Table Redundancy Method for Protecting against Differential Fault Analysis in White-box Cryptography

Seungkwang Lee[1] and Myungchul Kim[2]

[1] School of Computing, KAIST and
Information Security Research Division, ETRI, Korea.
skwang@etri.re.kr,
[2] School of Computing, KAIST, Korea.
mck@kaist.ac.kr

Abstract. Differential Fault Analysis (DFA) intentionally injects some fault into the encryption process and analyzes a secret key from the mathematical relationship between faulty and fault-free ciphertexts. Even white-box cryptographic implementations are still vulnerable to DFA. A common way to defend DFA is to use some type of redundancy such as time or hardware redundancy. However, previous work on software-based redundancy method can be easily bypassed by white-box attackers, who can access and even modify all resources. In this paper, we propose a secure software redundancy named *table redundancy* that exploits the characteristic of table diversity in white-box cryptography. We show how to apply this table redundancy technique to a white-box AES implementation with a 128-bit key. To prevent significant degradation of performance, the lookup tables which are not under DFA are shared and table redundancy are applied to the inner rounds under DFA. The outputs of the redundant computations are the SubBytes output multiplied by the MixColumns matrix in the 9-th round and encoded by different transformations. The XOR operation combines those redundant intermediate values and the combined transformation is canceled out in the following shared part of the encryption. Our security analysis shows that a success probability of DFA on our table redundancy is negligible and a brute-force attack becomes too costly. With three redundant computations, the total table size and the number of lookups are less than double compared to a non-protected implementation.

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 showed a successful attack on a CRT-RSA algorithm with both a faulty and a fault-free signature of the same message. Such attacks are known as fault attacks. Since then, the fault attack was also applied to the block ciphers by Biham and Shamir and it was called Differential Fault Analysis (DFA) [2]. 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 [15]. So far, this attack on AES has been improved by many studies in such a way to require less brute-force search and faulty ciphertexts [4, 10, 20, 29, 36, 37].

Most of countermeasures of DFA can be categorized into detection and infection. A detection method based on various types of redundancy strongly relies on the comparison step in general. An infection method, on the other hand, propagates the effect of faults to a wide range, making the faulty ciphertext unexploitable. However, a white-box attacker, who has full privilege of the target environment, is able to skip and change the flow sequence of the target implementation so that the detection and infective methods have no effect.

To solve this problem, we focus on white-box cryptography as a software countermeasure for protecting against DFA performed even in the white-box attack model. Originally, white-box cryptography aims to protect the keys hidden in cryptographic implementations from white-box attacks. After white-box DES (WB-DES) [9] and AES (WB-AES) [8] implementations were introduced, several vulnerabilities were presented, including algebraic analysis [3, 16, 24, 27, 39], and side-channel analysis [7, 35]. In particular, DFA on white-box cryptography was also demonstrated, where a white-box attacker can precisely inject any fault anywhere [34]. Here we remark that one of the open problems in white-box cryptography was to find certain techniques of white-box cryptography to improve the security against fault attacks [19].

In this paper, we present the technique of white-box cryptography to prevent DFA. To do so, we propose a new type of redundancy aptly named *table redundancy* using the white-box diversity. We show that DFA is unlikely to reveal the correct key from our WB-AES implementation, and additional costs of table size and lookups are less than 2 times, compared to a non-protected WB-AES if we use three redundant computations. The rest of the paper is organized as follows. Section 2 provides the basic principle of white-box cryptography with AES-128, and then explains DFA and previous countermeasures. Section 3 presents our key idea and proposes our WB-AES implementation. We then analyze its security and performance in Section 4. Section 5 concludes this paper and discusses how to improve white-box cryptography with other issues.

2 Background

In this section, we briefly explain the basic concepts of white-box cryptography and DFA. To provide a concrete example, a non-protected WB-AES implementation [8] with a 128-bit key is introduced. We then explain DFA and countermeasures.

2.1 White-box Cryptography and AES

The current white-box cryptography of block ciphers is mostly implemented in a table-based manner with nonlinear and linear transformations (the term *encoding* is often used) to hide key-dependent intermediate values. Here, the lookup table size is problematic to map the set of n-bit plaintexts to the n-bit ciphertexts under a fixed n-bit key. For example, if $n = 128$ like in the case of AES-128, the memory requirement of the entire lookup table would take $2^{128} \cdot 128$ bits. To solve this problem, a lookup table is generated for each step and each round, and then combined in a networked manner.

Since a white-box attacker using a disassembler/degugger is able to learn the content of any lookup table, nonlinear and linear transformations are used to protect the key incorporated in the table. Given a lookup table $\mathcal{T}$, we choose two transformations f and g (composed of nonlinear and linear transformations) in order to protect inputs and outputs, respectively, and produce the new table $\mathcal{T}'$:

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

To obtain the value of $\mathcal{T}(x)$, we input $f(x)$ to $\mathcal{T}'$ and then apply g^{-1}. If the $\mathcal{T}$ output feeds into another table $\mathcal{R}$, then transformations are applied in a networked manner so that the output transformation of $\mathcal{T}$ and the input transformation of $\mathcal{R}$ cancel each other out. 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

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

Let's take a close look at WB-AES. In the initial WB-AES implementation (with a 128-bit key) proposed by Chow *et al.* [8], AddRoundKey, SubBytes, and part of MixColumns are combined into a series of lookup tables 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, $k^{10}$)
$ciphertext$ ← 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. Upon this description, 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:

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

Through round 1 to 9, each *T-box* output is combined with MixColumns by multiplying a 32×32 matrix MC (representing MixColumns) defined to be:

$$\begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix}$$

To avoid huge tables, the MC matrix is divided into four column matrices MC_i (i = 0. . .3) and the multiplication is also performed separately. Let $[x_0, x_1, x_2, x_3]^T$ be a column vector of the intermediate state after mapping the round input to a *T-box*. By the linearity of a matrix multiplication, we have:

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

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:

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

As mentioned, linear ("mixing" bijection) and nonlinear transformations are used to obfuscate the tables. With respect to linear transformation, 8×8 mixing bijections are used to diffuse *T-box* inputs, and 32×32 mixing bijections are applied to Ty_i outputs. The nonlinear transformation is performed by two four-bit concatenated ones to avoid huge exclusive-OR (XOR) lookup tables. When generating the XOR lookup table, the inverse linear transformations are not involved as shown in Fig. 1 since the distributive property of multiplication over addition is satisfied.

In [8], *TypeII* combines T^1 to T^9 with Ty_i. Since the *TypeII* output is transformed by a 32×32 linear transformation, it is required to replace it with four 8×8 linear transformations. By doing so, a single-byte input to *TypeII* in the next round can be simply decoded by the inverse 8×8 linear transformation. This replacement contributes to the reduced size of *TypeII*, and is performed by *TypeIII*. All of the XOR operations between encoded intermediate values are conducted by *TypeIV*. This takes two four-bit encoded inputs and provides a four-bit encoded XOR result of the decoded inputs. This is aptly named *TypeIV_II* when used to combine the intermediate values from *TypeII*. Similarly, it is commonly named *TypeIV_III* when used to combine the lookup values from *TypeIII*. Fig. 2 illustrates the overall table lookup between *TypeII*, *TypeIII* and *TypeIV*. In the case of T^{10}, its output is a subbyte of the ciphertext and thus it is not protected unless the external encoding is used. Here, let *TypeV* denote the final round lookup table which provides the T^{10} output taking an encoded input.

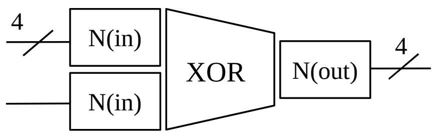
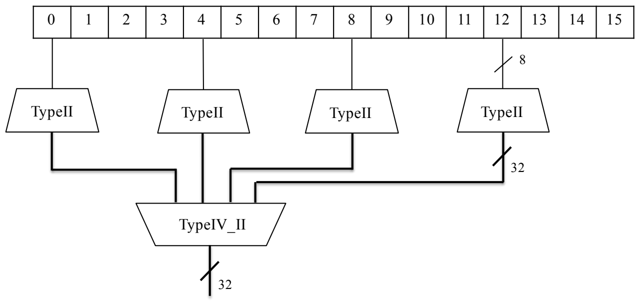


Fig. 1: A schematic of *TypeIV* generation. N: nonlinear transformation.

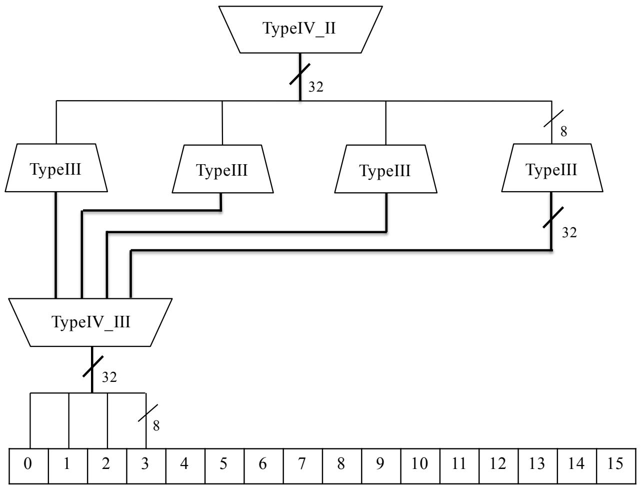
The external encoding is often used to encode the plaintext and ciphertext, and *TypeI* is used to perform these external encodings. However, we do not take into account because it reduces compatibility with other encryption systems that do not use external encoding.
There are two security metrics: the white-box diversity and ambiguity [8]. 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 exploit the diversity which can produce abundant lookup tables using the countless transformations.

2.2 DFA on AES

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 input and obtaining faulty ciphertexts. (3) analyzing the relationship between the faulty-free and faulty ciphertexts to reduce the search space of the key. An analysis of the relationship depends on the fault model with respect to the fault location and fault characteristic as follows. First, if an attacker is able to inject a single bit fault by setting or clearing a particular bit of the first round key (used in the initial AddRoundKey), each bit of the key can be recovered for each faulty ciphertext [4]. It is also possible to inject a single bit fault at the beginning of the final round [15]. In this attack, a 128-bit key can be determined by using less than 50 faulty ciphertexts.
Second, an attacker can inject a single byte or multiple byte fault between the 8-th round output and the 9-th MixColumns input. Because of MixColumns in the 9-th round, a disturbance in a subbyte of the round input affects four bytes in the round output. Because the final round does not involve MixColumns,



(a) *TypeII* and *TypeIV_II* lookup.



(b) *TypeIII* and *TypeIV_III* lookup

Fig. 2: A simple overview of *TypeII*, *TypeIII* and *TypeIV* table lookups.

the difference remains in four bytes at the ciphertext. Among many working principles of DFA based on this fault propagation, we briefly review a technique using the four 9-th round differential equations [36].

Suppose that a single byte difference is generated at the first subbyte, say x, of the 9-th round input. Let denote the difference by δ and the faulty byte 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 round output is represented by $(2\delta', \delta', \delta', 3\delta')$, where 2, 1, 1, and 3 are the elements of MC_0. ShiftRows will move the difference to four different locations as shown in Fig. 3.

With fault-free and faulty ciphertexts for the same plaintext, DFA can express the four-byte difference in terms of 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, $\widetilde{C}_1 = C_1 \oplus \varDelta_1$. Then we have the following equa-

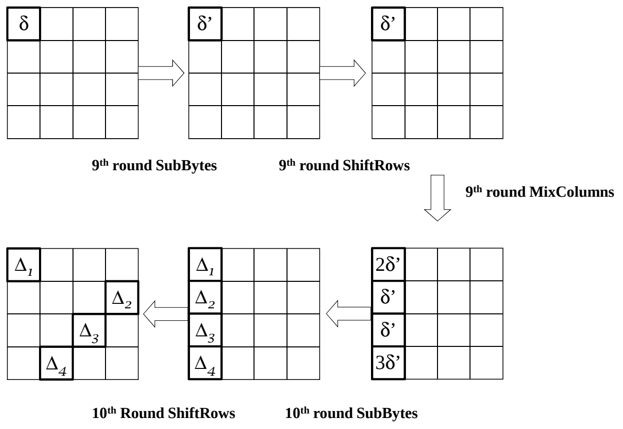


Fig. 3: Fault propagation across the last two rounds of AES.

tions, which take as inputs the fault-free and faulty ciphertexts, and each key candidate.

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

where $K_i^* \in \mathrm{GF}(2^8)$ means each subkey candidate. These equations are called 9-th round differential equations [36] which will reduce the search space of key quartet to an expected value of 2^8. This gives us 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.
For DFA on WB-AES [34], the differential equations are still valid though T^{10} in the last round of WB-AES combines two round keys. Thus, DFA can recover the key from WB-AES in the same way. It may be noted here that we do not take into account DFA on the AES key schedule since WB-AES is a key-instantiated implementation. In addition, we assume that there is no external encodings to apply DFA on WB-AES. In white-box attacker's DFA, the precise location for injecting faults can be found by static and dynamic code analysis. To that end, we can use several techniques including a DBI framework such as PIN [26] and Valgrind [30] or a scriptable debugger like vtrace and gdb.
Third, an attacker can inject faults between the 7-th round output and the 8-round MixColumns input. Injecting a single byte fault at this location gives an additional relationship similar to 9-th round differential equations. They call it

8-th round differential equations. An attacker with a single faulty ciphertext can further reduce the search space of the key from 2^{32} to 2^{8} using 8-th and 9-th round differential equations, with 2^{32} time complexity as each of 2^{32} candidates of the final round key are tested by set of four equations. Here, this time complexity can be reduced to 2^{30} by an acceleration technique [36].

There are some multiple byte fault attacks. Authors in [28] present two different multiple byte fault attacks covering all possible faults on the 9-th round MixColumns input. The first attack supposes that at least one byte of MixColumns in one column is fault free and requires only 6 faulty ciphertexts in average for discovering the key. In the second attack, all four bytes of one column are supposed to be faulty and then approximately 1,500 faulty ciphertexts can recover the key. In [33], a diagonal fault model using multiple byte faults is proposed. The authors divide the state matrix into four diagonals, each of four bytes of the state matrix. 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.

Interestingly, the probability of successful attacks is enhanced when an attacker is capable of injecting a biased fault [11, 13, 32]. These methods often use the *stuck-at* model in order to fix a target byte to a particular value. In this case, the correct key candidate produces small changes in the faulty intermediate value compared to other wrong key candidates, and thus the search space of the final round key is steeply reduced. The biased fault can be also used to enhance the probability of passing the comparison step of the fault detection.

2.3 DFA Countermeasure

Detection-based countermeasures, also known as Concurrent Error Detection (CED) [21], use additional redundancy to detect fault injection. Previous CED techniques are classified into four types of redundancy [17]. 1) A information redundancy is based on error detecting codes such as parity bit and robust code. 2) A time redundancy is a classical fault tolerance technique in which a cryptographic operation is computed twice 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, as noted previously, a biased fault can effectively defeat the time redundancy countermeasure because of relatively high fault collision probability [32]. 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 CED techniques. For example, a fault can be detected by comparison of the original plaintext with a decrypted plaintext. In this case, both encryption and decryption hardware are used on a single chip. Here we remind that this study focuses on software countermeasures, and the previous software-based detection can be easily disabled by white-box attacks intentionally skipping instructions or modifying intermediate values.

Infective countermeasures, on the other hand, use the diffusion effects of faults instead of comparative computations to make it impossible to obtain meaningful information from ciphertext. Specifically, Tupsamudre *et al.* [38] proposed to use intermediate dummy rounds to overcome the weaknesses of deterministic diffusion based infective methods [25] and a random variation [14]. Patranabis *et al.* [31] 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 an attacker who can change the flow sequence. However, we note that these cannot be a solution in the white-box attack model.

3 Proposed Method

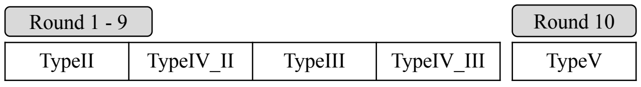
In this section, we present our white-box implementation for protecting against DFA. For this purpose, we propose a new concept, aptly named *table redundancy.* Based on this redundancy technique, we replace a detective comparison step with infective XOR operations. To explain our redundancy design, we divide WB-AES into three parts as depicted in Fig. 4.

1. From Round 1 to 6
2. From Round 7 to *TypeII* in Round 9
3. From *TypeIV_II* in Round 9 to Round 10

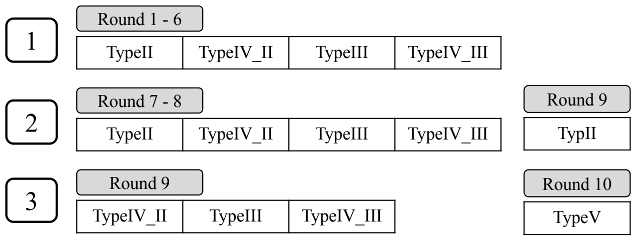
In order to reduce the total table size and lookups, the part (1) of the first 6 rounds are shared because those are not under the attack in this paper. For the part (2), we perform redundant computations with different sets of lookup tables generated using different transformations. This is why we call it table redundancy. Here we note that *TypeII* computes the SubBytes output multiplied by MC_i and *TypeIV_II* combines them with the XOR operation. Between the part (2) and (3), we perform the XOR operation as an infective computation with the redundant outputs of the part (2). This XOR result will be the input to the part (3) computing the ciphertext.
Previously, we provided a review of DFA mounted in several rounds. Before introducing our method we note that single bit fault attacks in the initial AddRoundKey are excluded from the white-box cryptographic point of view. This is because there is no guarantee that the difference in a certain bit in the input leads to a consistent difference in the output due to nonlinear and linear transformations applied to white-box cryptography. For this reason, we assume that DFA on WB-AES implementations takes place between the 7-th and 9-th round inputs.

3.1 Key Idea

Table redundancy. For our WB-AES implementation generated with the key $\mathcal{K}$, let $\mathcal{T}^b$ (b stands for “begin”) denote a set of shared lookup tables for the part (1) to be used in the 1 - 6 rounds. For the part (2), we generate two different sets of lookup tables which are generated using different sets of nonlinear and



(a) Normal sequence of table lookup



(b) Our partition and table lookup

Fig. 4: Our partition of WB-AES lookup tables.

linear transformations. Due to the white-box diversity, a key can generate different lookup tables using different transformations, and different lookup tables will output different intermediate values. Let $\mathcal{T}^0$ and $\mathcal{T}^1$ denote these two sets of lookup tables. Given a plaintext $\mathcal{P}$, the part (1) of the encryption using $\mathcal{T}^b$ is followed by the part (2) which is computed twice by $\mathcal{T}^0$ and $\mathcal{T}^1$. 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 encoded intermediate values from $\mathcal{T}^0$ and $\mathcal{T}^1$, respectively. In general, $\mathcal{Q}^0$ and $\mathcal{Q}^1$ will be provided in a 4×4×4 array because *TypeII* maps an 8-bit input to a 32-bit output. Fig. 5 briefly describes our table redundancy with a redundant computation and TABLE 1 explains the other notations used. We sometimes abuse the notations $\mathcal{N}^i$ and $\mathcal{L}^i$ to mean a substitution box of $\mathcal{N}^i$ and a binary matrix used in $\mathcal{L}^i$, respectively.

Notation	Description
$\mathcal{T}^x$	XOR lookup tables to combine redundant computation results.
$\mathcal{N}^i$	Nonlinear transformation applied to $\mathcal{Q}^{i \in \{0,1\}}$
$\mathcal{L}^i$	Linear transformation applied to $\mathcal{Q}^i$
$\mathcal{G}i$	$\mathcal{N}^i \circ \mathcal{L}^i$
$\mathcal{N}^x$	Nonlinear transformation applied to $\mathcal{Q}^x$
$\mathcal{Q}^x$	The lookup value from $\mathcal{T}^x$

Table 1: Notations for the key idea illustrated in Fig. 5

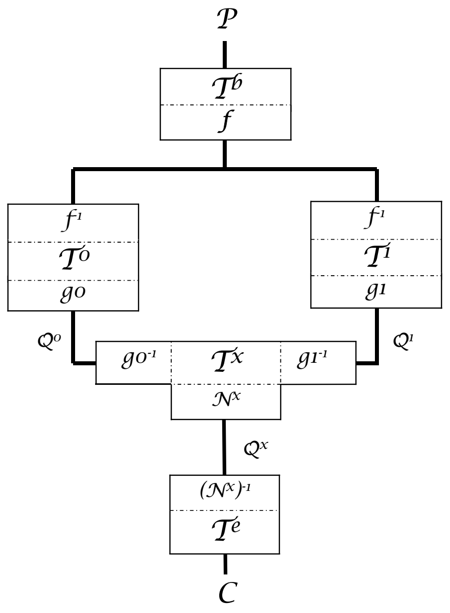


Fig. 5: Simple description of our key idea with a redundant computation. f: the encoding applied to the 6-th round output, $g0, g1$: the encoding applied to the 9-th round *TypeII* output.

XOR instead of comparison. The next step is to perform an infective computation with $\mathcal{Q}^0$ and $\mathcal{Q}^1$ so that fault injection cannot lead to valid differential equations. To achieve this goal, we replace a normal comparison step with XOR using a lookup table $\mathcal{T}^x$ (x stands for "XOR"), a type of *TypeIV* that contains a different number of copies. The main advantage of placing the XOR operation with $\mathcal{Q}^0$ and $\mathcal{Q}^1$ here is that a four-byte *TypeII* output is protected by a 32×32 linear transformation and thus a single-byte manipulation by an attacker has an infectious effect on the other three bytes. Further more, the total table size and table lookups can be reduced by sharing the part (3).
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

$$\mathcal{Q}^i = \mathcal{G}i(y_j) = \mathcal{N}^i \circ \mathcal{L}^i(y_j),$$

where $i \in \{0,1\}$ and $y_j = Ty_{j\in\{0,1,2,3\}}(x)$. 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 $\mathcal{T}^e$ (e stands for "end"), after *TypeIV_II* combines the *TypeII* output in the 9-th round, *TypeIII* performs $\mathcal{L}^e$, the inverse linear transformation applied to y_j

and applies a 8×8 linear transformation on each byte, as mentioned. Thus,

$$\mathcal{L}^0 \oplus \mathcal{L}^1 = (\mathcal{L}^e)^{-1}.$$

For this reason, $\mathcal{L}^e$ must be invertible (non-singular) while $\mathcal{L}^0$ and $\mathcal{L}^1$ do not necessarily have to be invertible. So we pick those table as follows:

- Generate a $32{\times}32$ invertable binary matrix $\mathcal{L}^e$.
- Generate a random $32{\times}32$ binary matrix $\mathcal{L}^0$.
- Compute $\mathcal{L}^1 = (\mathcal{L}^e)^{-1} \oplus \mathcal{L}^0$.

Then *TypeIII* in the 9-th round is generated with $\mathcal{L}^e$, and the remaining *TypeIV_III* and *TypeV* compute a ciphertext $\mathcal{C}$.

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 values. 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 with additional tables generated using different transformations, depicted as $\mathcal{T}^2$ and $\mathcal{T}^3$ in Fig. 6. If n = 3, we have three redundant computations as illustrated in Fig. 6b. Here, $\mathcal{L}^n$ is obtained from $\mathcal{L}^e$ and n random matrices $\mathcal{L}^{i\in[0,n-1]}$ as follows:

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

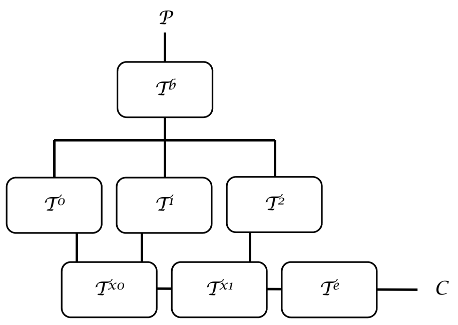
In addition, we need more $\mathcal{T}^x$ tables for the XOR operation of redundant computations. These are aptly named $\mathcal{T}^{x0}$, $\mathcal{T}^{x1}$ and $\mathcal{T}^{x2}$. In the following section, we analyze the security and performance with n redundant computations in more detail.

4 Evaluation

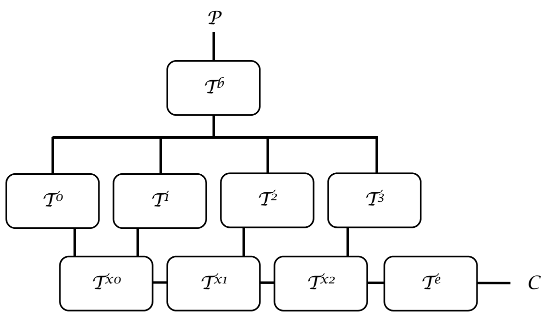
The security evaluation in this section analyzes a success probability and complexity of DFA on our method with n redundant computations and the performance is evaluated compared to an unprotected WB-AES implementation.

4.1 Security

Now 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 differential equations can be occurred if each of $n+1$ disturbed bytes is decoded to the same *T-box* input, say $x^f \in \mathrm{GF}(2^8)$. Then the probability of this event is $(2^{-8})^n$, and this is negligible as n increases. For example, this is approximately 5×10^{-8} if $n = 3$.



(a) With two redundant computations



(b) With three redundant computations

Fig. 6: Our proposed method with extended redundancy for enhanced security.

Suppose that a fault collision is not occurred in $\mathcal{T}^i$ of which $\mathcal{L}^i$ 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 a singular linear transformation. We call it a transformation collision. Note that the number of nonsingular $m \times m$ binary matrices denoted by $\#GL_m(\mathbb{F}_2)$ is negligible compared to the number of singular $m \times m$ binary matrices denoted by $\#Sg_m(\mathbb{F}_2)$, for $m = 32$ in the case of $\mathcal{L}$, where

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

For this reason, $\mathcal{L}^{i\in[0,n]}$ randomly generated is singular with a overwhelming probability, and thus we take transformation collisions into account. To do so, we generated 10,000 random singular matrices and counted the number of the *T-box* inputs producing transformation collisions for each matrix. As a result, an average of 1.47 inputs (among 256 elements) caused transformation collisions. In probability this is less than 2/256. Then, the probability of $k \in [0, n]$ fault collisions and $n-k$ transformation collisions can be upper bounded by

$$\sum_{k=0}^{n}(2^{-8})^k \cdot [2/256 \cdot \#Sg_{32}/(2^{32})^2]^{n-k}.$$

Another leakage of faulty ciphertext also takes place if a directly manipulated quartet in $\mathcal{Q}^x$ is feasible. Suppose that an attacker injects a single byte fault into the first subbyte of the 9-th round inputs in $\mathcal{T}^0$ to $\mathcal{T}^n$. Then a disturbed quartet q^x in $\mathcal{Q}^x$ is said to be feasible if

$$^{\exists}x' \in GF(2^8) \text{ such that } (\mathcal{N}^x)^{-1}(q^x) = (\mathcal{L}^e)^{-1}(Ty_0(x')).$$

We can easily know that this event happens with a negligible probability of $(2^{-8})^3$ due to the fixed coefficient of MC_0.
A brute-force attack can fix a target byte in the 8-th round input in $\mathcal{T}^0$ to a particular value and try every combination of each target byte in $\mathcal{T}^1$ to $\mathcal{T}^n$. For each trial of the combination, the attack is conducted with the 2^8 key search space and 2^{30} time complexity as explained previously. In this brute-force attack, the number of possible combinations is $(2^8)^n$. Consequently, our table redundancy method can significantly reduce the success probability of obtaining valid differential equations and also steeply increase the number of trials of brute-force attacks.

4.2 Performance

In the case of an unprotected WB-AES implementation, the table size and the number of lookups are decided by the original computation with ($\mathcal{T}^b$, $\mathcal{T}^0$, $\mathcal{T}^e$) provided that there is no external encoding. These are calculated as shown in Table 2.

	Size (byte)	Lookup
TypeII	147,456	144
TypeIV_II	110,592	864
TypeIII	147,456	144
TypeIV_III	110,592	864
TypeV	4,096	16
Total	520,192	2,032

Table 2: Table size and lookups of an unprotected WB-AES.

We represent the size and the number of lookups of the listed table for each round using the the following notations:

- s_1 : the size of *TypeII* or *TypeIII* (for each round)

- s_2 : the size of *TypeIV* (*TypeIV_II* or *TypeIV_III*)
- s_3 : the size of *TypeV*
- l_1 : the number of lookups on *TypeII* or *TypeIII*
- l_2 : the number of lookups on *TypeIV*
- l_3 : the number of lookups on *TypeV*.

In the unprotected WB-AES, the total table size is then represented by $(9 \cdot 2 \cdot s) + s_3 = 520{,}192$ bytes and the total number of lookups is by $(9 \cdot 2 \cdot l) + l_3 = 2{,}032$, where $s = s_1 + s_2$ and $l = l_1 + l_2$.
In our protected implementation with n redundant computations, the total table size consists of

- $\mathcal{T}^b$: $6 \cdot 2 \cdot s$
- $\mathcal{T}^0 - \mathcal{T}^n$: $[2 \cdot 2 \cdot s + s_1] \cdot (n+1)$
- $\mathcal{T}^x$: $4 \times 4 \times 4 \times 2 \times 128 \times n = 16{,}384 \times n$
- $\mathcal{T}^e$: $s_1 + 2 \cdot s_2 + s_3$,

and this requires $(4 \cdot s + s_1 + 16,384) \cdot n$ bytes additionally compared to the unprotected one. For $s = 28{,}672$ bytes, the table size will increase by 442,368 bytes if $n = 3$. This is an increase of about 85 percent in size. Similarly, the number of table lookups will increase by $(4 \cdot l + l_1 + 128) \cdot n$, where $l = 112$. If $n = 3$, the table lookups increase approximately 94 percent. In conclusion, three redundant computations will increase the table size and number of lookups by less than twice.

5 Conclusion and Discussion

In this paper, we propose a table redundancy method for protecting against DFA in the white-box cryptographic implementation. Because additional redundant computations increase the total table size and the number of lookups, we share the lookup tables of the outer rounds which are not attacked by DFA in the white-box cryptographic implementation. For the non-shared part of the encryption, redundant computations are performed from the 7-th round to the last MixColumns multiplication based on table redundancy generated using different transformations. Then the following XOR operations provide an infective computation using the intermediate values of table redundancy. The rest part of the encryption cancels out the combined transformation and computes the ciphertext. Applying three redundant computations to WB-AES with a 128-bit key roughly doubles the table size and the number of lookups, compared to an unprotected WB-AES implementation.
In addition to DFA, there are still several problems to solve for the secure white-box cryptography. First, a key-leakage preventive transformation is required to prevent power analysis on white-box cryptography. Previously, a masked white-box implementation was proposed [22] to protect against power analysis, the memory requirement is too costly to be adopted in low-cost devices. Second, a countermeasure of cryptanalysis should be combined to the above key-leakage

preventive technique. Although there are some commercial products [1, 12, 18] of white-box cryptography on the market, there is not enough study on the protection of cryptanalysis [23].

References

1. Axsan white-box cryptographic solution.: https://www.arxan.com/technology/white-box-cryptography/
2. 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
3. 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
4. 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)
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. Bos, J.W., Hubain, C., Michiels, W., Teuwen, P.: Differential Computation Analysis: Hiding your White-Box Designs is Not Enough. vol. 2015, p. 753 (2015), http://dblp.uni-trier.de/db/journals/iacr/iacr2015.html#BosHMT15
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. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: A White-Box DES Implementation for DRM Applications. In: Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers. pp. 1–15 (2002), http://dx.doi.org/10.1007/978-3-540-44993-5_1
10. 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)
11. 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
12. Gemalto white-box cryptographic solution.: https://sentinel.gemalto.com/software-monetization/white-box-cryptography/
13. 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)
14. 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

15. 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
16. Goubin, L., Masereel, J., Quisquater, M.: Cryptanalysis of White Box DES Implementations. In: Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers. pp. 278–295 (2007), http://dx.doi.org/10.1007/978-3-540-77360-3_18
17. Guo, X., Mukhopadhyay, D., Karri, R.: Provably Secure Concurrent Error Detection Against Differential Fault Analysis. IACR Cryptology ePrint Archive 2012, 552 (2012)
18. InsideSecure white-box cryptographic solution.: https://www.insidesecure.com/Products/Application-Protection/Software-Protection/WhiteBox
19. Joye, M.: On WhiteBox Cryptography (2008), http://joye.site88.net/papers/Joy08whitebox.pdf
20. 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
21. Koren, I., Krishna, C.M.: Fault-Tolerant Systems. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edn. (2007)
22. Lee, S., Kim, T., Kang, Y.: A Masked White-Box Cryptographic Implementation for Protecting Against Differential Computation Analysis. IEEE Transactions on Information Forensics and Security 13(10), 2602–2615 (Oct 2018)
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. Luk, C., Cohn, R.S., Muth, R., Patil, H., Klauser, A., Lowney, P.G., Wallace, S., Reddi, V.J., Hazelwood, K.M.: Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In: Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15, 2005. pp. 190–200 (2005), http://doi.acm.org/10.1145/1065010.1065034
27. 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
28. 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

29. 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
30. Nethercote, N., Seward, J.: Valgrind: a Framework for Heavyweight Dynamic Binary Instrumentation. In: Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, San Diego, California, USA, June 10-13, 2007. pp. 89–100 (2007), http://doi.acm.org/10.1145/1250734.1250746
31. 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
32. 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
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. Sanfelix, E., Mune, C., de Haas, J.: Unboxing the White-Box: Practical Attacks against Obfuscated Ciphers. In: Presented at BlackHat Europe 2015 (2015), https://www.blackhat.com/eu-15/briefings.html
35. Sasdrich, P., Moradi, A., Güneysu, T.: White-Box Cryptography in the Gray Box - - A Hardware Implementation and its Side Channels -. In: Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. pp. 185–203 (2016), http://dx.doi.org/10.1007/978-3-662-52993-5_10
36. Subidh Ali, S., Mukhopadhyay, D., Tunstall, M.: Differential fault analysis of AES: Towards reaching its limits. Journal of Cryptographic Engineering 3 (06 2012)
37. 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)
38. 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)
39. Wyseur, B., Michiels, W., Gorissen, P., Preneel, B.: Cryptanalysis of White-Box DES Implementations with Arbitrary External Encodings. In: Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers. pp. 264–277 (2007), http://dx.doi.org/10.1007/978-3-540-77360-3_17