as a linear relation of the input differences. The attacker exploits these properties in the following fashion to obtain the key. Say a single byte fault is induced at the input of $(r-1)$th (penultimate) round and the corresponding difference at the input of D^{r-1} is $\alpha \neq 0$. If the branch number of the diffusion layer is b, the input byte fault will spread to $b-1$ bytes $(\alpha_{\pi_0}, \ldots, \alpha_{\pi_{b-2}})$ at the output of D_{r-1}, where π denotes the transformation of the diffusion layer. Each of these active bytes then pass through the S-boxes, which non-linearly transform them. The attacker then represents these output bytes in terms of a pair of fault-free and faulty ciphertexts (C, C^*) as follows:

$$\alpha_{\pi_j} = S^{-1}(C_{\pi_j} \oplus K^r_{\pi_j}) \oplus S^{-1}(C^*_{\pi_j} \oplus K^r_{\pi_j}) \tag{6}$$

where $j \in \{0, \ldots, b-2\}$ and S^{-1} represent the inverse of the S-box operation. Now the attacker knows the S-box input difference $C_{\pi_j} \oplus C^*_{\pi_j}$. From the difference distribution table he knows on an average *few* values satisfy a chosen $(\alpha_{\pi_j}, C_{\pi_j} \oplus C^*_{\pi_j})$ pair.

Further, because of the linear mapping in D^{r-1}, α_{π_j} depends linearly on α. Therefore, the attacker guesses the value of α and get the values of α_{π_j} i.e. the output differences. Using the input–output difference he retrieves the value $C_{\pi_j} \oplus K_{\pi_j}$ from the difference distribution table of the S-box. As C_{π_j} and $C^*_{\pi_j}$ are known to the attacker, hence he can retrieve the value of K_{π_j}. The attacker may need to induce faults multiple times in order to get all the bytes of the round key.

In the next section, we present the fault models used for DFA of AES in the literature and a summary of all the attacks performed. Subsequently, we present the fault attacks on AES.

4 DFA and Associated Fault Models

DFA exploits a small subspace of all possible faults. The practicality of a fault attack largely relies on the underlying fault model: the nature of the faults and the ability to actually obtain such faults in practice. Any random fault is not attackable. Only

certain fault models are feasible enough to reveal the secret key in practical time. In the following section, we classify the DFA fault models in four scenarios by the location and round in which faults are injected.

4.1 Fault Models for DFA of AES

Table 1 is a summary of the published DFA of AES. Faults can be injected either (I) in AddRoundKey in round 0, (II) between the output of seventh and the input of eighth round MixColumns, or (III) between the output of eighth and the input of ninth round MixColumns. In each scenario, we analyze the (A) fault models, (B) number of faulty ciphertexts needed, (C) the key 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. It may be noted that we have purposefully omitted faults of permanent nature: namely stuck-at-1 or stuck-at-0 as they are not relevant from the DFA perspective. Rather transient faults are more relevant, because of their stealthy nature and ability to defeat counter-measures for classical fault tolerance. For detailed discussion in this direction, we would redirect the author to [56].

In the following discussions in this section we elaborate the fault models present in Table 1.

Table 1 A summary of DFA of AES

Fault model			No. of faulty CTs ⋆	Key space	Experiment
Section 4.1.1 Faults are injected in AddRoundKey in round 0					
Single bit		[11]	128	1	No
Section 4.1.2 Faults are injected between the output of seventh and the input of eighth round MixColumns					
Single byte		[45]	2	2^{40}	Underpowering [28, 49]
		[40]	2	2^{32}	No
		[54]	1	2^{8}	No
Multiple byte	DM0	[46]	1	2^{32}	Overclocking [46]
	DM1	[46]	1	2^{64}	
	DM2	[46]	1	2^{96}	
	DM3	[46]	2^{128}	2^{128}	
Section 4.1.3 Faults are injected between the output of eighth and the input of ninth round MixColumns					
Single bit		[16]	≈ 50	1	Overclocking [2]
Single byte		[15]	≈ 40	1	Underpowering [8]
		[36]†	6	1	No
Multiple byte	DM0	[36]‡	6	1	No
	DM0	[36]◇	1500	1	No

⋆ *CT* ciphertext, † Only 1 byte in a word is faulty, ‡ 2 or 3 bytes in a word are faulty, ◇ All 4 bytes in a word are faulty

4.1.1 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 [11], 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 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].

4.1.2 Faults are Injected Between the Output of Seventh and the Input of Eighth 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 [45], two faulty ciphertexts are needed to obtain the key. This fault model is experimentally verified in [28, 49]. In [49], 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 [28], the authors underpower an AES FPGA implementation to inject faults with a probability of 40 % for single byte fault injection.

Multiple Byte Transient Fault Saha et al. [46] 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 4 bytes. **A diagonal** is a set of 4 bytes of the AES state matrix, where the ith diagonal is defined as follows:

$$D_i = \{s_{j,(j+i)mod4} \; ; \; 0 \leq j < 4\} \tag{7}$$

We obtain the following four diagonals.

$$D_0 = (s_{0,0}, s_{1,1}, s_{2,2}, s_{3,3}), \; D_1 = (s_{0,1}, s_{1,2}, s_{2,3}, s_{3,0}),$$
$$D_2 = (s_{0,2}, s_{1,3}, s_{2,0}, s_{3,1}), \; D_3 = (s_{0,3}, s_{1,0}, s_{2,1}, s_{3,2})$$

Fault in diagonal D_i will affect the entire ith column of the state matrix after the `MixColumns` operation. The diagonal fault model is classified into four different cases, denoted as DM0, DM1, DM2, and DM3. As shown in Fig. 5, 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 (Fig. 5).

The authors also validate the diagonal fault model with a practical fault attack on AES FPGA implementation using overclocking.

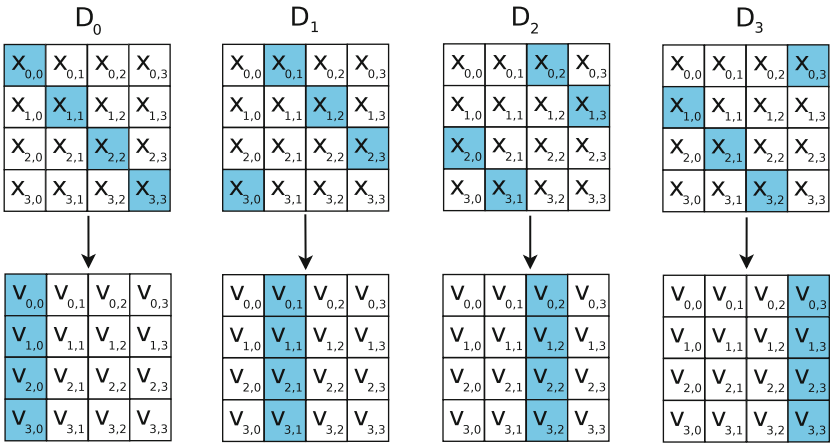


Fig. 5 Fault propagation of diagonal faults. The *upper row* shows the diagonals that faults are injected in. The *lower row* shows the corresponding columns being affected

4.1.3 Faults are Injected Between the Output of Eighth and the Input of Ninth MixColumns

Single Bit Transient Fault In [16], the attacker needs only three faulty ciphertexts to succeed with a probability of 97 %. The key space is trivial. Agoyan et al. [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 [15], 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 65 nm ASIC chip [8]. In this attack, 3,9881 faulty ciphertexts are collected during the ten experiments; 3,0386 of them were actually the outcome of a single byte fault. Thus, it has a successful injection rate of 76 %.

Multiple Byte Transient Fault Moradi et al. [36] 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 1, 2, or 3 bytes. So this fault model includes the single byte fault model. If only one single byte fault is injected, 6 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. In the following subsection, we present a detailed analysis on the inter-relationships of the fault models discussed so far.

4.2 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 among various fault models is the basis for understanding and comparing the various fault attacks on AES. Further the inter-relationships developed also help in analyzing the security of the counter-measured: both conventional as well as for designing new DFA-specific CED. Because DFA of AES targets the last few rounds,[1] we synthesize the relationships between different fault models based on the locations and rounds they are injected in.

4.2.1 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.

4.2.2 Faults are Injected Between the Output of Seventh and the Input of Eighth MixColumns

Figure 6a summarizes the relationships between the DFA-exploitable fault models by injecting faults in the output of seventh round MixColumns and the input of eighth 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 (8).

$$\text{Single Byte} \subset DM0 \subset DM1 \subset DM2 \subset DM3 \tag{8}$$

A more careful look reveals that 2 byte faults can be either DM0 or DM1 but not DM2. Similarly, 3 byte fault can not be DM3. The relationship between faulty bytes from 5 to 12 and diagonal fault models are summarized in Fig. 6a.

As shown in Fig. 6a, DM3 includes all possible byte transient faults. The attacks proposed in [46] 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 Sect. 4.1.2. 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 more encompassing the fault model is, the more realistic the attacks based on it are.

[1] In general, the practical faults used in DFA target the seventh, eighth, and ninth rounds.

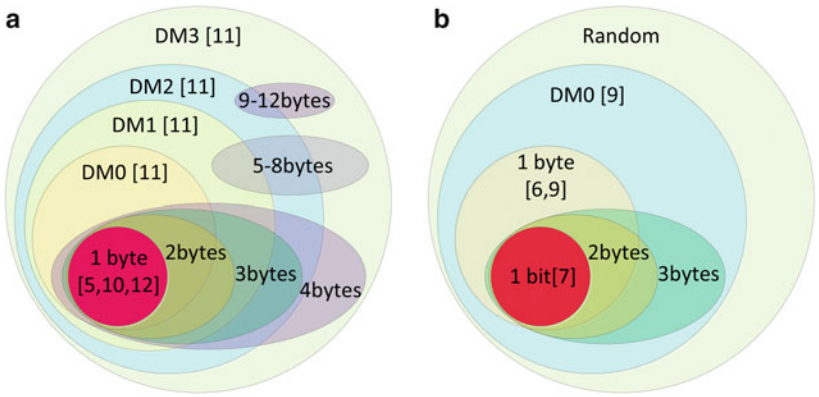


Fig. 6 Relationships between DFA fault models when faults are injected between (**a**) the output of seventh and the input of eighth round MixColumns, (**b**) output of eighth and the input of ninth round MixColumns

4.2.3 Faults are Injected Between the Output of Eighth and the Input of Ninth MixColumns

Figure 6b summarizes the relationships between the DFA-exploitable fault models by injecting faults in the output of eighth and the input of ninth 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.

In the following section, we detail the above mentioned fault attacks on AES.

5 Differential Fault Attacks on AES: Early Efforts

A very central property which is used in the algorithms to perform a DFA of AES is the differential features of the AES S-boxes. The following section presents the differential property of the AES S-Box.

5.1 Differential Properties of AES S-Box

In this section we discuss differential properties of S-box, which will be useful for DFA. In case of AES, the input to the S-box in each round is the XOR of previous round output and the round key. Figure 7a shows two S-box operations: one with normal input *in* and the other with a difference α to the input.

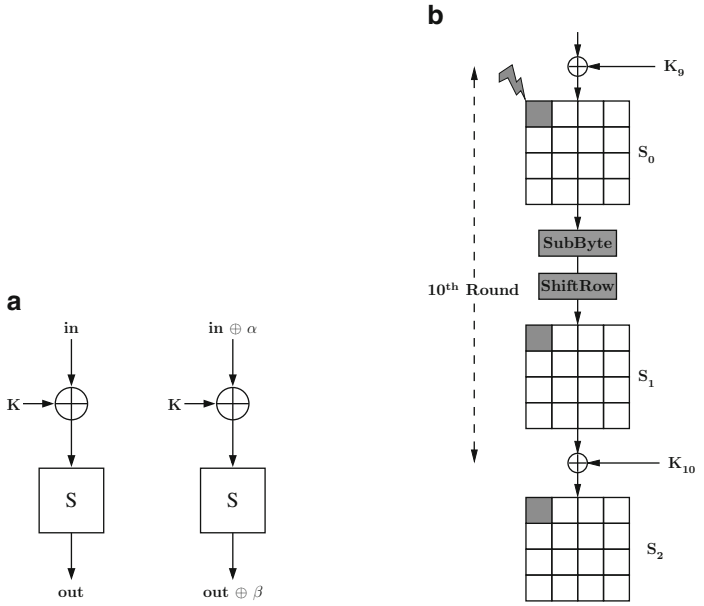


Fig. 7 Differential property of AES. (**a**) Difference across S-box. (**b**) Flow of fault in the last round

Here *in* is the previous round output byte and K is the round key byte. The AES S-box is a non-linear operation, therefore, input difference α will change to β at the S-box output *out*. Now if we replace the value of $in \oplus K$ by X, we can relate the input output differences by following equation:

$$\beta = S(X \oplus \alpha) \oplus S(X) \tag{9}$$

According to the properties of AES S-box for a particular value of α and β the above equation can have 0, 2, or 4 solutions of X [44]. For a fixed value of α, among the 256 possible values of β, only one value leads to four solutions of the equation and 126 values lead two solutions. The rest of the values will not produce any solution. This implies only 127 out of 256 choices of β produce solutions for X and the average number of solutions of X is one. It may also be noted that if we know the values of α, β, and *in*, we can get the values of K from the above equation. This property is being used in most of the advanced DFAs on AES. In the subsequent part of the chapter we explain DFA of AES using these properties.

5.2 DFA of AES Using Bit Faults

When AES was introduced at that time side-channel analysis and fault analysis were become two very prominent fields of research in the research community. The first DFA was already proposed on DES. Therefore, it was a challenge for the researchers in this field to analyze AES in the light of DFA. The initial attempts were further inspired by the fault injection techniques, which practically demonstrated that flipping a single bit of an intermediate computation result is possible using relatively less expensive devices like simple camera flash installed on a microscope or laser equipments [51]. However byte faults are more realistic compared to the bit faults.

In the following sections we discuss about DFA using bit level and byte level fault models. The induced fault is assumed to be random in nature and the attack algorithm is oblivious of the fault value. Let us first study the DFA which targets the last round of the AES encryption.

5.3 Bit Level DFA of Last Round of AES

In this attack, originally proposed in [16], it is assumed that the fault is induced at any particular bit of the last round input. However, the exact fault location, i.e., the exact bit where the fault is created is unknown.

Let us consider AES with 128-bit key for the sake of simplicity, though the discussion can be easily extended to the other AES versions. Figure 7b shows the flow of fault corresponding to the single-bit difference at the input of the tenth round. In the figure, K^9 and K^{10} denotes the ninth and tenth round keys respectively. The state matrices S_0, S_1 and S_2 show the corresponding XOR differences of the fault-free and the faulty states. As the fault is induced in a bit, therefore, the disturbance is confined within a single byte. So, from the XOR difference of fault-free and faulty ciphertexts one can easily get the location of the faulty byte (note that the bit is not evident because of the S-Box).

Consider the fault induced at the (i, j)th byte of the tenth round input state matrix S_0. Let x be the fault-free value at the tenth round input and ε be the corresponding fault value. Note that the attacker is aware of the fault-free (C) and faulty (C^*) ciphertexts. As already stated, from the XOR difference of C and C^* one can get the byte position (i, j), where the fault is induced. The (i, j) byte where the bit fault is induced in the byte x can be represented in terms of (C, C^*) as follows:

$$C_{i,l} \oplus C^*_{i,l} = SR(S(x_{i,j})) \oplus SR(S(x_{i,j} \oplus \varepsilon)) \tag{10}$$

Note that $l = (j - i) \bmod 4$ provides the corresponding column index, where the faulty byte in the jth column and ith row shifts due to the *Shiftrows* operation. In other words, the fault location (i, j) in the difference matrix S_0 changes to (i, l) at the tenth round output.

Having obtained the location of the fault, the attacker is now set to ascertain the value of the fault. Note the similarity of the above equation with that of Eq. (9). The value of $C_{i,l} \oplus C^*_{i,l}$ being known to the attacker, in order to get the value of $x_{i,j}$ he guesses eight possible values of ε. For each possible value of ε, he gets on an average one hypotheses for $x_{i,j}$, which will satisfy the above equation (refer Sect. 5.1). Thus, for all the eight possible values of ε, he gets on an average eight candidates for $x_{i,j}$. In order to identify the unique value for $x_{i,j}$ he obtains another faulty ciphertext by injecting another fault (i.e., the fault location is a different bit) in the same byte. A similar approach leads to another set of eight values for $x_{i,j}$. Intersection of these two sets from the two different faulty ciphertexts is expected to determine the exact value of $x_{i,j}$.

This same technique is repeated for the other bytes in order to get all the 16 bytes of x. On an average thus $2 \cdot 16 = 32$ faulty ciphertexts are needed to determine the value of the state matrix x. Thus, the attacker obtains the fault-free input of tenth round. Being aware of the fault-free ciphertext C, one can easily retrieve the tenth round key K^{10} from the relation $C = SR(S(x) \oplus K^{10})$. Then, as the AES key-schedule is invertible one trivially retrieve the master key.

5.4 Bit Level DFA of First Round of AES

In this section we describe another bit level DFA, where the fault is induced in the first round of AES encryption. This is a more general attack and applicable to most of the ciphers. The main difference of this attack from the previous ones and the others which follow is the underlying fault model. The fault model is a *bit reset* model, which implies that the attacker has capability to reset a specific bit at a targetted byte location of the AES encryption. The attacker targets the first key whitening operation before the *SubBytes* operation. The plaintext is set to a zero string of length 128 bits, denoted as P_{zero}. The plaintext is fixed throughout the attack and the objective of the attack is to obtain the whitening key K.

To start with, an encryption is done using P_{zero} and K under normal environment and the fault-free ciphertext C_{zero} is obtained and stored. Now a fault is induced according to the fault model discussed. It is assumed that the induced fault resets the lth bit of the (i, j) byte at the input to the first *SubBytes* operation. Let us assume that the fault free input to the *SubBytes* operation is x. Therefore, we can write $x = P_{zero} \oplus K$. The attacker tries to detect the value of l by repeating the following simple steps: He compares the fault-free ciphertext, C_{zero} with that of the faulty one, C^*_{zero}. If they are equal it implies that the lth bit of the (i, j)th byte of x, which was reset due to the fault, was already zero and thus the effect of the reset fault was inconsequential. Thus the corresponding bit of the (i, j)th key byte was zero (as the plaintext is all zero). On the other hand, a different value of C_{zero} and C^*_{zero} implies that the induced fault reset the bit $x^l_{i,j}$ with effect. That means the fault-free value of $x^l_{i,j}$ was one and after fault induction it changes to zero. This also means the

corresponding bit value of K is one. The fault thus reveals whether a particular bit of the whitening key is one or zero. The same technique is repeated for all the 128 bits, and thus 128 faulty ciphertexts are needed to get the master key.

The attack is relatively simple in nature, however is relatively less practical. The assumed fault model is impractical as it requires very precise control over the fault location to enable fault in every bit positions. Thus fault attacks on AES with more relaxed fault models and lesser fault induction requirements are desirous and topics of the future sections.

6 State-of-the-Art DFAs on AES

In this section, we present an overview on some of the more recent fault attacks on AES. These attacks use more practical fault model, namely the byte faults.

6.1 Byte Level DFA of Penultimate Round of AES

In byte level DFA, we assume that certain bits of a byte is corrupted by the induced fault and the induced difference is confined within a byte. Due to the fact that the fault is induced in the penultimate round, implies that apart from using the differential properties of S-box (as used in the bit level DFA on last round of AES), the attacker also uses the differential properties of the MixColumns operation of AES. As already mentioned in the AES, diffusion is provided using a 4×4 MDS matrix in the MixColumns. Due to this matrix multiplication, if 1 byte difference is induced at the input of a round function, the difference is spread to 4 bytes at the round output.

Figure 8a shows the flow of fault. The induced fault has generated a single byte difference at the input of the ninth round MixColumns. Let f be the byte value of the difference and the corresponding 4-byte output difference is $(2f, f, f, 3f)$, where $2, 1$, and 3 are the elements of the first row of the MixColumns matrix. The 4-byte difference is again converted to (f_0, f_1, f_2, f_3) by the non-linear S-box operation in the tenth round. The ShiftRows operation will shift the differences to four different locations. The attacker has access to the fault-free ciphertext C and faulty ciphertext C^*, which differs only in 4 bytes. Now, we can represent the 4-byte difference $(2f, f, f, 3f)$ in terms of the tenth round key K^{10} and the fault-free and faulty ciphertexts by the following equations:

$$\begin{aligned} 2f &= S^{-1}(C_{0,0} \oplus K^{10}_{0,0}) \oplus S^{-1}(C^*_{0,0} \oplus K^{10}_{0,0}) \\ f &= S^{-1}(C_{1,3} \oplus K^{10}_{1,3}) \oplus S^{-1}(C^*_{1,3} \oplus K^{10}_{1,3}) \\ f &= S^{-1}(C_{2,2} \oplus K^{10}_{2,2}) \oplus S^{-1}(C^*_{2,2} \oplus K^{10}_{2,2}) \\ 3f &= S^{-1}(C_{3,1} \oplus K^{10}_{3,1}) \oplus S^{-1}(C^*_{3,1} \oplus K^{10}_{3,1}) \end{aligned} \tag{11}$$

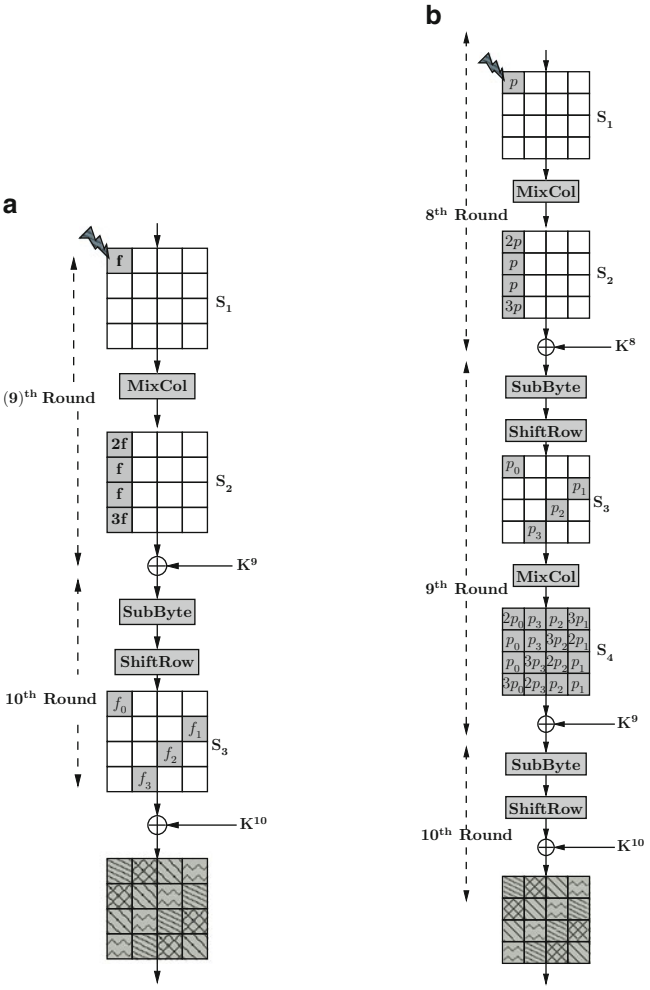


Fig. 8 Flow of faults in AES rounds. (**a**) Differences across the last two rounds. (**b**) Differences across the last three rounds

The above four equations can be expressed as the basic equation (9). Therefore, it can be represented in the form $A = B \oplus C$ where A, B, and C are bytes in $\mathbf{F_{2^8}}$, having 2^8 possible values each. Now a uniformly random choice of (A, B, C) is expected to satisfy the equation with probability $\frac{1}{2^8}$. Therefore, in this case 2^{16} out of 2^{24} random choices of (A, B, C) will satisfy the equation.

This fact can be generalized. Consider we have M such related equations. These M equations consist of N uniformly random byte variables. The probability that a random choice of N variables satisfy all the M equations simultaneously is $(\frac{1}{2^8})^M$. Therefore the reduced search space is given by $(\frac{1}{2^8})^M \cdot (2^8)^N = (2^8)^{N-M}$. For our

case we have four equations which consist of five unknown variables: f, $K^{10}_{0,0}$, $K^{10}_{1,3}$, $K^{10}_{2,2}$, and $K^{10}_{3,2}$. Therefore, the four equations will reduce the search space of the variables to $(2^8)^{5-4} = 2^8$. That means out of 2^{32} hypotheses of the four key bytes, only 2^8 hypotheses will satisfy the above four equations. Therefore, using one fault the attacker can reduce the search space of the four key byte to 2^8. Using two such faulty ciphertexts one can uniquely determine the key quartet. This implies, for one key quartet, one has to induce two faults in the required location. For all the four key quartets i.e., the entire AES key, an attacker needs to induce eight faults. Therefore using eight faulty ciphertexts and a fault-free ciphertext, it is expected to uniquely determine the 128-bit key of AES.

6.1.1 DFA Using Two Faults

The attack can further be improved. It was shown in [45] that instead of inducing fault in ninth round, if we induce fault in between seventh and eighth round MixColumns, we can determine the 128-bit key using only two faulty ciphertexts. Figure 8b shows the spreading of faults when it is induced in such a fashion. The single byte difference at the input of eighth round MixColumns is spread to 4 bytes. The Shiftrows operation ensures that there is one disturbed byte in each column of the state matrix. Each of the 4-byte difference again spreads to 4 bytes at ninth round MixColumns output. Therefore the relation between the fault values in the four columns of difference state matrix S_4 is equivalent to four faults at four different columns of ninth round input state matrix as explained in the previous attack. This implies that using two such faults we can uniquely determine the entire AES key.

Note that the exact working of the DFA proposed in [45] is slightly different from above, though the underlying principle is the same. The attack maintains a list $\mathcal{D}$ for each column of the difference matrix S_4 assuming a 1-byte fault in the input of the penultimate round MixColumns. The size of the table $\mathcal{D}$ is thus 4×255 4-byte values, as the input fault can occur in any byte of a column and can take 255 non-zero values. Assuming that the fault occurs in the difference matrix S_3 in the first column, then equations similar to Eq. (11) can be written, with the left hand side of the equations being a 4-byte tuple $(\Delta_0, \Delta_1, \Delta_2, \Delta_3)$. It is expected that the correct guess of the keys $K^{10}_{0,0}, K^{10}_{1,3}, K^{10}_{2,2}$, and $K^{10}_{3,2}$ should provide a 4-byte tuple which belongs to the list $\mathcal{D}$. There are other wrong keys which also pass this test, and analysis shows that on an average 1036 elements pass this test with a single fault. Repeating the same for all the 4-columns of the difference matrix S_4, reduces the AES key to $1036^4 \approx 2^{40}$ (note that as the fault is assumed to be between seventh and eighth round each column of S_3 has a byte disturbed). However, if two faults are induced, the unique AES key is returned with a probability of 0.98.

This is the best known DFA of AES till date when the attacker does not have access to the plaintext and he needs to determine the key uniquely. However when the attacker has access to the plaintexts, he can still improve the attack by performing the DFA using only one fault and a further reduced brute force guess. Also it is possible to reduce the time complexity of the attack further from 2^{32} to 2^{30}.

6.1.2 DFA Using Only One Fault

The attack proposed in [45] can be further improved when the attacker has access to the plaintexts in addition to the ciphertexts [39]. In that case, he can do brute-force on the possible keys. The objective of this attack or its extensions is to perform the attack using only one fault. While a unique key may not be obtainable with a single fault, the AES key size can reduce to such a small size that a brute force search can be easily performed. It may be noted that reducing the number of fault requirements from 2 to 1 should not be seen in terms of its absolute values. In an actual fault attack, it is very unlikely that the attacker can have absolute control over the fault injection method and hence may need more number of trials. Rather, these attacks are capable of reducing the number of fault requirements by half compared to the attacks proposed in [45].

Consider Fig. 8b, where from the first column of S_4 we get four differential equations [similar to Eq. (11)] corresponding to the 4-tuple $(2p_0,\ p_0,\ p_0,\ 3p_0)$. Using these four differential equations we only guess the 2^8 values of p_0 and get the corresponding possible 2^8 hypotheses of the key quartet by applying the S-box difference distribution table. Therefore, one column of S_4 will reduce the search space of one quartet of key to 2^8 choices. Similarly, solving the differential equations from all the four columns we can reduce the search space of all the four key quartets to 2^8 values each. Hence, if we combine all the four quartets we get $(2^8)^4 = 2^{32}$ possible hypotheses of the final round key K^{10}. We have assumed here that the initial fault value was in the $(0, 0)$th byte of S_1. If we allow the fault to be in any of the 16 locations, the key space of AES is around 2^{36} values. This space can be brute force searched within 1 min and hence, shows that effectively one fault is sufficient to break AES.

The search space of the final round key can be further reduced if we consider the relation between the fault values at the state matrix S_2, which was not utilized in the previous attacks. This step serves as a second stage, which is coupled with the first stage on all the 2^{32} keys (for an assumed location of the faulty byte). We can represent the fault value in the first column of S_2 in terms of the ninth round key K^9 and the ninth round fault-free and faulty output C^9 and C^{*9} respectively by the following four differential equations:

$$\begin{aligned} 2p = S^{-1}(&14(C^9_{0,0} \oplus K^9_{0,0}) \oplus 11(C^9_{1,0} \oplus K^9_{1,0}) \oplus \\ &13(C^9_{2,0} \oplus K^9_{2,0}) \oplus 9(C^9_{3,0} \oplus K^9_{3,0})) \oplus \\ S^{-1}(&14(C^{*9}_{0,0} \oplus K^9_{0,0}) \oplus 11(C^{*9}_{1,0} \oplus K^9_{1,0}) \oplus \\ &13(C^{*9}_{2,0} \oplus K^9_{2,0}) \oplus 9(C^{*9}_{3,0} \oplus K^9_{3,0})) \end{aligned} \tag{12a}$$

$$\begin{aligned} p = S^{-1}(&9(C^9_{0,3} \oplus K^9_{0,3}) \oplus 14(C^9_{1,3} \oplus K^9_{1,3}) \oplus \\ &11(C^9_{2,3} \oplus K^9_{2,3}) \oplus 13(C^9_{3,3} \oplus K^9_{3,3})) \oplus \\ S^{-1}(&9(C^{*9}_{0,3} \oplus K^9_{0,3}) \oplus 14(C^9_{1,3} \oplus K^{*9}_{1,3}) \oplus \\ &11(C^{*9}_{2,3} \oplus K^9_{2,3}) \oplus 13(C^{*9}_{3,3} \oplus K^9_{3,3})) \end{aligned} \tag{12b}$$

$$
\begin{aligned}
p = S^{-1}(13(C^9_{0,2} \oplus K^9_{0,2}) \oplus 9(C^9_{1,2} \oplus K^9_{1,2}) \oplus \\
14(C^9_{2,2} \oplus K^9_{2,2}) \oplus 11(C^9_{3,2} \oplus K^9_{3,2})) \oplus \\
S^{-1}(13(C^{*9}_{0,2} \oplus K^9_{0,2}) \oplus 9(C^{*9}_{1,2} \oplus K^9_{1,2}) \oplus \\
14(C^{*9}_{2,2} \oplus K^9_{2,2}) \oplus 11(C^{*9}_{3,2} \oplus K^9_{3,2}))
\end{aligned} \tag{12c}
$$

$$
\begin{aligned}
3p = S^{-1}(11(C^9_{0,1} \oplus K^9_{0,1}) \oplus 13(C^9_{1,1} \oplus K^9_{1,1}) \oplus \\
14(C^9_{2,1} \oplus K^9_{2,1}) \oplus 9(C^9_{3,1} \oplus K^9_{3,1})) \oplus \\
S^{-1}(11(C^{*9}_{0,1} \oplus K^9_{0,1}) \oplus 13(C^{*9}_{1,1} \oplus K^9_{1,1}) \oplus \\
14(C^{*9}_{2,1} \oplus K^9_{2,1}) \oplus 9(C^{*9}_{3,1} \oplus K^9_{3,1}))
\end{aligned} \tag{12d}
$$

In order to utilize the above equations we need the ninth round key. The ninth round key can be derived from the final round key by the following conversion matrix:

$$
\begin{pmatrix}
(K^{10}_{0,0} \oplus S[K^{10}_{1,3} \oplus K^{10}_{1,2}] \oplus h_{10}) & K^{10}_{0,1} \oplus K^{10}_{0,0} & K^{10}_{0,2} \oplus K^{10}_{0,1} & K^{10}_{0,3} \oplus K^{10}_{0,2} \\
(K^{10}_{1,0} \oplus S[K^{10}_{2,3} \oplus K^{10}_{2,2}]) & K^{10}_{1,1} \oplus K^{10}_{1,0} & K^{10}_{1,2} \oplus K^{10}_{1,1} & K^{10}_{1,3} \oplus K^{10}_{1,2} \\
(K^{10}_{2,0} \oplus S[K^{10}_{3,3} \oplus K^{10}_{3,2}]) & K^{10}_{2,1} \oplus K^{10}_{2,0} & K^{10}_{2,2} \oplus K^{10}_{2,1} & K^{10}_{2,3} \oplus K^{10}_{2,2} \\
(K^{10}_{3,0} \oplus S[K^{10}_{0,3} \oplus K^{10}_{0,2}]) & K^{10}_{3,1} \oplus K^{10}_{3,0} & K^{10}_{3,2} \oplus K^{10}_{3,1} & K^{10}_{3,3} \oplus K^{10}_{3,2}
\end{pmatrix}.
$$

Thus for each of the possible hypotheses of K^{10} produced by the first stage, and using the ciphertexts, (C, C^*), we get the values of (K^9, C^9, C^{*9}). Then the attacker tests the above four equations with these values. If satisfies, the candidate key is accepted, else rejected. For completeness, we state the detailed equations as follows:

$$
\begin{aligned}
2p = S^{-1}\big[14(S^{-1}[K^{10}_{0,0} \oplus C_{0,0}] \oplus K^{10}_{0,0} \oplus S[K^{10}_{1,3} \oplus K^{10}_{1,2}] \oplus h_{10}) \oplus \\
11(S^{-1}[K^{10}_{1,3} \oplus C_{1,3}] \oplus K^{10}_{1,0} \oplus S[K^{10}_{2,3} \oplus K^{10}_{2,2}]) \oplus \\
13(S^{-1}[K^{10}_{2,2} \oplus C_{2,2}] \oplus K^{10}_{2,0} \oplus S[K^{10}_{3,3} \oplus K^{10}_{3,2}]) \oplus \\
9(S^{-1}[K^{10}_{3,1} \oplus C_{3,1}] \oplus K^{10}_{3,0} \oplus S[K^{10}_{0,3} \oplus K^{10}_{0,2}])\big] \oplus \\
S^{-1}\big[14(S^{-1}[K^{10}_{0,0} \oplus C^*_{0,0}] \oplus K^{10}_{0,0} \oplus S[K^{10}_{1,3} \oplus K^{10}_{1,2}]) \oplus \\
11(S^{-1}[K^{10}_{1,3} \oplus C^*_{1,3}] \oplus K^{10}_{1,0} \oplus S[K^{10}_{2,3} \oplus K^{10}_{2,2}]) \oplus \\
13(S^{-1}[K^{10}_{2,2} \oplus C^*_{2,2}] \oplus K^{10}_{2,0} \oplus S[K^{10}_{3,3} \oplus K^{10}_{3,2}]) \oplus \\
9(S^{-1}[K^{10}_{3,1} \oplus C^*_{3,1}] \oplus K^{10}_{3,0} \oplus S[K^{10}_{0,3} \oplus K^{10}_{0,2}])\big]
\end{aligned} \tag{13}
$$

Similarly, the other three faulty bytes can be expressed by the following equations:

$$\begin{aligned} p = S^{-1}\big[&9(S^{-1}[K^{10}_{0,3} \oplus C_{0,3}] \oplus K^{10}_{0,3} \oplus K^{10}_{0,2}) \oplus \\ &14(S^{-1}[K^{10}_{1,3} \oplus C_{1,3}] \oplus K^{10}_{1,3} \oplus K^{10}_{1,2}) \oplus \\ &11(S^{-1}[K^{10}_{2,1} \oplus C_{2,1}] \oplus K^{10}_{2,3} \oplus K^{10}_{2,2}) \oplus \\ &13(S^{-1}[K^{10}_{3,0} \oplus C_{3,0}] \oplus K^{10}_{3,3} \oplus K^{10}_{3,2})\big] \oplus \\ S^{-1}\big[&9(S^{-1}[K^{10}_{0,3} \oplus C_{0,3}] \oplus K^{10}_{0,3} \oplus K^{10}_{0,2}) \oplus \\ &14(S^{-1}[K^{10}_{1,3} \oplus C_{1,3}] \oplus K^{10}_{1,3} \oplus K^{10}_{1,2}) \oplus \\ &11(S^{-1}[K^{10}_{2,1} \oplus C_{2,1}] \oplus K^{10}_{2,3} \oplus K^{10}_{2,2}) \oplus \\ &13(S^{-1}[K^{10}_{3,0} \oplus C_{3,0}] \oplus K^{10}_{3,3} \oplus K^{10}_{3,2})\big] \oplus \end{aligned} \tag{14}$$

$$\begin{aligned} p = S^{-1}\big[&13(S^{-1}[K^{10}_{0,2} \oplus C_{0,2}] \oplus K^{10}_{0,2} \oplus K^{10}_{0,1}) \oplus \\ &9(S^{-1}[K^{10}_{1,1} \oplus C_{1,1}] \oplus K^{10}_{1,2} \oplus K^{10}_{1,1}) \oplus \\ &14(S^{-1}[K^{10}_{2,0} \oplus C_{2,0}] \oplus K^{10}_{2,2} \oplus K^{10}_{2,1}) \oplus \\ &11(S^{-1}[K^{10}_{3,3} \oplus C_{3,3}] \oplus K^{10}_{3,2} \oplus K^{10}_{3,1})\big] \oplus \\ S^{-1}\big[&13(S^{-1}[K^{10}_{0,2} \oplus C^{*}_{0,2}] \oplus K^{10}_{0,2} \oplus K^{10}_{0,1}) \oplus \\ &9(S^{-1}[K^{10}_{1,1} \oplus C^{*}_{1,1}] \oplus K^{10}_{1,2} \oplus K^{10}_{1,1}) \oplus \\ &14(S^{-1}[K^{10}_{2,0} \oplus C^{*}_{2,0}] \oplus K^{10}_{2,2} \oplus K^{10}_{2,1}) \oplus \\ &11(S^{-1}[K^{10}_{3,3} \oplus C^{*}_{3,3}] \oplus K^{10}_{3,2} \oplus K^{10}_{3,1})\big] \end{aligned} \tag{15}$$

$$\begin{aligned} 3p = S^{-1}\big[&11(S^{-1}[K^{10}_{0,1} \oplus C_{0,1}] \oplus K^{10}_{0,1} \oplus K^{10}_{0,0}) \oplus \\ &13(S^{-1}[K^{10}_{1,0} \oplus C_{1,0}] \oplus K^{10}_{1,1} \oplus K^{10}_{1,0}) \oplus \\ &14(S^{-1}[K^{10}_{2,3} \oplus C_{2,3}] \oplus K^{10}_{2,1} \oplus K^{10}_{2,0}) \oplus \\ &9(S^{-1}[K^{10}_{3,2} \oplus C_{3,2}] \oplus K^{10}_{3,1} \oplus K^{10}_{3,0})\big] \oplus \\ S^{-1}\big[&11(S^{-1}[K^{10}_{0,1} \oplus C^{*}_{0,1}] \oplus K^{10}_{0,1} \oplus K^{10}_{0,0}) \oplus \\ &13(S^{-1}[K^{10}_{1,0} \oplus C^{*}_{1,0}] \oplus K^{10}_{1,1} \oplus K^{10}_{1,0}) \oplus \\ &14(S^{-1}[K^{10}_{2,3} \oplus C^{*}_{2,3}] \oplus K^{10}_{2,1} \oplus K^{10}_{2,0}) \oplus \\ &9(S^{-1}[K^{10}_{3,2} \oplus C^{*}_{3,2}] \oplus K^{10}_{3,1} \oplus K^{10}_{3,0})\big] \end{aligned} \tag{16}$$

We thus get four differential equations and the combined search space of (K^9, C^9, C^{*9}) and p is $2^{32} \cdot 2^8 = 2^{40}$. Therefore, the four equations will reduce this search space of K_{10} to $\frac{2^{40}}{(2^8)^4} = 2^8$. Hence, using only one faulty ciphertext,

one can reduce the search space of AES-128 key to 256 choices. However, the time complexity of the attack is 2^{32}, as we have to test all the hypothesis of K^{10} by the above four equations. In the next subsection, we present an improvement to reduce the time complexity of the attack to 2^{30} from 2^{32}.

6.1.3 DFA with Reduced Time Complexity

The above second phase of the analysis is based on four equations: (13), (14), (15), and (16). All the 2^{32} possible key hypotheses are tested by these four equations. The key hypotheses which are satisfied by all four equations are considered and rest are discarded.

However, if we consider the above four equations in pairs we observe that each possible pair does not contain all the 16 bytes of the AES key. For example, the pair of Eqs. (14) and (15) contains 14 key bytes excluding $K^{10}_{0,0}$ and $K^{10}_{0,1}$. This fact can be utilized to reduce the time complexity of the attack. We use this observation to split the lists of key which are exported in the first phase of the attack and subsequently filtered in the second phase.

In the first phase of the attack we have four quartets $(K^{10}_{0,0}, K^{10}_{1,3}, K^{10}_{2,2}, K^{10}_{3,1})$, $(K^{10}_{0,1}, K^{10}_{1,0}, K^{10}_{2,3}, K^{10}_{3,2})$, $(K^{10}_{0,2}, K^{10}_{1,1}, K^{10}_{2,0}, K^{10}_{3,3})$, and $(K^{10}_{0,3}, K^{10}_{1,2}, K^{10}_{2,1}, K^{10}_{3,0})$ Let us assume one value of the first quartet is (a_1, b_1, c_1, d_1). As per the property of the S-Box, there will be another value a_2 of $K^{10}_{0,0}$, which satisfies the system of equations generated from the first column of S_4, with the rest of the key byte values remaining same.

Using this idea, we can divide the list for the quartet $(K^{10}_{0,0}, K^{10}_{1,3}, K^{10}_{2,2}, K^{10}_{3,1})$ into two sublists, L_1, L_2. As depicted in Fig. 9 The list L_1 contains the pair values for the key byte $K^{10}_{0,0}$ (note that the key byte $K^{10}_{0,0}$ has always an even number of possible choices). The list L_2 contains the distinct values for the remaining part of the quartet, $(K^{10}_{1,3}, K^{10}_{2,2}, K^{10}_{3,1})$. Thus the expected size of the lists L_1 and L_2 is 2^7 each, compared to the previous list size of 2^8 with 4-tuple $(K^{10}_{0,0}, K^{10}_{1,3}, K^{10}_{2,2}, K^{10}_{3,1})$.

Similarly, we store the possible values of quartet $(K^{10}_{0,1}, K^{10}_{1,0}, K^{10}_{2,3}, K^{10}_{3,2})$ in two lists, L_3 and L_4. Here L_3 stores the pair values for the key byte $K^{10}_{0,1}$, while the list L_4 contains the distinct values for the key bytes $(K^{10}_{1,0}, K^{10}_{2,3}, K^{10}_{3,2})$. Next, we select the key bytes from the six lists, $L_1, L_2, L_3, L_4, L_5, L_6$, to solve the equations of the second phase of the attack such that the time complexity is reduced.

Because of the observations regarding the pair of Eqs. (13), (16) and (14), (15), the second phase can be divided into two parts. In part one, we test the keys generated from the first phase of the attack by the pair of Eqs. (14) and (15). In Fig. 9 this is denoted as *Test1*. As the two equations for *Test1* does not require key bytes $K^{10}_{0,0}$ and $K^{10}_{0,1}$, we only consider all possible keys generated from lists L_2, L_4, L_5, L_6. There are 2^{30} such possible keys. In the second part we combine each of the 14 byte keys satisfying *Test1* with one of the four possible values arising out of the four combinations of the pair of values for $K^{10}_{0,0}$ in L_1 and $K^{10}_{0,1}$ in L_3. These keys are further tested in parallel by Eqs. (13) and (16). In Fig. 9, we refer to this test as *Test2*.

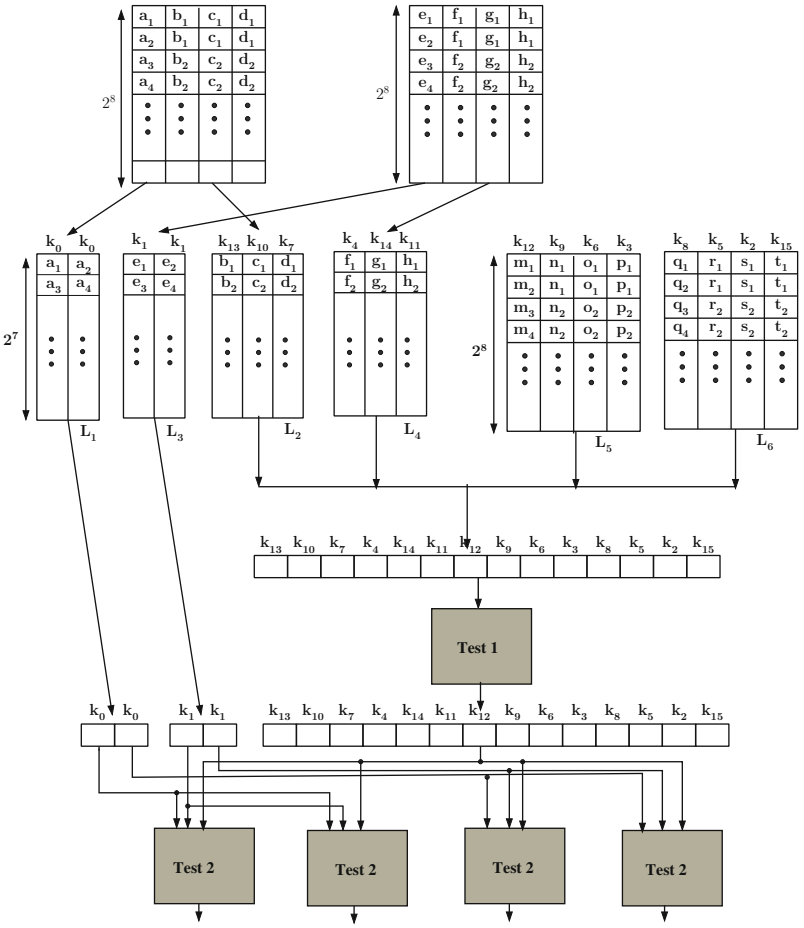


Fig. 9 Model for data-flow parallelization in the second phase

The size of the lists L_2 and L_4 is 2^7; and the size of lists L_5 and L_6 is 2^8. Therefore the number of possible keys generated from this four lists is $2^7 \times 2^7 \times 2^8 \times 2^8 = 2^{30}$. These 2^{30} keys are fed as input to *Test1* which is expected to reduce the key hypotheses by 2^8. Therefore each instance of *Test2* will receive input of $(\frac{2^{30}}{2^8}) = 2^{22}$ expected key hypotheses. The chance of each key satisfying *Test2* is 2^{-16} which implies each instance of *Test2* will result in 2^6 key hypotheses.

It may be easily observed that the time required is because of step 3, which is equal to 2^{30}, making the overall attack four times faster on an average, and still reducing the overall keyspace of AES to around 2^8 values.

The above fault models are based on single byte fault models, which assume that the fault is localized in a single byte. However due to impreciseness in the fault induction, the fault can spread to more than 1 bytes. Such a *multiple-byte* fault

requires a revisit at the DFA methods. In [47], a technique for performing DFA when such faults occur where presented, which generalize further the DFA proposed in [6] and later extended in [30]. The underlying fault models assumed in this attack were already introduced in Sect. 4.1.3 and were called as *diagonal fault models*. In the next section, we outline the idea of these attacks.

7 Multiple Byte DFA of AES-128

In this section, we present the DFAs under the multiple byte fault models. The DFAs are efficient to obtain the AES key using 2–4 faults, when the faults corrupt upto three diagonals of the four diagonals of the AES state matrix at the input of the eighth round MixColumns. In the next subsection, we first observe the DFAs when the fault is confined to one diagonal of the state matrix, i.e., the fault is according to the fault model $DM0$.

*7.1 DFA According to Fault Model DM*0

We first show that faults which are confined to one diagonal are equivalent and can be used to retrieve the key using the same method.

7.1.1 Equivalence of Faults in the Same Diagonal

Let us first observe the propagation of a fault injected in diagonal D_0 through the round transformations from the input of the eighth round to the output of the ninth round.

Figure 10 shows some cases of fault induction in diagonal D_0. The faults vary in the number of bytes that are faulty in D_0 at the input of the 8 round. We emphasize the fact that irrespective of the number or positions of bytes that are faulty in D_0, due to the subsequent ShiftRows operation the fault is confined to the first column C_0 of the state matrix at the end of the eighth round. So the fault propagation in the ninth round for all these cases is similar and leads to the same byte inter-relations at the end of the ninth round.

In general any fault at the input of the eighth round in the ith diagonal, $0 \leq i \leq 3$, leads to the ith column being affected at the end of the round. There are four diagonals and faults in each diagonal maps to four different byte inter-relations at the end of the ninth round. These relations are depicted in Fig. 11. These relations will remain unchanged for any combination of faults injected within a particular diagonal. Each of the four sets of relations in Fig. 11 will be used to form key dependent equations. Each of the equation sets will comprise of four equations of similar nature as shown in Eq. (11).

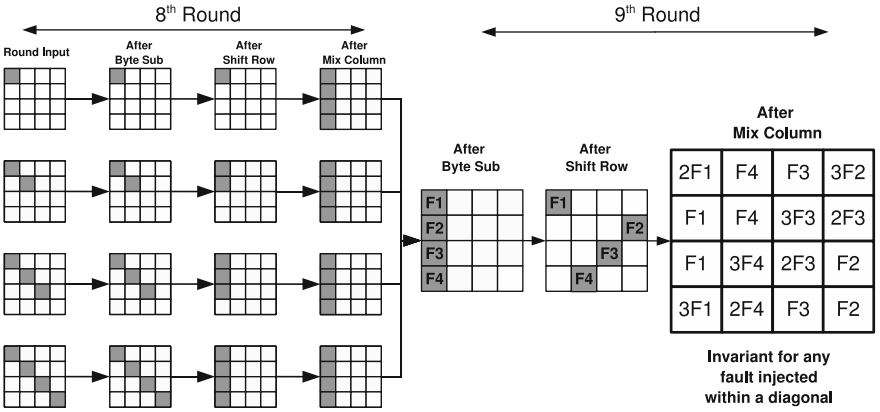


Fig. 10 Equivalence of different kinds of faults induced in diagonal D_0 at the input of eighth round of AES

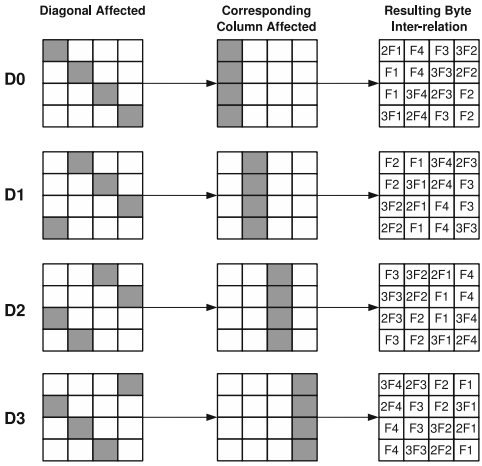


Fig. 11 Byte inter-relations at the end of ninth round corresponding to different diagonals being faulty

As before these equations reduce the AES key to an average size of 2^{32}. If the attacker is unaware of the exact diagonal, he can repeat for all the above four sets of equations, and the key size will still be $2^{32} \times 4 = 2^{34}$, which can be brute forced feasibly with present day computation power.

Next, we consider briefly the cases when the faults spread to more than one diagonal.

*7.2 DFA According to Fault Model DM*1

In Fig. 12, we observe the propagation of faults when the diagonals, D_0 and D_1 are affected at the input of the ninth round MixColumns.

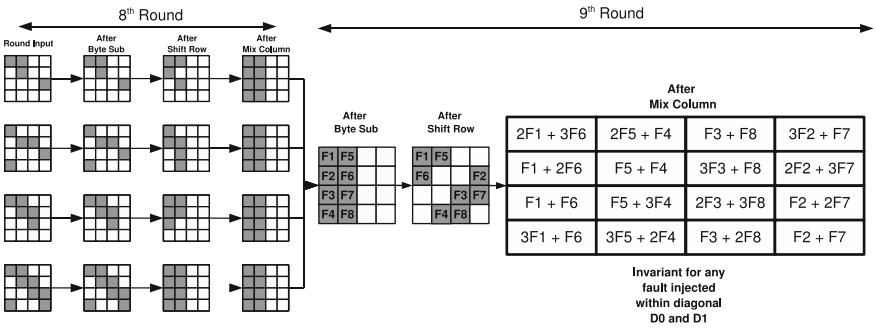


Fig. 12 Fault propagation if diagonals D_0 are D_1 are affected

We observe that the nature of the faults in the state matrix at the input of the ninth round MixColumns and hence at the output remains invariant for all possible faults in these two diagonals. This property is exploited to develop equations which are used to retrieve the correct key.

We denote the fault values in the first column of the output of the ninth round MixColumns by a_0, a_1, a_2, a_3, where each a_i is a byte $0 \leq i \leq 3$. Then using the inter-relationships among the faulty bytes one can easily show that:

$$a_1 + a_3 = a_0$$
$$2a_1 + 3a_3 = 7a_2$$

We can express a_0, a_1, a_2, a_3 in terms of the fault free ciphertext (CT), faulty ciphertext (CT^*) and 4 bytes of the tenth round key ($\mathbf{K_{10}}$). The equations reduce the key space of 4 bytes of the key to 2^{16}. Similarly, performing the analysis for other columns, helps to reduce the AES key to a size of $(2^{16})^4 = 2^{64}$. Using two such faulty inductions it is expected that the unique key is returned.

Depending on the combination of two diagonals affected out of the four diagonals, there are six such sets of equations. Hence even in such case, the attacker reduces the AES key space to 6 possible keys, which he can easily brute force.

In the next section, we present an attack strategy if the fault gets spread to atmost three diagonals. This fault model, $DM2$ thus covers the first two models of fault.

7.3 DFA According to Fault Model DM2

In Fig. 13, we observe the propagation of faults when the diagonals, D_0, D_1 and D_2 are affected.

From Fig. 13, we note that for all possible faults corrupting the diagonals D_0, D_1 and D_2, the nature of the faults at the input of the ninth round MixColumns is an invariant. The fault nature at the output of the ninth round MixColumns is as seen

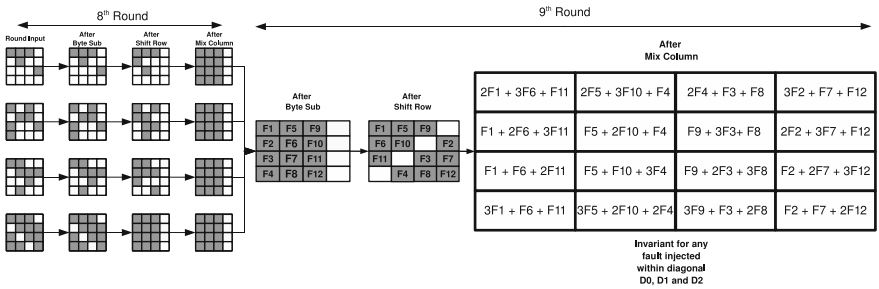


Fig. 13 Fault propagation if diagonals D_0, D_1 and D_2 are affected

in the figure, also an invariant. We denote the fault values in the first column of the output of the ninth round MixColumns by a_0, a_1, a_2, a_3, where each a_i is a byte $0 \leq i \leq 3$.

The following equation can be obtained by observing the inter-relationships (refer Fig. 12):

$$11a_0 + 13a_1 = 9a_2 + 14a_3$$

As before in case of faults modeled by $DM1$, we can express a_0, a_1, a_2, a_3 in terms of the fault free ciphertext (CT), faulty ciphertext (CT') and the tenth round key ($\mathbf{K_{10}}$). One equation reduces 4 bytes of the key to 2^{24} values. We can have similar equations for each of the remaining three columns of the state matrix after the ninth round MixColumns, and thus the AES key space reduces to an expected value of $(2^{24})^4 = 2^{96}$. However, using four faults and taking the intersection of the key space, it is expected that the key space reduces to a unique value.

It may be noted that when the faults occur according to the model $DM3$, that is all the four diagonals are affected, the DFA fails.

8 Extension of the DFA to Other Variants of AES

In the previous sections we described DFAs using different fault models on AES with 128-bit key. However, AES has two more variants: AES-192 and AES-256 with key length 192 and 256 bits. These to variants of AES follows different key scheduling. If we observe the key scheduling algorithm, we see that for AES-192 and AES-256, last round is not sufficient to retrieve the master key. It requires to retrieve the last two round keys rather any two consecutive round keys. For the sake of simplicity we consider the last two round keys. In case of AES-192, the last round key and the last two columns of penultimate round key is sufficient. Because the first two columns of penultimate round key can be directly derived from the final round key.

The first complete DFA on AES-192 and AES-256 was proposed in [33]. The proposed attacks were based on two different fault models which requires 6 and 3000 pairs of fault free and faulty ciphertexts. A new attack was proposed in [52] which first time exploited the relations between the round keys of the key scheduling algorithm. The attack on AES-192 required three pairs of correct and faulty ciphertexts and the attack on AES-256 required two pairs of correct and faulty ciphertexts and two pairs of correct and faulty plaintexts. The attack was further improved in [29] where the DFA on AES-192 required two pairs of fault free and faulty ciphertexts, and on AES-256 required three pairs of fault free and faulty ciphertexts. Recently a DFA on AES-256 was proposed in [4], which required two pairs of fault-free and faulty ciphertexts and a brute-force search of 16 bits with attack time complexity of 2^{32}. This is the best known attack on AES-256 till date. More details of the attacks on the other versions of AES can be obtained from [14].

9 DFA of AES Targeting the Key-Schedule

In the previous sections we described how an induced difference at the state of a particular round of AES can be exploited to reveal the secret key. In order to protect AES from such attacks a designer has to use some countermeasures which will not allow the attacker to induce faults in AES round operations. Even if fault is induced, the attacker will not be able to get the faulty ciphertexts to apply a DFA. Subsequently, the attackers have developed new attacking technique known as DFA on AES key schedule which work even if the rounds of the AES are protected against faults. In this kind of DFAs, faults are induced at the round keys. Therefore, even if the rounds are protected against DFA, the attack will work as the protection will not be able to distinguish between a fault-free round key and a faulty round key.

However, the DFA on AES key schedule are more challenging than DFA on AES state. A difference induced in a round key will spread to more number of bytes in the subsequent round keys during the key schedule operation, which in turn creates more number of unknown variables in the differential equations. Therefore, the differential equations are more complex than the differential equations in a DFA on AES state.

The first complete DFA on AES key schedule was proposed in [13]. The attack was targeted on AES-128 and required less than 30 pairs of fault-free and faulty ciphertexts. An improved attack in [53] showed that a DFA on AES key schedule is possible using two pairs of fault-free and faulty ciphertexts and a brute-force search of 48-bit. Subsequently, there are two more attacks proposed in [32] and [31] using two pairs of fault-free and faulty ciphertexts each. Most optimum attack on AES key schedule was proposed in [3, 5], which required only one pair of fault-free and faulty ciphertexts. The attack used a complex divide and conquer strategy to solve the differential equations. The most recent attacks on AES, both state and key schedule, can be found in [7].

Table 2 Comparison of CED techniques

CED		†Fault coverage	CED/original	
			Throughput (%)	Hardware (%)
Hardware redundancy	[18]	91 %	∼100	∼200
	[42]	25 %	∼100	126
Time redundancy	[35]	99.9 %	∼100	136
	[46]	100 %	50–90.9	102.3
Information redundancy	[55]	48–53 %	∼100	122.3
	[9]	99.997 %	67.86	188.9
	[41]	99.20 %	–	137.35
	[19]	$1 - 2^{-56}$	87	177
Hybrid redundancy	[22]	100 %	73.45	197.6
	[48]	–	85.6	188.9

† Fault coverage are based on multiple bit random fault model, ∼ Estimated values: authors did not give precise figures

10 CED for AES

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) [50]. Cryptographic chips are sensitive to faults in the hardware. A small number of excited faults can cause a large number of output bits of AES to be faulty [9]. As previously explained, attackers have injected faults into cryptographic circuits to steal secret information. Previous work on CED can be classified into four types of redundancy: hardware, time, information, and hybrid redundancy. Table 2 shows a comparison of different CED techniques based on fault coverage, throughput and hardware utilization. The fault coverage is obtained from the multiple bit random fault model. The throughput and hardware are the ratio between the CED implementation and the original.

10.1 Hardware Redundancy

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

In [18], the authors propose a novel hardware redundancy technique for AES to detect faults. Because an attacker can potentially inject the same faults to both of the AES circuits, the straightforward hardware redundancy can be bypassed by the attacker. As shown in Fig. 14, the idea is to mix byte states between the operations in

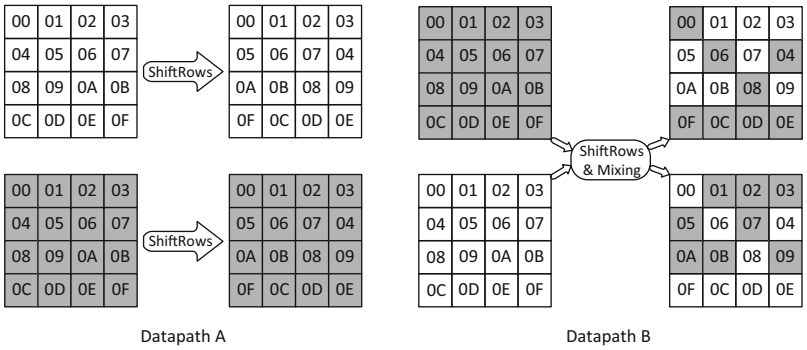


Fig. 14 Datapath mixing (byte level)

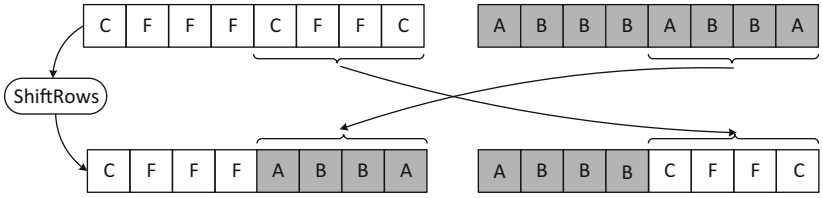


Fig. 15 Datapath mixing (bit level)

two pieces of hardware, in different ways and at different locations. The mixing can be done at byte level or bit level as shown in Figs. 14 and 15. Because the attacker does not know how the circuits are mixed, the attacker needs to reverse engineer the circuit layout to figure out where to inject the faults. Although this technique adds an extra layer of obscurity on top of the straightforward hardware redundancy, its effectiveness is yet to be proven. It also requires significant changes to the AES datapath. Because the entire hardware is duplicated, hardware redundancy has high fault coverage, low fault detection latency, and low performance overhead. However, because one extra piece of hardware and comparison circuitries are needed, the hardware overhead is approximately 200 % as shown in Table 2. It provides 91 % fault coverage and the performance is close to the original implementation.

To reduce the hardware overhead, [42] proposes a partial hardware redundancy technique as shown in Fig. 16. This technique focuses on parallel AES architecture and S-box protection. The idea is to add an additional S-box to every set of four S-boxes, and perform two tests of every S-box per encryption cycle (10 rounds). Although the hardware overhead is reduced to 26 %, this process has a fault coverage of 25 % at a certain clock cycle, because it can only check one S-box among every four in one clock cycle.

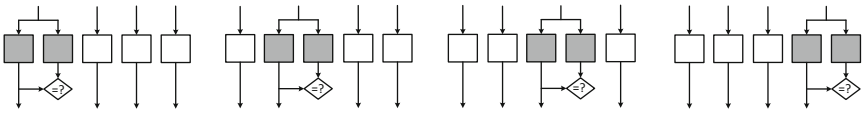


Fig. 16 Partial hardware redundancy

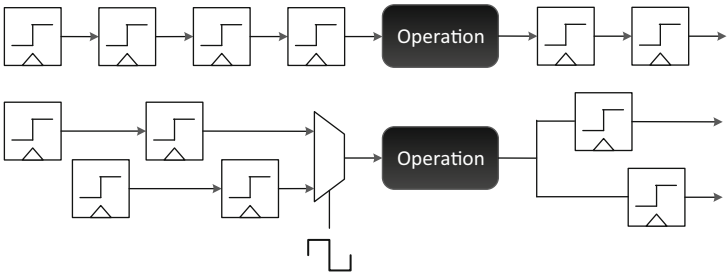


Fig. 17 Candidate operation for DDR application, and operation computing after DDR application

10.2 Time Redundancy

Time redundancy computes the same input twice using the same function and compares the results.

In [35], the authors propose time redundancy with a Double-Data-Rate (DDR) mechanism as shown in Figs. 17 and 18. The pipelined AES data path logic is partitioned into two classes, where nonadjacent stages in the pipeline are driven by two opposite clock signals. The DDR architecture allows us to halve the number of clock cycles per round, though maybe with a light impact on clock frequency as compared to a design without protection. This takes advantage of the free cycles for recomputing the round on the same hardware. Two successive round operation outputs obtained from two copies of the same input data are checked for possible mismatches. It shows an almost maximal fault coverage on the datapath at the cost of 36 % hardware overhead. Under some conditions, this technique allows the encryption to be computed twice without affecting the global throughput. However, this technique becomes difficult to implement as technology scales.

A technique that is suited for any pipeline-based block cipher design is proposed in [46]. The key idea is to use different pipeline stages to check against each other by shifting the computation from one stage to another. Let us assume the pipeline has n stages as shown in Fig.19. In the normal computation, the plaintext will be computed by the first stage and the then the second stage and so on. The nth stage will produce the ciphertext. In the CED computation, the plaintext will be computed by the nth stage, and then the output of the nth stage will be computed by the first stage. Therefore, the output of the $(n-1)$th stage is the ciphertext and it will be compared with the previous ciphertext. Compared to the original design, this CED provides a throughput of 50–90.0 %, depending on the frequency of the redundant check from every one to ten rounds. Hardware overhead is only 2.3 %.

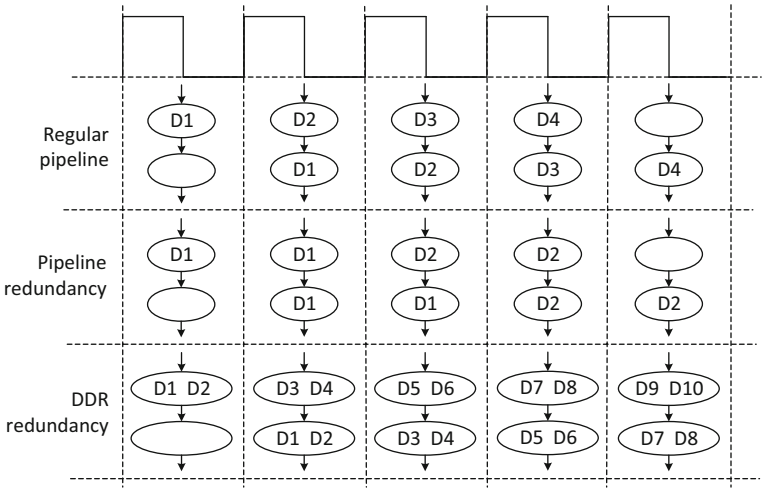
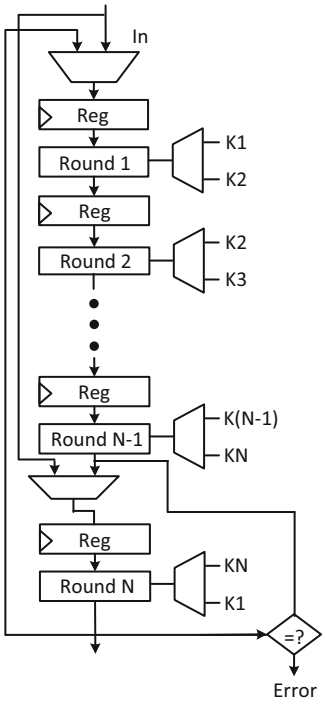


Fig. 18 Scheduling of a regular pipeline, pipeline redundancy and the DDR redundancy (D_x is data token X)

Fig. 19 Sliced architecture



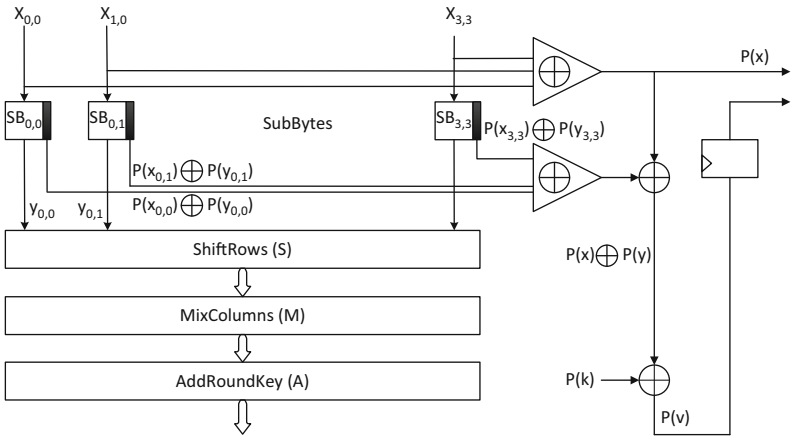


Fig. 20 Parity-1 CED

10.3 Information Redundancy

Information redundancy techniques are based on error detecting codes (EDC). 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. Parity code and robust code are proposed for CED in various research [9, 24, 25, 27, 37–39].

10.3.1 Parity-1

A technique in which a parity bit is used for the entire 128-bit state matrix is developed in [55]. The parity bit is checked once for the entire round as shown in Fig. 20. This approach targets low-cost CED. Parity-1 is based on a general CED design for Substitution Permutation Networks (SPN) [23], in which the input parity of SPN is modified according to its processing steps into the output parity and compared with the output parity of every round. The authors adapt this general approach to develop a low-cost CED. First, they determine the parity of the 128-bit input using a tree of XOR gates. Then for the nonlinear S-box, inversion in $GF(2^8)$ and a linear affine transformation. They add one additional binary output to each of the 16 S-boxes. This additional S-box output computes the parity of every 8-bit input and the parity of the corresponding 8-bit output.

Each of the modified S-boxes is 8-bit by 9-bit. The additional single-bit outputs of the 16 S-boxes are used to modify the input parity for SubBytes. Because ShiftRows implements a permutation, it does not change the parity of the entire state matrix from its input to output. MixColumns does not change the parity of the state matrix from inputs to outputs either. Moreover, MixColumns does not

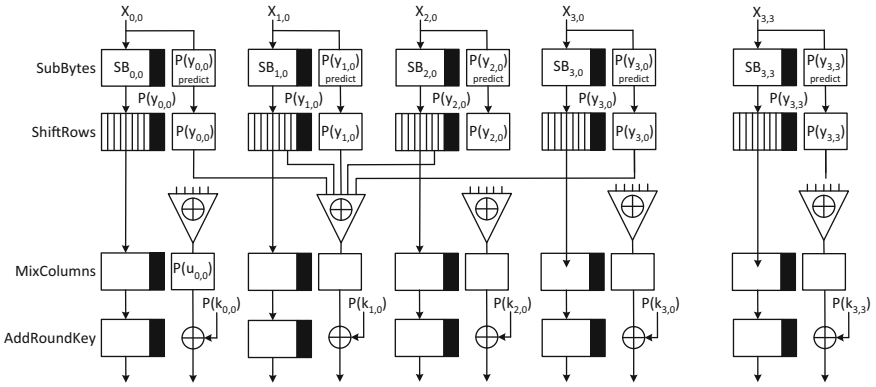


Fig. 21 Parity-16 CED

change the parity of each column. Finally, the bit-wise XOR of the 128-bit round key needs a parity modification by a single precomputed parity bit of the round key. Because the output of a round is the input to the next round, the output parity of a round can be computed with the same hardware for computing the input parity of the previous round.

Although this technique has only 22.3 % hardware overhead, it has 48–53 % fault coverage for multiple bit random fault model.

10.3.2 Parity-16

Parity-16 is first proposed in [9]. In this technique, 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.

In [9], the authors propose the use of a parity bit that is associated with each byte of the state matrix of a 128-bit iterated hardware implementation with LUT-based S-Boxes as shown in Fig. 21. Predicted parity bits on S-Box outputs are stored as additional bits in the ROMs (nine bits instead of eight in the original S-Boxes). In order to detect errors in the memory, the authors propose increasing each S-box to 9-bit by 9-bit in such a way that all the ROM words addressed with a wrong input address (i.e. S-Boxes input with a wrong associated parity), deliberately store values with a wrong output parity so that the CED will detect the fault. As before, the parity bit associated with each byte is not affected by ShiftRows. In Parity-1, the global parity bit on the 128 bits remains unchanged after MixColumns. Conversely, at the byte level, the parity after MixColumns is affected. Therefore, parity-16 requires the implementation of prediction functions in MixColumns. Finally, the parity bits after AddRoundKey are computed as before, by adding the current parity bits to those of the corresponding round key. This technique incurs 88.9 % hardware overhead because of the LUT size is doubled. The throughput is 67.86 % of the original.

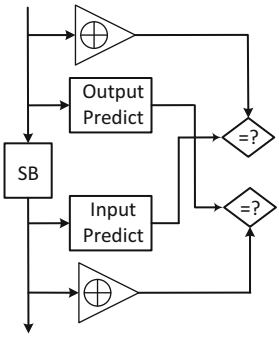


Fig. 22 Parity-32 CED

10.3.3 Parity-32

As shown in Fig. 22, a technique that strengthen fault detection on S-Boxes is proposed in [41]. With respect to parity-16, this technique still uses one parity bit for each byte in all the operations except SubBytes. It adds one extra parity bit for each S-box in SubBytes; one parity bit for the input byte and one for the output byte. The actual output parity is compared with the predicted output parity, and the actual input parity bit is compared with the predicted input parity. It has 37.35 % hardware overhead and 99.20 % fault coverage.

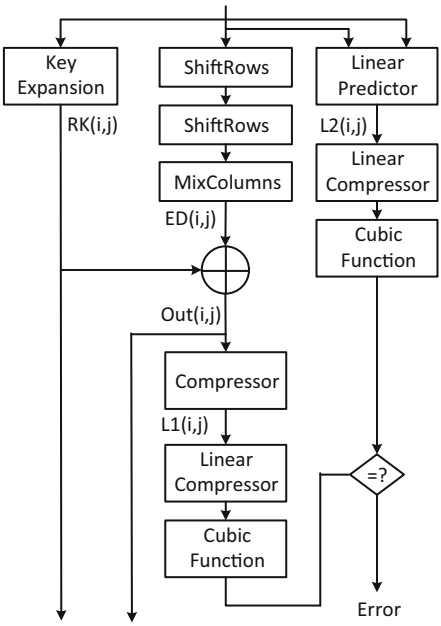
10.3.4 Parity Code vs Residue Code

In [12], the authors try to apply EDCs in a systematic way by investigating 11 different symmetric key encryption algorithms. They study the feasibility of two EDCs for different cryptographic operations mainly from a hardware overhead point of view. For EDCs, parity and residue code are considered. Cryptographic algorithms are divided into XOR, AND/OR, finite field addition/subtraction (mod n), finite field multiplication [mod n and mod $G(x)$], expansion, S-Box, word rotation, word shift, and permutation. The conclusion is that parity code is superior to residue code in logic operations, while residue code is more feasible for arithmetic operations. The authors recommend one parity bit per byte for the AES error detection from a low-cost and high fault coverage point of view.

10.3.5 Robust Code

Robust codes is first proposed in [19]. The idea is to use non-linear EDC instead of linear. Robust codes can be used to extend the error coverage of any linear prediction technique for AES. The advantage of non-linear EDC is because it has uniform fault coverage. If all the data vectors and error patterns are equiprobable, then the probability of injecting an undetectable fault is the same for all of them.

Fig. 23 Robust code



The architecture of AES with robust protection is presented in Fig. 23. In this architecture, two extra unit are needed. One is the prediction unit at the round input, and it includes a linear predictor, a linear compressor, and a cubic function. The other one is the comparison unit at the output of the round, and it includes a compressor, a linear compressor, and a cubic function This architecture protects the encryption and decryption as well as key expansion.

Let us first introduce the prediction unit. A linear predictor and linear compressor is designed to generate an 32-bit output, and we call them the linear portion in the rest of the chapter. The output of the linear portion is linearly related to the output of the round of AES as shown in Fig. 23. They offer a relatively compact design compared to the original round of AES. They simplify the round function by XORing the bytes in the same column. The effect of MixColumns is removed by the linear portion. As a result, the linear portion is greatly simplified as it no longer needs to perform multiplication associated with the MixColumns or InvMixColumns. For the cubic function, the input of is cubed in $GF(2^r)$ to produce the r-bit output, and thus it is non-linear with respect to the output of the round.

In the comparison unit, the compressor and the linear compressor are designed to generate a 32-bit output from the 128-bit round output. The bytes in the same column of the output is XORed. Again, the 32-bit output is cubed in the cubic function to generate r-bit output. This output is then compared with the output from the prediction unit.

This technique provides $1 - 2^{-56}$ fault coverage, and it has a 77 % hardware overhead.

Although these countermeasures can thwart DFA, the designer needs to be cautious when implementing information redundancy-based techniques. Because they increase the correlation of the circuit power consumption with the processed data, the side channel leakage is also increased [34].

10.4 Hybrid Redundancy

In [20–22], the authors consider CED at the operation, round, and algorithm levels for AES. In these schemes, an operation, a round, or the encryption and decryption are followed by their inverses. To detect faults, the results are compared with the original input.

The underlying assumption is that a complete encryption device operating in ECB mode consists of encryption and decryption modules, Thus, a low-cost and low-latency systematic CED is proposed for encryption and decryption datapaths. They describe algorithm-, round-, and operation-levels CEDs that exploit the inverse relationship properties of AES. Because AES uses the same set of round keys for both encryption and decryption, they can be generated a priori, stored in the key RAM, and retrieved in any order depending upon whether encryption or decryption is in progress. They then extend the proposed techniques to full duplex mode by trading off throughput and CED capability.

As shown in Fig. 24a, the algorithm-level CED approach exploits the inverse relationship between the entire encryption and decryption. Plaintext is first processed

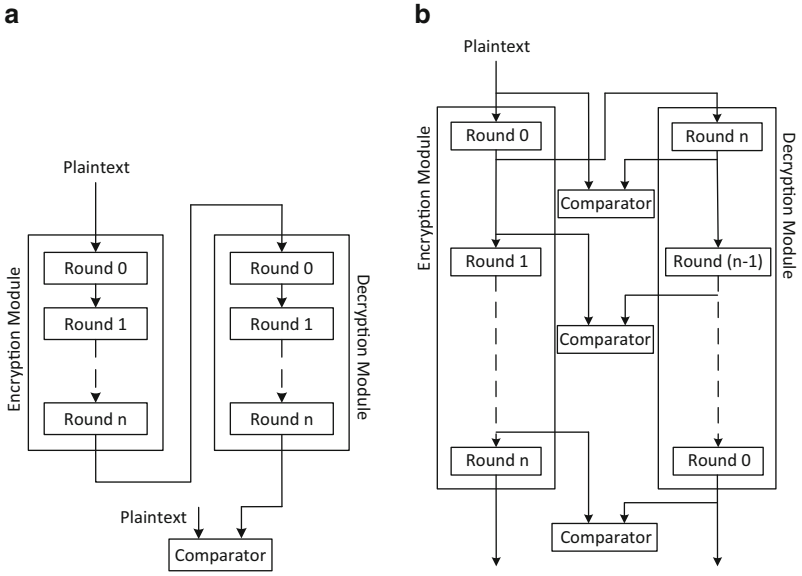


Fig. 24 Hybrid redundancy. (**a**) Algorithm level. (**b**) Round level

by the encryption module. After the ciphertext is available, the decryption module is enabled to decrypt the ciphertext. While the decryption module is decrypting ciphertext, the encryption module can process the next block of data or be idle. A copy of plaintext is also temporarily stored in a register. The output of decryption is compared with this copy of the input plaintext. If there is a mismatch, an error signal will be raised, and the faulty ciphertext will be suppressed.

For AES, the inverse relationship between encryption and decryption exists at the round level as well. Any input data passed successively through one encryption round are recovered by the corresponding decryption round.

For almost all the symmetric block cipher algorithms, the first round of encryption corresponds to the last round of decryption; the second round of encryption corresponds to the next-to-the-last round of decryption, and so on. Based on this observation, CED computations can also be performed at the round level. At the beginning of each encryption round, the input data is stored in a register before being fed to the round module. After one round of encryption is finished, output is fed to the corresponding round of decryption. Then, the output of the decryption round is compared with the input data saved previously. If they are not the same, encryption is halted and an error signal is raised. Encryption with round-level CED is shown in Fig. 24b.

Depending on the block ciphers and their hardware implementation, each round may consume multiple clock cycles. Each round can be partitioned into operations and subpipelined to improve performance. Each operation can consume one or more clock cycles, such that the operations of encryption and corresponding operations of decryption satisfy the inverse relationship. As shown in Fig. 25a, applying input data to the encryption operation and the output data of the encryption operation to the corresponding inverse operation in decryption yields the original input data. The boundary on the left shows the rth encryption round while the boundary on the right shows the $(n - r + 1)$th decryption round, where r is the total number of rounds in encryption/decryption. Figure 25a also shows that the first operation of the encryption round corresponds to the mth operation of the decryption round, which is the last operation of the decryption round. Output from operation one of encryption is fed into the corresponding inverse operation m of decryption.

Although these techniques has close to 100 % fault coverage, their throughput is 73.45 % of the original AES in half-duplex mode. It can suffer from more than 100 % throughput overhead if the design is in full-duplex mode. The hardware overhead is minimal if both encryption and decryption are on the chip. However, if only encryption or decryption is used in the chip, it will incur close to 100 % hardware overhead.

To reduce the hardware overhead of the previous technique, [48] proposes a novel hardware optimization, and thus, reduced the hardware utilization significantly. Figure 25b shows the architecture. It divides a round function block into two sub-blocks and uses them alternatively for encryption (or decryption) and error detection. Therefore, no extra calculation block is needed, even though only a pipeline register, a selector and a comparator are added. The number of operating cycles is doubled, but the operating frequency is boosted because the round function

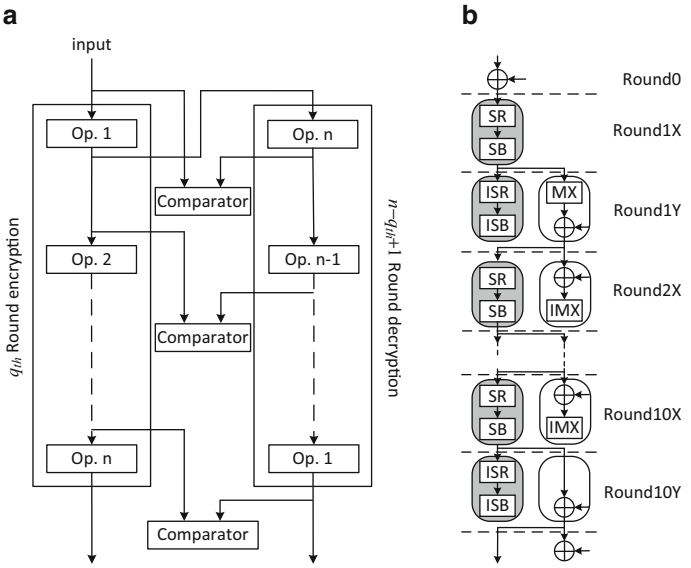


Fig. 25 Hybrid redundancy. (**a**) Operation level. (**b**) Optimized version

block in the critical path is halved. Therefore, the technique provides 85.6 % throughput compared to 73.45 % in the previous one. The hardware overhead also decrease from 97.6 to 88.9 %.

10.5 Other Techniques

In [24, 37, 39], parity is obtained for S-box implementation in finite field arithmetic with polynomial basis. In [25, 27, 38], parity is obtained for S-box implementation in finite field arithmetic with normal basis. In [26], an AES parity detection method with mixed basis is proposed. 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.

While traditional CED techniques have their strengths and limitations, techniques based on algorithmic invariances can offer new tradeoff choices. An invariance-based CED is proposed in [17]. It utilizes a round-level invariance of the AES and checks for the invariance property. Because the invariance does not constrain the input pattern of the round, it is very flexible and the fault coverage is very high. Because the invariance property is a permutation property, the hardware overhead only includes the comparator and muxes that are used to select the regular or the permuted datapath.

11 Conclusion

The chapter first shows that faults can be threatening to the security of modern day block ciphers. Taking the present day standard block cipher, AES, as an example the chapter shows techniques for performing differential fault analysis of block ciphers. The attacks have been described on the assumptions of various types of fault models: namely bit, single byte, and multiple byte. The fault analysis of AES using only one instance of a random single byte fault have been shown to reduce the key space of AES to on an average 28 values through an attack with time complexity 2^{30}. Likewise, the chapter also shows even when the faults spread to multiple bytes, corrupting 12 out of 16 bytes of the AES state matrix, DFA of AES is feasible. All these attacks unfurl that DFA of block ciphers is a very strong attack model, where even random faults can completely collapse the security of even very strong block cipher algorithms.

The practicality of such fault models and the requirement of such small number of faulty computations, emphasizes the need for CED techniques. We present a study on CED techniques deployed for detecting DFA of AES hardware implementation. We classify the CED techniques into hardware, time, information, and hybrid redundancies. We also summarize the overheads and fault coverage of the countermeasures. Hardware and hybrid redundancies provides high security and reliability but the hardware overhead is also high. Time redundancy is low cost but also decrease the performance. It also has limitations when there are permanent faults and long transient faults. Information redundancy provides high reliability with relatively low hardware and performance overhead. Although these countermeasures can thwart DFA, the designer needs to be cautious in implementation. Some techniques may enable backdoors for other types of attacks. For example, information redundancy-based techniques tend to increase the correlation of the circuit power consumption, with the processed data increasing the side channel leakage. While traditional CED techniques have their strengths and limitations, techniques based on algorithmic invariances can offer new tradeoff choices.

Acknowledgements The fourth author would like to acknowledge Indo-US S&T Forum for providing fellowship to support the above collaboration. The chapter was partially written during his visit as a researcher under the Indo-USSTF Fellowship to NYU-Poly, USA in 2012.

References

1. Agoyan, M., Dutertre, J.M., Mirbaha, A.P., Naccache, D., Ribotta, A.L., Tria, A.: How to flip a bit? In: IEEE International On-Line Testing Symposium (IOLTS), pp. 235–239 (2010)
2. Agoyan, M., Dutertre, J.M., Naccache, D., Robisson, B., Tria, A.: When clocks fail: on critical paths and clock faults. In: International Conference on Smart Card Research and Advanced Application (CARDIS), pp. 182–193 (2010)
3. Ali, S.S., Mukhopadhyay, D.: A differential fault analysis on AES key schedule using single fault. In: Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 35–42 (2011)

4. Ali, S.S., Mukhopadhyay, D.: An improved differential fault analysis on AES-256. In: International Conference on Cryptology in Africa (AFRICACRYPT), pp. 332–347 (2011)
5. Ali, S.S., Mukhopadhyay, D.: Differential fault analysis of AES-128 key schedule using a single multi-byte fault. In: International Conference on Smart Card Research and Advanced Applications (CARDIS), pp. 50–64 (2011)
6. Ali, S.S., Mukhopadhyay, D., Tunstall, M.: Differential fault analysis of AES using a single multiple-byte fault. Cryptology ePrint Archive, Report 2010/636 (2010). http://eprint.iacr.org/
7. Ali, S.S., Mukhopadhyay, D., Tunstall, M.: Differential fault analysis of AES: towards reaching its limits. J. Cryptogr. Eng. **3**(2), 73–97 (2013). doi:10.1007/s13389-012-0046-y. http://dx.doi.org/10.1007/s13389-012-0046-y
8. Barenghi, A., Hocquet, C., Bol, D., Standaert, F.X., Regazzoni, F., Koren, I.: Exploring the feasibility of low cost fault injection attacks on sub-threshold devices through an example of a 65nm AES implementation. In: Proceedings of Workshop RFID Security Privacy, pp. 48–60 (2011)
9. Bertoni, G., Breveglieri, L., Koren, I., Maistri, P., Piuri, V.: Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. IEEE Trans. Comput. **52**(4), 492–505 (2003)
10. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Proceedings of Eurocrypt. Lecture Notes in Computer Science, vol. 1233, pp. 37–51 (1997)
11. Blömer, J., Seifert, J.P.: Fault based cryptanalysis of the Advanced Encryption Standard (AES). In: Financial Cryptography, pp. 162–181 (2003)
12. Breveglieri, L., Koren, I., Maistri, P.: An operation-centered approach to fault detection in symmetric cryptography ciphers. IEEE Trans. Comput. **56**, 635–649 (2007)
13. Chen, C.N., Yen, S.M.: Differential fault analysis on AES key schedule and some coutnermeasures. In: Australasian Conference on Information Security and Privacy (ACISP), pp. 118–129 (2003)
14. Debdeep, M., Rajat Subhra, C.: Hardware Security: Designs, Threats, and Safeguards, CRC Press (2014)
15. Dusart, P., Letourneux, G., Vivolo, O.: Differential fault analysis on AES. In: Cryptology ePrint Archive, pp. 293–306 (2003)
16. Giraud, C.: DFA on AES. In: IACR e-print archive 2003/008, p. 008 (2003). http://eprint.iacr.org/2003/008
17. Guo, X., Karri, R.: Invariance-based concurrent error detection for advanced encryption standard. In: ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 573–578 (2012)
18. Joye, M., Manet, P., Rigaud, J.: Strengthening hardware AES implementations against fault attack. IET Inf. Secur. **1**, 106–110 (2007)
19. Karpovsky, M., Kulikowski, K.J., Taubin, A.: Differential fault analysis attack resistant architectures for the advanced encryption standard. In: World Computer Congress on Smart Card Research and Advanced Applications VI (CARDIS), pp. 177–192 (2004)
20. Karri, R., Wu, K., Mishra, P., Kim, Y.: Concurrent error detection of fault-based side-channel cryptanalysis of 128-bit symmetric block ciphers. In: Design Automation Conference (DAC), pp. 579–585 (2001)
21. Karri, R., Wu, K., Mishra, P., Kim, Y.: Fault-based side-channel cryptanalysis tolerant rijndael symmetric block cipher architecture. In: IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT) pp. 427–435 (2001)
22. Karri, R., Wu, K., Mishra, P., Kim, Y.: Concurrent error detection schemes of fault based side-channel cryptanalysis of symmetric block ciphers. IEEE Trans. Comput. Aided Des. **21**(12), 1509–1517 (2002)
23. Karri, R., Kuznetsov, G., Goessel, M.: Parity-based concurrent error detection of substitution-permutation network block ciphers. In: International Workshop on Cryptographic Hardware and Embedded Systems (CHES), pp. 113–124 (2003)
24. Kermani, M.M., Reyhani-Masoleh, A.: Parity prediction of S-box for AES. In: Canadian Conference on Electrical and Computer Engineering (CCECE), pp. 2357–2360 (2006)

25. Kermani, M.M., Reyhani-Masoleh, A.: A Low-cost S-box for the advanced encryption standard using normal basis. In: IEEE International Conference on Electro/Information Technology (EIT), pp. 52–55 (2009)
26. Kermani, M.M., Reyhani-Masoleh, A.: A high-performance fault diagnosis approach for the AES subbytes utilizing mixed bases. In: Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 80–87 (2011)
27. Kermani, M.M., Reyhani-Masoleh, A.: A low-power high-performance concurrent fault detection approach for the composite field S-Box and inverse S-Box. IEEE Trans. Comput. **60**(9), 1327–1340 (2011)
28. Khelil, F., Hamdi, M., Guilley, S., Danger, J.L., Selmane, N.: Fault analysis attack on an AES FPGA implementation. In: ESRGroups, pp. 1–5 (2008)
29. Kim, C.H.: Differential fault analysis against AES-192 and AES-256 with minimal faults. In: Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 3–9 (2010)
30. Kim, C.H.: Differential fault analysis of AES: toward reducing number of faults. Cryptology ePrint Archive, Report 2011/178 (2011). http://eprint.iacr.org/
31. Kim, C.H.: Improved differential fault analysis on AES key schedule. IEEE Trans. Inf. Forensics Secur. **7**(1), 41–50 (2012)
32. Kim, C.H., Quisquater, J.J.: New differential fault analysis on AES key schedule: two faults are enough. In: International Conference on Smart Card Research and Advanced Applications (CARDIS), pp. 48–60 (2008)
33. Li, W., Gu, D., Wang, Y., Li, J., Liu, Z.: An extension of differential fault analysis on AES. In: International Conference on Network and System Security (NSS), pp. 443–446 (2009)
34. Maingot, V., Leveugle, R.: Influence of error detecting or correcting codes on the sensitivity to DPA of an AES S-Box. In: International Conference on Signals, Circuits and Systems (ICSES), pp. 1–5 (2009)
35. Maistri, P., Leveugle, R.: Double-data-rate computation as a countermeasure against fault analysis. IEEE Trans. Comput. **57**(11), 1528–1539 (2008)
36. Moradi, A., Shalmani, M.T.M., Salmasizadeh, M.: A generalized method of differential fault attack against AES cryptosystem. In: International Workshop on Cryptographic Hardware and Embedded Systems (CHES), pp. 91–100 (2006)
37. Mozaffari-Kermani, M., Reyhani-Masoleh, A.: Parity-based fault detection architecture of s-box for advanced encryption standard. In: IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT), pp. 572–580 (2006)
38. Mozaffari-Kermani, M., Reyhani-Masoleh, A.: A lightweight concurrent error detection scheme for the AES S-boxes using normal basis. In Proceedings of Cryptographic Hardware and Embedded Systems (CHES), pp. 113–129 (2008)
39. Mozaffari-Kermani, M., Reyhani-Masoleh, A.: A lightweight high-performance fault detection scheme for the advanced encryption standard using composite field. IEEE Trans. VLSI Syst. **19**(1), 85–91 (2011)
40. Mukhopadhyay, D.: An improved fault based attack of the advanced encryption standard. In: Cryptology in Africa, AFRICACRYPT, pp. 421–434 (2009)
41. Natale, G.D., Flottes, M.L., Rouzeyre, B.: A novel parity bit scheme for SBox in AES circuits. In: IEEE Design and Diagnostics of Electronic Circuits and Systems (DDECS '07), pp. 1–5 (2007)
42. Natale, G.D., Flottes, M.L., Rouzeyre, B.: On-line self-test of AES hardware implementation. In: Workshop on DSN (2007)
43. National Institute of Stardards and Technology (NIST).: Advanced Encryption Standard (AES). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (2001)
44. Nyberg, K.: Differentially uniform mappings for cryptography. In: Advances in Cryptology - EUROCRYPT'93, pp. 55–64 (1993)
45. Piret, G., Quisquater, J.: A differential fault attack technique against SPN structures, with application to the AES and khazad. In: Proceedings of Cryptographic Hardware and Embedded Systems (CHES), pp. 77–88 (2003)

46. Rajendran, J., Borad, H., Mantravadi, S., Karri, R.: Sliced: slide-based concurrent error detection technique for symmetric block cipher. In: International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 70–75 (2010)
47. Saha, D., Mukhopadhyay, D., RoyChowdhury, D.: A diagonal fault attack on the advanced encryption standard. Cryptology ePrint Archive, Report 2009/581 (2009). http://eprint.iacr.org/
48. Satoh, A., Sugawara, T., Homma, N., Aoki, T.: High-performance concurrent error detection scheme for AES hardware. In: Cryptographic Hardware and Embedded Systems (CHES), pp. 100–112 (2008)
49. Selmane, N., Guilley, S., Danger, J.L.: Practical setup time violation attacks on AES. In: European Dependable Computing Conference, pp. 91–96 (2008)
50. Siewiorek, D.P., Swarz, R.S.: Reliable Computer Systems: Design and Evaluation, 3rd edn. A K Peters/CRC Press, A. K. Peters, Ltd. (1998)
51. Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: Proceedings of Cryptographic Hardware and Embedded Systems (CHES), pp. 2–12 (2002)
52. Takahashi, J., Fukunaga, T.: Differential fault analysis on AES with 192 and 256-bit keys. Cryptology ePrint Archive, Report 2010/023 (2010). http://eprint.iacr.org/
53. Takahashi, J., Fukunaga, T., Yamakoshi, K.: DFA mechanism on the AES key schedule. In: Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 62–74 (2007)
54. Tunstall, M., Mukhopadhyay, D., Ali, S.: Differential fault analysis of the advanced encryption standard using a single fault. In: Proceedings of the 5th IFIP WG 11.2 International Conference on Information Security Theory and Practice: Security and Privacy of Mobile Devices in Wireless Communication (WISTP), pp. 224–233 (2011)
55. Wu, K., Karri, R., Kuznetsov, G., Goessel, M.: Low cost concurrent error detection for the advanced encryption standard. In: International Test Conference (ITC), pp. 1242–1248 (2004)
56. Guo, X., Mukhopadhyay, D., Jin, C., Karri, R.: Security analysis of concurrent error detection against differential fault analysis. J. Cryptogr. Eng. (2014)