Table Redundancy Method for Protecting against Fault Attacks

Seungkwang Lee, Nam-su Jho, and Myungchul Kim

Information Security Research Division, ETRI
skwang@etri.re.kr

Abstract. Fault attacks (FA) intentionally inject some fault into the encryption process for analyzing a secret key based on faulty intermediate values or faulty ciphertexts. One of the easy ways for software-based countermeasures is to use time redundancy. However, existing methods can be broken by skipping comparison operations or by using non-uniform distributions of faulty intermediate values. In this paper, we propose a secure software-based redundancy, aptly named table redundancy, applying different linear and nonlinear transformations to redundant computations of table-based block cipher structures. To reduce the table size and the number of lookups, some outer tables that are not subjected to FA are shared, while the inner tables are protected by table redundancy. The basic idea is that different transformations protecting redundant computations are correctly decoded if the redundant outcomes are combined without faulty values. In addition, this recombination provides infective computations because a faulty byte is likely to propagate its error to adjacent bytes due to the use of 32-bit linear transformations. Our method also presents a stateful feature in the connection with detected faults and subsequent plaintexts for preventing iterative fault injection. We demonstrate the protection of AES-128 against FA and show a negligible advantage of FA.

Keywords: Software cryptography, fault attacks, countermeasure.

1 Introduction

The idea of inducing errors during the computation of a cryptographic algorithm to recover the key was first introduced by Boneh *et al.* [5,6] in 1997. They presented a successful attack on a CRT-RSA algorithm with both fault-free and faulty signatures of the same message. Such attacks are known as fault attacks. Since then, the fault attack was also applied to block ciphers by Biham and Shamir, and it was called Differential Fault Analysis (DFA) [1]. After AES was chosen to be the successor of DES, Giraud investigated two ways of DFA on AES by inducing faults in intermediate states or in the AES key schedule [17]. So far, DFA has been improved in such a way to require less brute-force search and faulty ciphertexts [3,13,19,28,34,36]. In addition, novel attack techniques that take advantage of faulty intermediate values have been proposed such as ineffective fault attacks (IFA) [9], statistical fault attacks (SFA) [14], and statistical ineffective fault attacks (SIFA) [12].

To protect the key from fault attacks, most of software countermeasures focus on detection and infection. Detection-based methods are mostly based on simple time redundancy with subsequent comparison. Infection-based methods, on the other hand, propagate the effect of faults in order to make faulty ciphertexts useless. Unfortunately, the existing methods of detection and infection are known to be vulnerable to attacks including instruction skips and SIFA. In this paper, we propose a new type of redundancy aptly named *table redundancy* to prevent FA on software implementations of block ciphers. By taking advantage of the internal encoding of white-box cryptography we apply different transformations to each redundant computation of a table-based AES implementation. Unless every redundant computation is fault-free, the proposed method leads to the following consequences with overwhelming probability. First, one or more faulty intermediate values have a propagation effect on the next lookup values which prevents the correct key from being recovered. Second, the proposed method is stateful, so it is likely to compute faulty ciphertexts for the subsequent encryption once some fault is detected. By doing so, it can avoid attempts to analyze a number of fault-free and faulty ciphertexts without any penalty.

Contribution. This study introduces *table redundancy*, a software countermeasure for protecting against fault attacks. It improves on a simple time redundancy method in such a way to withstand biased fault attacks. By adapting the internal encoding to table-based implementations of block ciphers, our proposed method can be easily applied to every block cipher. Table redundancy increases the likelihood of fault detection and error propagation because each redundant lookup table is generated by applying different encoding. Also, the previously detected faults are propagated to the next plaintexts thereby reducing the advantage of iterative fault injection. The encryption consists mostly of table lookups and does not require dedicated random sources to defend against fault attacks.

Outline. The rest of the paper is organized as follows. Section 2 reviews the internal encoding with the table structure of a white-box AES-128 implementation and explains previous FA and countermeasures. Section 3 presents our key idea and proposes a secure AES-128 implementation with table redundancy. We then analyze its security and performance in Section 4. Section 5 concludes this paper.

2 Preliminaries

In order to obfuscate the intermediate values of block ciphers, white-box cryptography applies the external and internal encodings to table-based implementations. In particular, the linear transformation provides a diffusion effect on the encoding of intermediate blocks. In addition, the nonlinear transformation realizes information confusion and conceals the value of 0. To implement our table redundancy method for a block cipher, we will adapt the internal encoding of white-box cryptography. However, it does not mean that our proposed method is resistant to white-box attacks or every gray-box attack; this study is restricted to the fault attacks on the gray-box model in symmetric key cryptography, where an attacker has no visibility and control into memory. The internal encoding and the table diversity will contribute to providing detection and infection features. In this section, we review an internally encoded implementation of AES-128 from white-box cryptography [8]. Afterwards, we briefly explain previous FA and countermeasures.

2.1 Internal Encoding on AES-128

White-box cryptography of block ciphers is mostly implemented in a table-based manner with linear and nonlinear transformations (the term *encoding* is often used) in order to hide key-dependent intermediate values. Given an n-bit key, the table size is certainly problematic when mapping all n-bit plaintexts to n-bit ciphertexts by using a single lookup table. For example, if $n = 128$ like in the case of AES-128, the entire lookup table requires $2^{128} \cdot 128$ bits. To solve this problem, a set of lookup tables is generated for each step and each round. The table lookups are then properly ordered in a networked manner.

Given a lookup table $\mathcal{T}$, let's choose two secret encodings f and g in order to obfuscate inputs and outputs, respectively. A new table $\mathcal{T}'$ can be generated by

$$\mathcal{T}' = g \circ \mathcal{T} \circ f^{-1}.$$

To get $\mathcal{T}(x)$, the input to $\mathcal{T}'$ will be $f(x)$, and $\mathcal{T}'(f(x))$ will be decoded by g^{-1} in the next lookup table, say $\mathcal{R}$. To feed the $\mathcal{T}$ output into $\mathcal{R}$, the encoding and decoding should be connected to each other at the boundary of the tables. For example,

$$\mathcal{T}' = g \circ \mathcal{T} \circ f^{-1} \text{ and } \mathcal{R}' = h \circ \mathcal{R} \circ g^{-1},$$

then we have

$$\mathcal{R}' \circ \mathcal{T}' = (h \circ \mathcal{R} \circ g^{-1}) \circ (g \circ \mathcal{T} \circ f^{-1}).$$

To reduce the number of lookups, the initial white-box AES (WB-AES) implementation [8] turns AddRoundKey, SubBytes, and part of MixColumns into a composition by re-writing AES as follows:

state ← *plaintext*
for $r = 1 \cdots 9$
    ShiftRows(state)
    AddRoundKey(state, $\hat{k}^{r-1}$)
    SubBytes(state)
    MixColumns(state)
ShiftRows(state)
AddRoundKey (state, $\hat{k}^9$)

SubBytes(state)
AddRoundKey(state, k^{10})
ciphertext $\leftarrow$ state,

where k^r is a 4×4 round key matrix at round r, and $\hat{k}^r$ is the result of applying ShiftRows to k^r. AddRoundKey and SubBytes are first combined into *T-boxes*, a series of 160 (one per cell per round) $8{\times}8$ lookup tables as follows:

$$\begin{aligned} T^r_{i,j}(x) &= S(x \oplus \hat{k}^{r-1}_{i,j}), \qquad \text{for } i,j \in [0,3] \text{ and } r \in [1,9], \\ T^{10}_{i,j}(x) &= S(x \oplus \hat{k}^{9}_{i,j}) \oplus k^{10}_{i,j} \text{ for } i,j \in [0,3]. \end{aligned}$$

In round 1 to 9, each *T-box* output is multiplied with each column of the MixColumns matrix MC to reduce the table size. Let $[x_0\ x_1\ x_2\ x_3]^T$ be a column vector of the outcome state after mapping the round input to *T-boxes*. By the linearity of a matrix multiplication, MixColumns can be decomposed as follows:

$$\begin{aligned} &\begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \\ &= x_0 \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus x_1 \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus x_2 \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus x_3 \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \\ &= x_0 \cdot MC_0 \oplus x_1 \cdot MC_1 \oplus x_2 \cdot MC_2 \oplus x_3 \cdot MC_3. \end{aligned}$$

For the right-hand side (say y_0, y_1, y_2, y_3), the commonly named Ty_i tables mapping 8-bits to 32-bits are defined as follows:

$$\begin{aligned} Ty_0(x) &= x \cdot [02\ 01\ 01\ 03]^T \\ Ty_1(x) &= x \cdot [03\ 02\ 01\ 01]^T \\ Ty_2(x) &= x \cdot [01\ 03\ 02\ 01]^T \\ Ty_3(x) &= x \cdot [01\ 01\ 03\ 02]^T. \end{aligned}$$

To put it simply, WB-AES is a series of table lookups, consisting of encoded inputs and outputs of Ty_i tables. Precisely, the input is protected by $8{\times}8$ linear transformations while the output is protected by $32{\times}32$ linear transformations. The nonlinear transformation on each byte is then divided into two four-bit concatenated forms to avoid huge XOR lookup tables. In the following explanation, it is assumed for convenience that nonlinear transformations are applied to the input/output values of all tables.

Our proposed implementation of AES-128 will adapt the four types of lookup tables used in WB-AES that are internally encoded [8]. First, *TypeII* is a composition of *T-boxes* and Ty_i. This is an 8-bit to 32-bit lookup table, and each 32-bit output is protected by a $32{\times}32$ linear transformation, say $\mathcal{L}$. The MixColumns multiplication computed by the *TypeII* outputs is followed by the XOR operations to compute the encoded round output. This is conducted by *TypeIV* that takes two four-bit encoded inputs and provides a four-bit encoded XOR result. However, each single byte of a 32-bit round output protected by $\mathcal{L}$ cannot be solely decoded without the other three bytes. For this reason, the linear transformation $\mathcal{L}$ will be replaced with four 8×8 linear transformations by looking up *TypeIII*. Let $\hat{\mathcal{L}}$ denote their concatenated transformation. Given a 32-bit vector $[v_0\ v_1\ v_2\ v_3]^T$ protected by $\mathcal{L}$, looking up *TypeIII* and *TypeIV* performs

$$\bar{\mathcal{L}}\begin{bmatrix} v_0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \oplus \bar{\mathcal{L}}\begin{bmatrix} 0 \\ v_1 \\ 0 \\ 0 \end{bmatrix} \oplus \bar{\mathcal{L}}\begin{bmatrix} 0 \\ 0 \\ v_2 \\ 0 \end{bmatrix} \oplus \bar{\mathcal{L}}\begin{bmatrix} 0 \\ 0 \\ 0 \\ v_3 \end{bmatrix}$$

where $\bar{\mathcal{L}} = \hat{\mathcal{L}} \circ \mathcal{L}^{-1}$. By doing so, a single-byte input to *TypeII* in the next round can be simply decoded by $\hat{\mathcal{L}}^{-1}$. Lastly, *TypeV* is the lookup table of T^{10} in the final round. Since no MixColumns is involved in the final round, each 8-bit to 8-bit mapping by *TypeV* gives the corresponding subbyte of the ciphertext. Because the external encoding is not used in the proposed method, its output is not encoded.

There are two security metrics: the white-box diversity and ambiguity. The white-box diversity is a measure of variability, counting distinct constructions for a particular table type. The white-box ambiguity of a table, on the other hand, is a measure of the number of alternative interpretations and counts the number of distinct constructions producing the same table of that type. In our proposed method, we take advantage of the diversity which can produce abundant lookup tables using the countless transformations.

2.2 DFA based on a single-byte fault

The basic idea of DFA is as follows: (1) running the target cryptographic algorithm and obtaining a fault-free ciphertext. (2) injecting faults during the execution of the target algorithm with the same plaintext and obtaining faulty ciphertexts. (3) analyzing the relationship between the fault-free and faulty ciphertexts to reduce the search space of the key. The analysis depends on the fault model with respect to the fault location and characteristic as follows.

First, injecting a single-byte between the 8-th and 9-th round MixColumns affects four bytes of the ciphertext because the final round does not involve MixColumns. Among many working principles of DFA based on this fault propagation, we briefly review a technique using the four 9-th round differential equations [34].

Suppose that the first subbyte denoted by x of the 9-th round input is changed to a faulty intermediate value denoted by $x \oplus \delta$, where $x, \delta \in \mathrm{GF}(2^8)$. Then δ is changed to δ' after SubBytes, and the four-byte difference in the 9-th round output is represented by $(2\delta', \delta', \delta', 3\delta')$, where the coefficients are the elements of MC_0. ShiftRows will move the difference to four different locations as shown in Fig. 1. With fault-free and faulty ciphertexts for the same plaintext, DFA can express the four-byte difference with respect to the key K. Let S^{-1}

denote the inverse SubBytes, $C = C_1C_2\ldots C_{16}$ the fault-free ciphertext, and $\widetilde{C} = \widetilde{C}_1\widetilde{C}_2\ldots\widetilde{C}_{16}$ the faulty ciphertext. For example, $\widetilde{C}_1 = C_1 \oplus \Delta_1$. Then the following equations take the fault-free and faulty ciphertexts as well as each subkey candidate $K_i^* \in \mathrm{GF}(2^8)$.

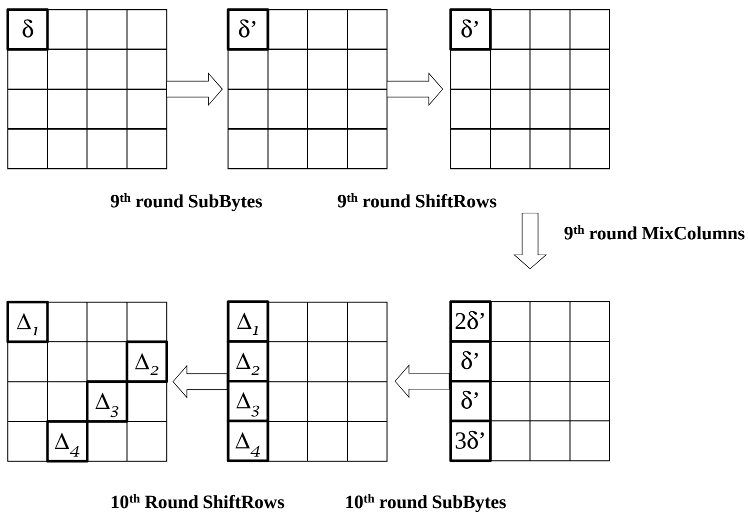


Fig. 1: Fault propagation across the last two rounds of AES.

$$\begin{aligned}
2\delta' &= S^{-1}(C_1 \oplus K_1^*) \oplus S^{-1}(\widetilde{C}_1 \oplus K_1^*)\\
\delta' &= S^{-1}(C_8 \oplus K_8^*) \oplus S^{-1}(\widetilde{C}_8 \oplus K_8^*)\\
\delta' &= S^{-1}(C_{11} \oplus K_{11}^*) \oplus S^{-1}(\widetilde{C}_{11} \oplus K_{11}^*)\\
3\delta' &= S^{-1}(C_{14} \oplus K_{14}^*) \oplus S^{-1}(\widetilde{C}_{14} \oplus K_{14}^*).
\end{aligned} \tag{1}$$

These equations are called the 9-th round differential equations [34] which will reduce the search space of key quartet to an expected value of 2^8. This means that only 2^8 candidates of the key quartet will satisfy the differential equations. By injecting two such faults the key quartet can be uniquely determined and the remaining three quartets can be similarly analyzed. In Section 4, getting the 9-th round differential equations by injecting a single-byte fault into a non-protected WB-AES implementation will be demonstrated.

Second, injecting a single-byte fault between the 7-th and the 8-th round MixColumns gives additional information called the 8-th round differential equations. By using both 8-th and 9-th round differential equations, a single faulty ciphertext can further reduce the search space of the key from 2^{32} to 2^8 with

2^{32} time complexity, as each of 2^{32} candidates of the final round key is tested by set of four equations. This attack cost can be reduced to 2^{30} by an acceleration technique [34].

2.3 DFA based on a multi-byte fault

Authors in [27] presented two different multi-byte fault attacks covering all possible faults on the MixColumns input in the 9-th round. The first attack requires at least one fault-free byte in one column of MixColumns input, and 6 faulty ciphertexts discover the key in average. In the second attack, where all four bytes of the column are supposed to be faulty, approximately 1,500 faulty ciphertexts can recover the key.
In [33], a diagonal fault model was proposed, where the state matrix is divided into four diagonals. If faults are injected into one, two, or three diagonals, the key search space is reduced to 2^{32}, 2^{64}, or 2^{96}, respectively. In the case of injecting faults into four diagonals, the search space becomes larger than brute force.

2.4 FA based on faulty intermediate values

Impossible DFA (IDFA) [11,31] on block ciphers looks for probability zero differentials between fault-free and faulty intermediate values to remove the wrong key candidates from the list. A biased fault model is known to be effective to induce exactly the same faults in both computations of the time redundancy countermeasures [30]. Differential Fault Intensity Analysis (DFIA) [15] combines fault injection under different intensity with the principles of Differential Power Analysis [20]. By using biased fault models as the leakage source, an attacker finds a correct key producing the minimum of cumulative Hamming Distance among all key candidates.
IFA [9], as a type of Safe Error Analysis [35], exploits ineffective faults that result in no computation error. If there is no change in the ciphertext after injecting the fault, the internal state of the attacked bit or byte can be determined with a high probability. This approach is likely to bypass time redundancy, as only one computation needs to be faulted. In practice, however, most attackers are not powerful enough to inject precise faults for a great number of encryption. In the case of infective countermeasures, false positives should also be considered because an attacker does not know whether the attacked byte belongs to a dummy round. SFA [14], on the other hand, works on faulty ciphertexts under three types of fault models: stuck-at-0; stuck-at-0 with a probability of 0.5 or logical AND with random uniform value with a probability of 0.5; logical AND with random uniform value. For each subkey candidate, every ciphertext is decrypted back to the attacked point and the key is guessed by the highest squared euclidean imbalance (SEI) of the faulted byte. However, this attack is less likely to succeed with increasing redundancy of the countermeasure. SIFA [12] is an extension from IFA and SFA that exploits both ineffective faults and non-uniformly distributed intermediate values. For a wide range of faults such as stuck-at, random, and biased faults that can happen in practice, fault distribution tables

can be computed, where the diagonal gives a non-uniform distribution of the ineffective fault for each value. This attack exploits ciphertexts in which the attacked variable follows the non-uniform distribution determined by the diagonal and recovers the subkey candidate by SEI. This approach is known to be effective to detection- and infection-based countermeasures. Persistent Fault Attack (PFA) [38] injects one fault on an element in the SBox table. Based on biased distribution on ciphertexts resulting from this faulty SBox, an attacker statistically recovers the key.

Note that in the above attacks, the target implementation is considered stateless which means that the previous detection of faults does not have an influence on the next execution of encryption. Therefore, an attacker can recover the entire secret key by repeating the fault injection. In order to hinder iterative collection of such information, the proposed method will attempt to prevent a number of fault injections through a stateful implementation of the block cipher.

2.5 Countermeasures

Detection-based countermeasures, also known as Concurrent Error Detection (CED) [21], use additional redundancy to detect FA. There are four types of redundancy as follows. (1) Information redundancy is based on error detecting codes such as parity bit and robust code. Recently, many hardware implementations (including Toffoli gates) of error correcting codes that protect against SIFA have been proposed [7, 10, 18]. Here we note that this study focuses on software techniques. (2) Time redundancy is a classical fault tolerance technique in which a cryptographic operation is computed more than once with the same input. If there is a mismatch of the results, a random ciphertext or an error code is returned. Assuming that the injected fault is uniformly distributed, an attacker must inject exactly the same faults in both computations. However, a biased fault can defeat a time redundancy countermeasure because of relatively high fault collision probability [30]. (3) In hardware redundancy techniques, the same inputs are fed into both original and duplicated circuits, and the outputs are compared to each other. (4) A hybrid redundancy combines the characteristics of the previous techniques. For example, a fault can be detected by comparison of an original plaintext with a decrypted plaintext. In this case, both encryption and decryption hardware are used on a single chip.

Infection-based countermeasures, on the other hand, use the diffusion effects of faults instead of comparative computations in order to make a faulty ciphertext unexploitable. Specifically, Tupsamudre *et al.* [37] proposed to use intermediate dummy rounds to overcome the weaknesses of deterministic diffusion based infective methods [25] and a random variation [16]. Patranabis *et al.* [29] modified it in such a way to randomize the order of the redundant and cipher rounds along with masking the previous round outputs in the consideration of instruction skip attacks.

3 Proposed Method

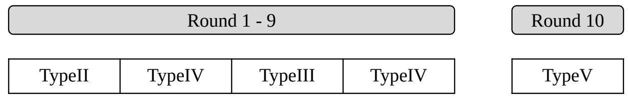
In this section, we present a secure AES-128 implementation protected by our proposed method, aptly named *table redundancy*, for preventing non-invasive FA attacks. To this end, the internal encoding will be utilized for the following properties. 1) Table redundancy will take advantage of redundant computations of time redundancy, but each computation will involve a different set of lookup tables generated by different encoding. Since an intermediate value is encoded into different values for each redundant computation, injecting the same faulty value into all computations will not guarantee a successful attack without detection. Note that simple time redundancy can be attacked by inserting the same faulty value to each redundant computation. 2) If a fault is injected into the intermediate value protected by a 32-bit linear transformation, it also has the effect of spreading the error during the decoding of other nearby values. 3) Instead of simple comparison operations, the integrity will be verified through an infective recombination logic composed of table lookups, so skipping several operations cannot bypass the detection. 4) Unless every computation is fault-free, the correct decoding of plaintexts for subsequent encryption is unlikely to be guaranteed. This represents the stateful nature of our method, which hinders iterative fault injection into software implementations.

3.1 Basic idea

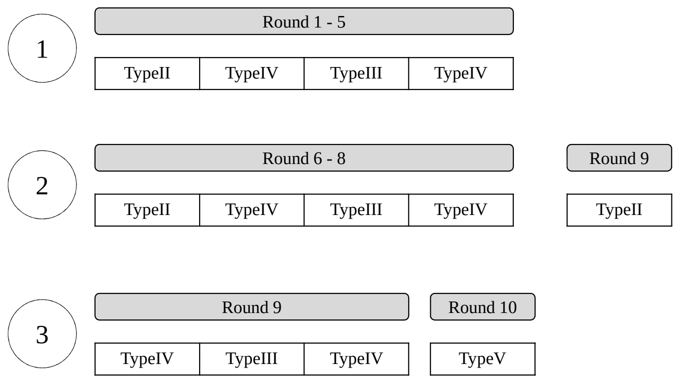
Table redundancy. Before going into depth, we note that a single-bit fault attack on the initial AddRoundKey or the final round is not considered because there is no guarantee that the one-bit difference in the input leads to a consistent difference in the output due to the use of encoding. Based on this fact, redundancy is not applied for the first few rounds of tables that will not be subject to FA attacks to reduce the total table size. In other words, we perform the redundant computations which are subjected to FA; the other parts of the computation are shared. To the best of our knowledge, the earliest location of DFA on AES is AddRoundKey at the 4-th last round in IDFA [11]. Because AddRoundKey in the 6-th round of AES-128 was shifted into the next round in our structure, the original sequence of table lookups of WB-AES illustrated in Fig. 2a is conservatively divided into three parts as depicted in Fig. 2b.

1. From Round 1 to 5
2. From Round 6 to *TypeII* in Round 9
3. From *TypeIV* in Round 9 to Round 10.

In Part 1, the first 5 rounds are not under the attack in this paper and therefore are shared without redundancy. In Part 2, we perform redundant computations with different sets of lookup tables. The redundant outputs of Part 2 will be the SubBytes outputs multiplied by each column vector of *MC* protected by different encoding. Before computing the operations in Part 3, the redundant outputs should be recombined to check if there is no faulty byte; otherwise a



(a) Sequence of table lookups in WB-AES



(b) Our partitions in the proposed method

Fig. 2: Comparison of table partition.

fault spreads to the adjacent four bytes. This summarizes the redundancy and infective properties of the proposed method.

For the lookup tables generated with the key $\mathcal{K}$, let $\mathcal{T}_b$ (b stands for "begin") denote a set of shared lookup tables of Part 1. Given a plaintext $\mathcal{P}$, Part 1 is followed by Part 2 consisting of two different sets of lookup tables, $\mathcal{T}_0$ and $\mathcal{T}_1$, which are generated by using different sets of transformations.

By the table diversity, each redundant computation will produce different intermediate values, but their decoded values must be the same. Here, we call the computation and recomputation using $\mathcal{T}_0$ and $\mathcal{T}_1$ original and redundant, respectively. The lookup values from $\mathcal{T}_0$ and $\mathcal{T}_1$ are then the encoded SubBytes output multiplied by a column vector of MC in the 9-th round. We denote by $\mathcal{Q}_0$ and $\mathcal{Q}_1$ these output states of $\mathcal{T}_0$ and $\mathcal{T}_1$, respectively. In general, $\mathcal{Q}_0$ and $\mathcal{Q}_1$ will be provided in a $4{\times}4{\times}4$ array because *TypeII* maps an 8-bit input to a 32-bit output.

Let $\mathcal{T}_x$ denote a set of *TypeIV* tables regardless of the number of copies. After recombining the original and redundant outputs through an additional $\mathcal{T}_x$, the rest of computation in Part 3 is performed by $\mathcal{T}_e$ (e stands for "end"). Sharing Part 3 also reduces the total table size and the number of lookups. Table 1 explains the encoding notations, and Fig. 3 briefly describes our table redundancy with a single redundant computation. Note that the *TypeIV* tables involve only nonlinear transformations on the input and output due to the distributive property of multiplication over addition.

Table 1: Notations for the encoding. The subscript $_*$ will be either a number or a letter.

Notation	Description
$\mathcal{L}_*$	Linear transformation
$\mathcal{N}_*$	Nonlinear transformation
$\mathcal{E}_*$	$\mathcal{N}_* \circ \mathcal{L}_*$

XOR instead of comparison. In Fig. 3, $\mathcal{Q}_x$ is a result of infective XOR operations between $\mathcal{Q}_0$ and $\mathcal{Q}_1$. This step plays an important role of detection and infection at the same time because two fault-free states guarantee the correct computation of Part 3 which would otherwise propagate errors violating the differential equations. Since each 32-bit quartet in $\mathcal{Q}_0$ and $\mathcal{Q}_1$ is protected by a 32×32 linear transformation, a single-byte manipulation has an infectious effect on the other three bytes.

Now we explain how to pick the 32×32 binary matrices used in $\mathcal{T}_0$, $\mathcal{T}_1$ and $\mathcal{T}_e$, which are denoted by $\mathcal{L}_0$, $\mathcal{L}_1$ and $\mathcal{L}_e$, respectively. Here we recall that

$$\mathcal{Q}_i = \mathcal{E}_i(Y_j) = \mathcal{N}_i \circ \mathcal{L}_i(Y_j),$$

where $i \in \{0, 1\}$ and $Y_j = Ty_{j\in\{0,1,2,3\}}(\cdot)$. Then it is easy to know that $\mathcal{T}_x$ gives us $\mathcal{Q}_x$:

$$z = \mathcal{L}_0 \cdot Y_j \oplus \mathcal{L}_1 \cdot Y_j = (\mathcal{L}_0 \oplus \mathcal{L}_1) \cdot Y_j$$

$$\mathcal{Q}_x = \mathcal{N}_x(z).$$

In the beginning of $\mathcal{T}_e$, *TypeIV* combines the *TypeII* output in the 9-th round (given by $\mathcal{Q}_x$). Next, the *TypeIII* and the following *TypeIV* replace the linear transformation $(\mathcal{L}_0 \oplus \mathcal{L}_1)$ with four 8×8 linear transformations. For

$$\mathcal{L}_0 \oplus \mathcal{L}_1 = (\mathcal{L}_e)^{-1},$$

$\mathcal{L}_e$ must be invertible while $\mathcal{L}_0$ and $\mathcal{L}_1$ do not necessarily have to be invertible. So we pick those matrices as follows:

- Generate a 32×32 invertible binary matrix $\mathcal{L}_e$.
- Generate a random 32×32 binary matrix $\mathcal{L}_0$.
- Compute $\mathcal{L}_1 = (\mathcal{L}_e)^{-1} \oplus \mathcal{L}_0$.

The last step for computing a ciphertext $\mathcal{C}$ is to lookup *TypeV*.

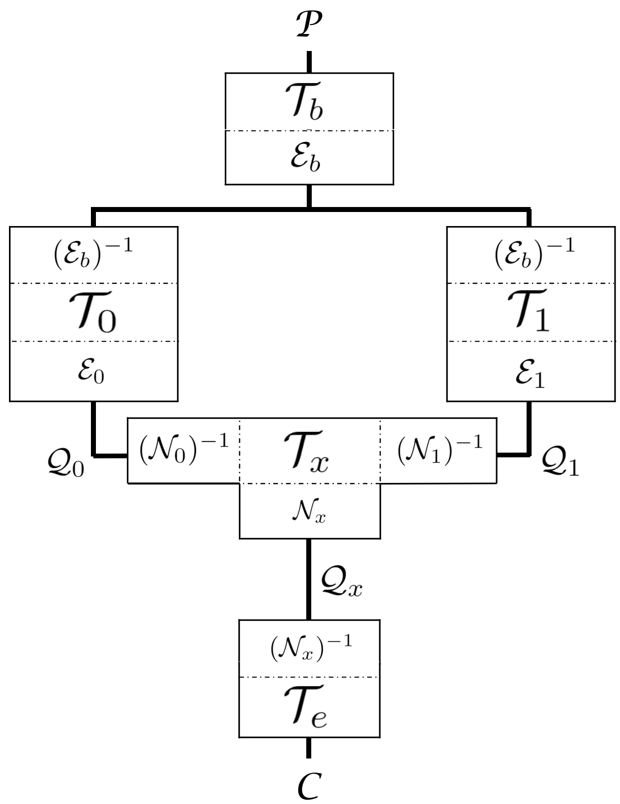


Fig. 3: Simple description of our key idea with a redundant computation.

3.2 Enhancing security with additional redundancy

Suppose that an attacker injects two single-byte faults on the 8-th round inputs in $\mathcal{T}_0$ and $\mathcal{T}_1$, respectively, and tries to make a fault collision in which two disturbed bytes will be decoded to the same value. The probability of getting valid differential equations by this event is then 2^{-8}. To further reduce this probability, we increase the number of redundant computations by $n > 1$ with additional tables generated using different transformations. If $n = 3$, we have three redundant computations as illustrated in Fig. 4. Here, $\mathcal{L}_n$ is obtained from $\mathcal{L}_e$ and n random binary matrices $\mathcal{L}_{i\in[0,n-1]}$ as follows:

$$\mathcal{L}_n = (\mathcal{L}_e)^{-1} \oplus \bigoplus_{i=0}^{n-1} \mathcal{L}_i.$$

In addition, we need more $\mathcal{T}_x$ tables for the XOR operation of redundant computations. These are aptly named $\mathcal{T}_{x0}$, $\mathcal{T}_{x1}$ and $\mathcal{T}_{x2}$.

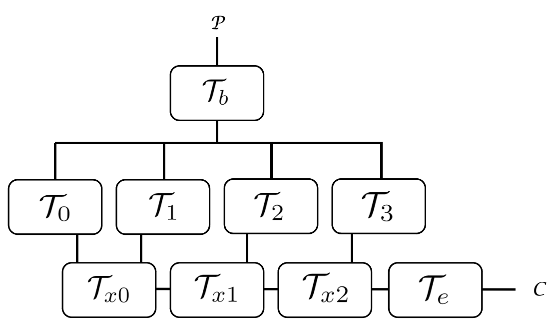


Fig. 4: Extension with three redundant computations.

3.3 From stateless to stateful encryption

The execution of FA actually needs an attacker owning the victim's device. In this case, it is advantageous for the device to strategically avoid iterative attack attempts to protect the secret key, reducing the repeated leakage of information. In addition, it is recommended to update the secret key even if the user takes back the ownership of the device. So it is not necessary to perform correct encryption operations until the secret key is updated after detecting faults. From this practical point of view, we add a stateful feature to our proposed implementation. The following explains it with a single redundant computation depicted in Fig. 3.

In order to perform the initial encryption, the state of nonlinearly transformed zeros $\mathcal{Z}$ is calculated using $\mathcal{N}_z$ and is then stored. In Fig. 5, a black square

indicates $\mathcal{Z}$. In the first step of encryption, the states of the plaintext $\mathcal{P}$ and $\mathcal{Z}$ are XORed via $\mathcal{T}_x$. As shown in Fig. 5, the first $\mathcal{T}_x$ here does not decode $\mathcal{P}$, but only $\mathcal{Z}$ by $(\mathcal{N}_z)^{-1}$, and its XOR outcomes are nonlinearly transformed by $\mathcal{N}_p$. This will give the input state to $\mathcal{T}_b$ of Part 1 which is now generated with the input decoding by $(\mathcal{N}_p)^{-1}$.
In addition to the original and redundant computations, another redundancy, called a clean-up computation, is performed by $\mathcal{T}_d$ (d stands for "detect") to interfere with next encryption in case of fault detection. What is important over here is that the linear transformation $\mathcal{L}_d$ protecting the output of $\mathcal{T}_d$ is set to $(\mathcal{L}_e)^{-1} = \mathcal{L}_0 \oplus \mathcal{L}_1$. By doing so, $\mathcal{T}_x$ between $\mathcal{Q}_x$ and $\mathcal{Q}_d$ produces only zeros which are nonlinearly transformed if there is no error. The XOR operations via $\mathcal{T}_x$ at here require additional XOR operations to squeeze the nonlinearly transformed zeros filled in a 4×4×4 array into a 4×4 state $\mathcal{Z}$.
This clean-up computation can contribute to reducing the probability of successful FA. In particular, collecting correct ciphertexts (including intermediate values) and observing ineffective faults are hindered. If $\mathcal{Z}$ turns out to be disturbed, it is possible to call an additional routine for requesting the key update (the entire table). However, we do not deal with key updates in this study.

4 Evaluation

The security evaluation in this section will analyze a success probability and complexity of FA on our method with n redundant computations. For simplicity, we use the n-th redundant one as the clean-up computation ($\mathcal{T}_n = \mathcal{T}_d$). To put it simply, there are n - 1 redundant computations with a clean-up computation. For a successful attack, the original and redundant outcomes should be decoded as the same intermediate state, and there should be no faulty byte in $\mathcal{Z}$ resulted from the recombination and the clean-up computation. The performance will be evaluated in terms of the table size and the number of lookups.

4.1 Protection of DFA

Consider a single-byte fault injection on the first subbyte of each 9-th (or 8-th) round input in $\mathcal{T}_0$ to $\mathcal{T}_n$. The fault collision for obtaining valid faulty ciphertexts with the correct clean-up computation can be occurred if each of $n+1$ disturbed bytes is decoded to the same *T-box* input, say $x^f \in \mathrm{GF}(2^8)$. The probability of this event is $(2^{-8})^n$, which is negligible as n increases. It is approximately 5×10^{-8} if $n = 3$.
Suppose that a fault collision is not occurred in $\mathcal{T}_{i \neq n}$, where $\mathcal{L}_i$ is a singular linear transformation. Then there can exist $x' \in \mathrm{GF}(2^8)$ such that

$$x' \neq x^f \text{ but } \mathcal{L}_i(Ty_0(x')) = \mathcal{L}_i(Ty_0(x^f))$$

due to the property of singular linear transformations. We call it a transformation collision. The number of nonsingular $m \times m$ binary matrices denoted

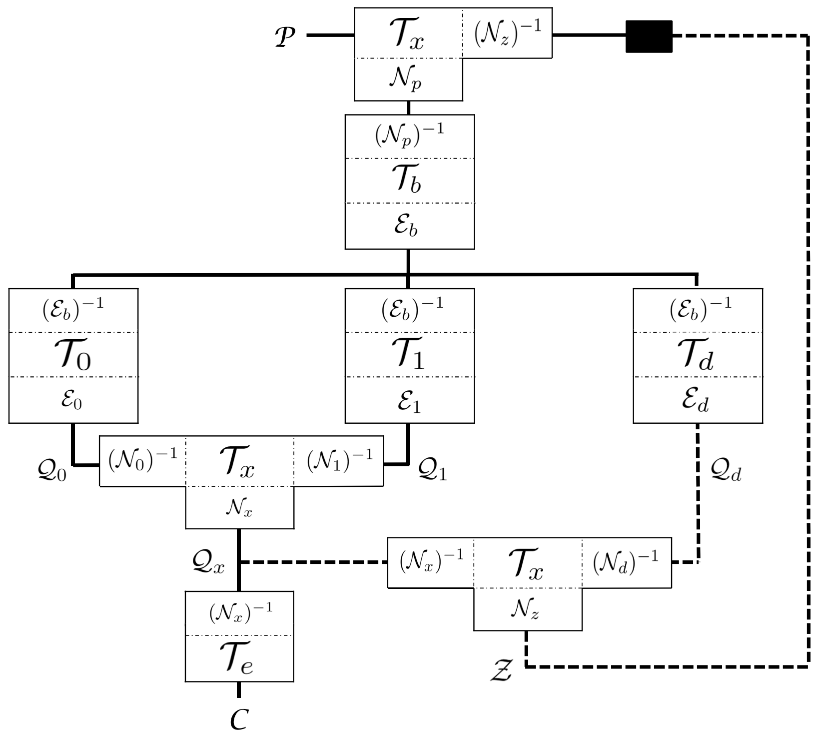


Fig. 5: A stateful version of the proposed method with a redundant computation. The black square is $\mathcal{Z}$ connected to the clean-up computation (dotted line)

by $\#GL_m(\mathbb{F}_2)$ is negligible compared to the number of singular $m \times m$ binary matrices denoted by $\#Sg_m(\mathbb{F}_2)$ if $m = 32$ like in the case of $\mathcal{L}_*$, where

$$\#GL_m(\mathbb{F}_2) = \prod_{k=0}^{m-1} (2^m - 2^k),$$

and

$$\#Sg_m(\mathbb{F}_2) = 2^{m^2} - \#GL_m(\mathbb{F}_2).$$

Therefore, if $\mathcal{L}_{i\in[0,n)}$ is randomly generated, it is more likely to be singular than the probability of nonsingular. For 10,000 singular matrices which are randomly generated, an average of 1.47 inputs (among 256 elements) to the *T-box* caused transformation collisions for each matrix. This is less than 2/256. Then, the probability of $k \in [1, n]$ fault collisions and $n - k$ transformation collisions is negligible which can be upper bounded by

$$\sum_{k=1}^{n} \binom{n}{k} (1/256)^k \cdot [2/256 \cdot \#Sg_{32}/(2^{32})^2]^{n-k}.$$

Next, consider a multi-byte fault which is injected randomly by a non-invasive way to a quartet (four-byte intermediate value) in $\mathcal{Q}_x$ and $\mathcal{Q}_d$. Then each faulted quartet denoted by q_x and q_d in $\mathcal{Q}_x$ and $\mathcal{Q}_d$, respectively, is valid if

$$^{\exists}x' \in GF(2^8) \text{ such that } (\mathcal{N}_x)^{-1} \circ (q_x) = (\mathcal{L}_e)^{-1} \circ (Ty_0(x'))$$

and

$$(\mathcal{N}_x)^{-1} \circ (q_x) = (\mathcal{N}_d)^{-1} \circ (q_d).$$

Because the faults are assumed to be induced randomly, these events happen with a negligible probability of $(2^{-8})^{4\cdot 2}$ due to the fixed elements of MC.

By injecting a single-byte fault at the first subbyte of the 9-round inputs, we simply demonstrate that the 9-th differential equations work on the unprotected WB-AES implementation (with a 128-bit key), but does not work on our proposed method. Within the algorithm, we introduced code for injecting random faults in the right location, resulting in the four faulty bytes with a particular pattern illustrated in Fig. 1. Let the plaintext and key have the same value:

$$0x000102030405060708090A0B0C0D0E0F.$$

This computes the final round key and the fault-free ciphertext as represented in Fig. 6a and Fig. 6b, respectively. By changing the first subbyte of the 9-th round input of the unprotected WB-AES, we obtained a faulty ciphertext as shown in Fig. 6c. Plugging the subkeys, the fault-free and faulty bytes shaded in Fig 6a - Fig. 6c into the 9-th round differential equations, we get

$$\begin{aligned}
2\delta' &= S^{-1}(0x0\text{A} \oplus 0x13) \oplus S^{-1}(0x34 \oplus 0x13)\\
\delta' &= S^{-1}(0x53 \oplus 0x2\text{B}) \oplus S^{-1}(0x72 \oplus 0x2\text{B})\\
\delta' &= S^{-1}(0x94 \oplus 0x\text{A}7) \oplus S^{-1}(0x90 \oplus 0x\text{A}7)\\
3\delta' &= S^{-1}(0x45 \oplus 0x17) \oplus S^{-1}(0x02 \oplus 0x17),
\end{aligned} \tag{2}$$

where $\delta' = 0x\text{D}4$ ($2\delta' = 0x\text{B}3$, $3\delta' = 0x67$). This shows that DFA can extract the key from WB-AES as the coefficients of δ' exactly follow the 9-th round differential equations.

Next, let us demonstrate the protection of DFA in our protected AES with a redundant computation. With a single-byte fault at each of the first subbyte of the 9-th round inputs in original and redundant computations, we obtained a faulty ciphertext as shown in Fig. 6d. Plugging the faulty bytes into the 9-th round differential equations gives us

$$\begin{aligned}
0x\text{BF} &= S^{-1}(0x0\text{A} \oplus 0x13) \oplus S^{-1}(0x\text{D}4 \oplus 0x13)\\
0x\text{F}9 &= S^{-1}(0x53 \oplus 0x2\text{B}) \oplus S^{-1}(0x2\text{C} \oplus 0x2\text{B})\\
0x4\text{C} &= S^{-1}(0x94 \oplus 0x\text{A}7) \oplus S^{-1}(0x42 \oplus 0x\text{A}7)\\
0x90 &= S^{-1}(0x45 \oplus 0x17) \oplus S^{-1}(0x76 \oplus 0x17),
\end{aligned} \tag{3}$$

where the differences between the inverse SubBytes have nothing to do with the coefficient elements of MC_0. Thus, the differential equations are not valid.

13	E3	F3	4D
11	94	07	2B
1D	4A	A7	30
7F	17	8B	C5

(a) Final round key

0A	41	F1	C6
94	6E	C3	53
0B	F0	94	EA
B5	45	58	5A

(b) Fault-free ciphertext

34	41	F1	C6
94	6E	C3	72
0B	F0	90	EA
B5	02	58	5A

(c) Faulty ciphertext obtained from the unprotected WB-AES.

D4	41	F1	C6
94	6E	C3	2C
0B	F0	42	EA
B5	76	58	5A

(d) Faulty ciphertext obtained from our protected AES.

Fig. 6: Final round key, fault-free and faulty ciphertexts (column-major order). Light shaded: involved subkeys of the final round key and corresponding subbytes in the faulty-free ciphertext. Gray shaded: faulty bytes after injecting a single-byte fault.

With a single redundant computation and a clean-up computation, there exist only 256 fault-free triplets of the three first subbytes of the 9-th round input for a fixed key. In other words, only 256 triplets lead to fault collisions. For the rest of faulty 2^{24} - 256 triplets, transformation collisions seem unlikely to take place based on our experimental results; less than 2 inputs to the *T-box* result in transformation collisions in the case of singular matrices.

4.2 Effect of the clean-up computation

In the connection with the attack above, $\mathcal{Z}$ and its decoded state $(\mathcal{N}_z)^{-1}(\mathcal{Z})$ are shown in Fig. 7. The four non-zero bytes in the decoded state imply that there were not successful collisions. In the next encryption, the faulty bytes will distort the four corresponding subbytes of the plaintext, and the errors will be propagated to the whole state after the rounds. Thus, all subsequent ciphertexts are useless for the attacker.
Not only DFA, but also other attacks using the bias in faulty intermediate values require a target to be stateless in order to induce multiple faults without being noticed. Otherwise, some procedures essential to the above attacks cannot be carried out. It is hard to observe the infectiveness of the injected fault by comparing it with a fault-free ciphertext. Therefore, filtering ineffective faults for reducing the key search space is not feasible if there is faulty $\mathcal{Z}$. Getting the fault-free $\mathcal{Z}$, which looks like a state of random numbers, by accurately inducing faults in the clean-up computation is also infeasible for a non-invasive attack. This stateful feature of managing $\mathcal{Z}$ in the proposed encryption is thus effective to prevent various types of FA. This is reminiscent of a sensor-based hardware cryptographic implementation for shielding the internal circuit.

4.3 Performance

For n redundant computations, where the n-th redundancy is dedicated to the clean-up computation, the total table size is calculated as follows. At the first shared computation of Part 1 including the initial XOR of $\mathcal{P}$ and $\mathcal{Z}$, the sum of the table sizes of *TypeII*, *TypeIII*, and *TypeIV* is 290,816 bytes. The sum of the sizes between Part 1 and Part 3 is given by 221,184 $\times$ $(n + 1)$ + 16,384 $\times$ n + 12,288. Finally, the tables of Part 3 need 45,056 bytes. In total, the table size including $\mathcal{Z}$ can be expressed as

$$221,184 \times (n+1) + 16,384 \times n + 348,176.$$

When it comes to table access, the number of lookups in Part 1 is 1,152. Next, the table lookups counted in Part 2 and Part 3 are 432 $\times$ $(n + 1)$ + 128 $\times$ n + 96 and 224, respectively. In total, the number of table lookups are given by

$$432 \times (n+1) + 128 \times n + 1,472.$$

Additionally, there will be load and store operations for $\mathcal{Z}$.

A0	60	6D	2F
9C	63	33	52
6C	24	31	36
FF	D5	70	76

(a) $\mathcal{Z}$

D2	0	0	0
0	0	0	28
0	0	5C	0
0	DE	0	0

(b) Decoded $\mathcal{Z}$

Fig. 7: The result of the clean-up computation in the presence of detected faults. Gray shaded: non-zeros due to the faults.

Table 2: Table size and the number of lookups in the proposed method.

n	Bytes	# of lookups
2	1,044,496	3,024
3	1,282,064	3,584
4	1,519,632	4,144

For $n \in \{2, 3, 4\}$ redundant computations, the table size and the number of lookups (except for ShiftRows) are summarized in Table 2. The AES implementation of Daemen and Rijmen requires 4,352 bytes for lookup tables and approximately 300 operations (lookups and XORs) [8]. Simple time redundancy with n redundant computations based on this implementation will require roughly 300 $\times$ $(n + 1)$ operations. Note that in a software-based redundant implementation, lookup tables are probably reused. However, as explained in Section 2, this is vulnerable to IFA or SIFA attacks. When it comes to software-based infective countermeasures [29, 37], these execute a redundant computation of encryption with up to 30 dummy rounds. In the case of AES-128, infective countermeasures will require approximately 300 $\times$ 5 operations as well as run-time random number generation. Importantly, SIFA can also recover the key from them. Compared to the costs of the existing countermeasures, our countermeasure seems to be costly. However, it takes advantage of only lightweight operations such as lookups for protecting against FA without any run-time random source in the device.

5 Conclusion and Discussion

In this paper, we propose a *table redundancy* method using internally encoded lookup tables for protecting against fault attacks. Because additional redundant computations increase the total table size and the number of lookups, the tables of the outer rounds for an AES-128 algorithm which are not attacked by FA are shared. For the non-shared part of the encryption, redundant computations are performed from the 6-th round to the last MixColumns multiplication based on internally encoded tables generated using different linear and nonlinear transformations. The redundant outcomes, except for the last one, are recombined in such a way to propagate errors in the intermediate values. The result of the last one is summed with the result of the recombination in order to make iterative fault attacks useless. If no fault is detected, it is designed to produce a state of encoded zero. Since this state is combined with the plaintext for each encryption, the previously detected faults will add faulty values to subsequent plaintexts making useless ciphertexts.
In addition to FA, there are still threats of gray-box attacks on internally encoded tables for cryptographic implementations [32]. Importantly, a key-leakage preventive transformation is required to prevent statistical analysis. An alternative is to adapt a masking technique in classical or customized ways [4, 22]. It is also possible for a white-box attacker to extract the key by adopting debuggers or cryptanalysis [2, 23, 24, 26]. When counteracting various threats and merging several techniques, the disadvantages always include the memory requirement and computational costs. For example, if *table redundancy* is applied to a customized masking technique in [22], every redundant computation must be masked and the masks must also be stored in the lookup table. Because the secure implementations of software cryptography consume resources and costs, we must first consider where to apply them and what to protect.

References

1. Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology. pp. 513–525. CRYPTO '97, Springer-Verlag, London, UK, UK (1997), http://dl.acm.org/citation.cfm?id=646762.706179
2. Billet, O., Gilbert, H., Ech-Chatbi, C.: Cryptanalysis of a White Box AES Implementation. In: Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers. pp. 227–240 (2004), http://dx.doi.org/10.1007/978-3-540-30564-4_16
3. Blömer, J., Seifert, J.P.: Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). In: Wright, R.N. (ed.) Financial Cryptography. pp. 162–181. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
4. Bogdanov, A., Rivain, M., Vejre, P.S., Wang, J.: Higher-Order DCA against Standard Side-Channel Countermeasures. In: Polian, I., Stöttinger, M. (eds.) Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11421, pp. 118–141. Springer (2019), https://doi.org/10.1007/978-3-030-16350-1_8
5. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the Importance of Checking Cryptographic Protocols for Faults. In: Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques. pp. 37–51. EUROCRYPT'97, Springer-Verlag, Berlin, Heidelberg (1997), http://dl.acm.org/citation.cfm?id=1754542.1754548
6. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the Importance of Eliminating Errors in Cryptographic Computations. J. Cryptol. 14(2), 101–119 (Jan 2001), http://dx.doi.org/10.1007/s001450010016
7. Breier, J., Khairallah, M., Hou, X., Liu, Y.: A countermeasure against statistical ineffective fault analysis. IEEE Transactions on Circuits and Systems II: Express Briefs 67(12), 3322–3326 (2020)
8. Chow, S., Eisen, P., Johnson, H., Oorschot, P.C.V.: White-Box Cryptography and an AES Implementation. In: Proceedings of the Ninth Workshop on Selected Areas in Cryptography (SAC 2002). pp. 250–270. Springer-Verlag (2002)
9. Clavier, C.: Secret external encodings do not prevent transient fault analysis. In: Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4727, pp. 181–194. Springer (2007), https://iacr.org/archive/ches2007/47270181/47270181.pdf
10. Daemen, J., Dobraunig, C., Eichlseder, M., Gross, H., Mendel, F., Primas, R.: Protecting against statistical ineffective fault attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems 2020(3), 508–543 (Jun 2020), https://tches.iacr.org/index.php/TCHES/article/view/8599
11. Derbez, P., Fouque, P.A., Leresteux, D.: Meet-in-the-middle and impossible differential fault analysis on aes. In: CHES. Lecture Notes in Computer Science, vol. 6917, pp. 274–291. Springer (2011), https://www.iacr.org/archive/ches2011/69170275/69170275.pdf
12. Dobraunig, C., Eichlseder, M., Korak, T., Mangard, S., Mendel, F., Primas, R.: SIFA: exploiting ineffective fault inductions on symmetric cryptography. IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018(3), 547–572 (2018), https://doi.org/10.13154/tches.v2018.i3.547-572

13. Dusart, P., Letourneux, G., Vivolo, O.: Differential Fault Analysis on A.E.S. In: Zhou, J., Yung, M., Han, Y. (eds.) Applied Cryptography and Network Security. pp. 293–306. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
14. Fuhr, T., Jaulmes, E., Lomné, V., Thillard, A.: Fault Attacks on AES with Faulty Ciphertexts Only. In: Proceedings of the 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography. pp. 108–118. FDTC '13, IEEE Computer Society, Washington, DC, USA (2013), http://dx.doi.org/10.1109/FDTC.2013.18
15. Ghalaty, N.F., Yuce, B., Taha, M., Schaumont, P.: Differential Fault Intensity Analysis. In: 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography. pp. 49–58 (Sep 2014)
16. Gierlichs, B., Schmidt, J.M., Tunstall, M.: Infective Computation and Dummy Rounds: Fault Protection for Block Ciphers without Check-before-Output . Lecture Note in Computer Science (LNCS), Springer (2012), in press
17. Giraud, C.: DFA on AES. In: Proceedings of the 4th International Conference on Advanced Encryption Standard. pp. 27–41. AES'04, Springer-Verlag, Berlin, Heidelberg (2005), http://dx.doi.org/10.1007/11506447_4
18. Khairallah, M., Bhasin, S., Abdellatif, K.M.: On comparison of countermeasures against statistical ineffective fault attacks. In: 2019 31st International Conference on Microelectronics (ICM). pp. 122–125 (2019)
19. Kim, C.H.: Differential Fault Analysis of AES: Toward Reducing Number of Faults. Inf. Sci. 199, 43–57 (Sep 2012), https://doi.org/10.1016/j.ins.2012.02.028
20. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. pp. 388–397 (1999), http://dx.doi.org/10.1007/3-540-48405-1_25
21. Koren, I., Krishna, C.M.: Fault-Tolerant Systems. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edn. (2007)
22. Lee, S., Kim, M.: Improvement on a masked white-box cryptographic implementation. IEEE Access 8, 90992–91004 (2020)
23. Lee, S., Choi, D., Choi, Y.J.: Conditional Re-encoding Method for Cryptanalysis-Resistant White-Box AES. vol. 5. Electronics and Telecommunications Research Institute (Oct 2015), http://dx.doi.org/10.4218/etrij.15.0114.0025
24. Lepoint, T., Rivain, M., Mulder, Y.D., Roelse, P., Preneel, B.: Two Attacks on a White-Box AES Implementation. In: Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers. pp. 265–285 (2013), http://dx.doi.org/10.1007/978-3-662-43414-7_14
25. Lomne, V., Roche, T., Thillard, A.: On the Need of Randomness in Fault Attack Countermeasures - Application to AES. In: Proceedings of the 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography. pp. 85–94. FDTC '12, IEEE Computer Society, Washington, DC, USA (2012), http://dx.doi.org/10.1109/FDTC.2012.19
26. Michiels, W., Gorissen, P., Hollmann, H.D.L.: Cryptanalysis of a Generic Class of White-Box Implementations. In: Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers. pp. 414–428 (2008), http://dx.doi.org/10.1007/978-3-642-04159-4_27
27. Moradi, A., Shalmani, M.T.M., Salmasizadeh, M.: A Generalized Method of Differential Fault Attack Against AES Cryptosystem. In: Proceedings of the 8th International Conference on Cryptographic Hardware and Embedded Systems. pp. 91–

100. CHES'06, Springer-Verlag, Berlin, Heidelberg (2006), http://dx.doi.org/10.1007/11894063_8
28. Mukhopadhyay, D.: An Improved Fault Based Attack of the Advanced Encryption Standard. In: Proceedings of the 2Nd International Conference on Cryptology in Africa: Progress in Cryptology. pp. 421–434. AFRICACRYPT '09, Springer-Verlag, Berlin, Heidelberg (2009), https://doi.org/10.1007/978-3-642-02384-2_26
29. Patranabis, S., Chakraborty, A., Mukhopadhyay, D.: Fault Tolerant Infective Countermeasure for AES. In: Proceedings of the 5th International Conference on Security, Privacy, and Applied Cryptography Engineering - Volume 9354. pp. 190–209. SPACE 2015, Springer-Verlag, Berlin, Heidelberg (2015), https://doi.org/10.1007/978-3-319-24126-5_12
30. Patranabis, S., Chakraborty, A., Nguyen, P.H., Mukhopadhyay, D.: A Biased Fault Attack on the Time Redundancy Countermeasure for AES. In: Revised Selected Papers of the 6th International Workshop on Constructive Side-Channel Analysis and Secure Design - Volume 9064. pp. 189–203. COSADE 2015, Springer-Verlag New York, Inc., New York, NY, USA (2015), http://dx.doi.org/10.1007/978-3-319-21476-4_13
31. Phan, R.C.W., Yen, S.M.: Amplifying side-channel attacks with techniques from block cipher cryptanalysis. In: Proceedings of the 7th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Applications. p. 135–150. CARDIS'06, Springer-Verlag, Berlin, Heidelberg (2006), https://doi.org/10.1007/11733447_10
32. Rivain, M., Wang, J.: Analysis and Improvement of Differential Computation Attacks against Internally-Encoded White-Box Implementations. IACR Trans. Cryptogr. Hardw. Embed. Syst. 2019(2), 225–255 (2019), https://doi.org/10.13154/tches.v2019.i2.225-255
33. Saha, D., Mukhopadhyay, D., Chowdhury, D.R.: A Diagonal Fault Attack on the Advanced Encryption Standard. IACR Cryptology ePrint Archive 2009, 581 (2009)
34. Subidh Ali, S., Mukhopadhyay, D., Tunstall, M.: Differential fault analysis of AES: Towards reaching its limits. Journal of Cryptographic Engineering 3 (06 2012)
35. Sung-Ming Yen, Joye, M.: Checking before output may not be enough against fault-based cryptanalysis. IEEE Transactions on Computers 49(9), 967–970 (2000)
36. Takahashi, J., Fukunaga, T., Yamakoshi, K.: DFA Mechanism on the AES Key Schedule. In: Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007). pp. 62–74 (Sep 2007)
37. Tupsamudre, H., Bisht, S., Mukhopadhyay, D.: Destroying Fault Invariant with Randomization. In: Batina, L., Robshaw, M. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2014. pp. 93–111. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
38. Zhang, F., Lou, X., Zhao, X., Bhasin, S., He, W., Ding, R., Qureshi, S., Ren, K.: Persistent fault analysis on block ciphers. IACR Transactions on Cryptographic Hardware and Embedded Systems 2018(3), 150–172 (Aug 2018), https://tches.iacr.org/index.php/TCHES/article/view/7272