symmetry

MDPI

Article

Meet-in-the-Middle Differential Fault Analysis on ITUbee Block Cipher

Yongze Kang [1,2], Qingyuan Yu [1,2], Lingyue Qin [3] and Guoyan Zhang [1,2,4,*]

[1] School of Cyber Science and Technology, Shandong University, Qingdao 266237, China; yzkang@mail.sdu.edu.cn (Y.K.); yuqy@mail.sdu.edu.cn (Q.Y.)
[2] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Jinan 266237, China
[3] BNRist, Tsinghua University, Beijing 100084, China; qinly@tsinghua.edu.cn
[4] Shandong Institute of Blockchain, Jinan 250014, China
* Correspondence: guoyanzhang@sdu.edu.cn

Abstract: Differential fault analysis (DFA) was introduced by Biham and Shamir. It is a powerful analysis technique to retrieve the secret key by injecting fault into an internal state and utilizing the differences between the correct ciphertexts and the faulty ciphertexts. Based on the idea of meet-in-the-middle, some differential characters can help to recover the key of some symmetric ciphers. At CHES 2011, this technique was utilized to give analyses on AES. In this article, we propose several DFA schemes on ITUbee, a software-oriented block symmetric cipher for resource-constrained devices based on the meet-in-the-middle idea. Our attacks are efficient enough and more powerful than previous works. Furthermore, the attacks in this article break the protection countermeasure, meaning we have to review the protection method on devices for ITUbee.

Keywords: differential fault attack; meet-in-the-middle; lightweight block cipher

Citation: Kang, Y.; Yu, Q.; Qin, L.; Zhang, G. Meet-in-the-Middle Differential Fault Analysis on ITUbee Block Cipher. *Symmetry* **2023**, *15*, 1196. https://doi.org/10.3390/sym15061196

Academic Editor: Xiaoyang Dong

Received: 9 May 2023
Revised: 29 May 2023
Accepted: 1 June 2023
Published: 2 June 2023

1. Introduction

With the popularization and development of computer network technology, cryptographic techniques have been widely used to ensure the confidentiality or integrity of messages and the authenticity of communication parties. However, for many resource-constrained devices such as mobile phones, public transport systems, smart cards, RFID tags, and Internet of Things devices, the majority of which employ lightweight cryptographic algorithms [1,2], these devices are convenient yet vulnerable. The sensitive information within them might be easily exposed by adversaries through side-channel analysis because of the vulnerability of physical information. Among the attacks of side-channel analysis, fault analysis (FA) is a renowned attack, first introduced by Boneh et al. [3] in 1997, which enables the attacker to obtain additional side-channel information and achieve the key recovery in practical time. At the same time, Biham and Shamir proposed a differential fault analysis (DFA) on DES [4] in 1997. This was the first time DFA was introduced to key recovery for block symmetric ciphers. Utilizing an induced error to disturb the actual implementation of the encryption process and obtain differential information between correct and faulty ciphertext pairs, DFA recovers the correct key efficiently. The key point of DFA is that it allows the adversary to analyze a small number of rounds of a block cipher. DFA has been widely applied to attacks on DES [5,6], `AES` [7–14], `PRESENT` [15–17], and others [18,19]. The countermeasures against DFA include cipher- or mode-level (e.g., `FRIET` [20], `CRAFT` [21], DEFAULT [22], and others [23–25]) and implementation-level countermeasures [26]. A widely used implementation-level countermeasure against DFA is to perform the computation twice and check whether the same result is obtained [27–30].

Proposed initially by Diffie and Hellman [31], the meet-in-the-middle strategy stands as a pivotal tool in symmetric-key primitive cryptanalysis, specifically targeting block ciphers. The effectiveness of this method is particularly showcased when an MITM attack is deliberately imposed on a block cipher designed to be susceptible to this sort of assault. This strategy's variations are well documented in numerous academic works [32–36]. Let a block symmetric cipher $E_K(\cdot)$ have an n-bit block size satisfying the condition that $C = E_K(P) = F_{K_2}(F_{K_1}(P))$, where $K = K_1||K_2$, "||" denotes the concatenation operation and K_1 and K_2 are two independent parts of the K. Consequently, both sides of the function $F_{K_1}(P) = F_{K_2}^{-1}(C)$ can be computed independently. In addition, the function will be satisfied if the guesses of K_1 and K_2 are correct. Therefore, we can reduce the search space from $2^{|K|} = 2^{|K_1|+|K_2|}$ to $2^{|K_1|+|K_2|-n}$, where $|K|$ represents the length of the key K, similarly to $|K_1|$ and $|K_2|$, and n denotes the number of advantage bits we can obtain from the structure. This approach helps us recover the secret key in lower complexity. In 2011, Derberz [12] combined this idea with differential fault analysis (DFA) to analyze the security of AES. In their work, they injected faults in the sixth round of AES and successfully recovered the key with a complexity of 2^{40} in time and memory.

Designed on the framework of the Feistel network, ITUbee is a block cipher with a software emphasis introduced by Karakoc, et al. [37]. With an 80-bit key length and block size, ITUbee incorporates a unique structure and key scheduling. It is specifically designed to cater to the constraints of resource-limited devices, such as 8-bit microcontrollers found in sensor nodes. In an earlier study, an in-depth cryptanalysis of ITUbee was carried out by Soleimany [38], disclosing a diminution of one bit in the security level of the 8-round cipher under the single-key model. Fu et al. [39] performed a DFA attack on ITUbee by injecting faults into its last two rounds, which required the injection of four faults and had a computational complexity of $2^{25.2}$.

Our Contribution. In this paper, we propose three improved DFA schemes on the ITUbee block cipher. In our schemes, the faults are injected to the third to last round of ITUbee, while, in [39], the fault was injected to the last and second to the last round. We find filters to construct differential equations in perspective of the meet-in-the-middle attack. Concretely, in our first scheme, based on observations of the cipher's round function and its structure, we carry out a DFA with 2^{48} complexity using an exhaustive search. Additionally, combining DFA with the idea of meet-in-the-middle, our next two schemes' complexity is reduced to 2^{40} and 2^{32}, respectively. Furthermore, our schemes can break the protection countermeasures proposed in [39]; thus, the security should be reviewed. Our results are summarized in Table 1.

The rest of the article is organized as follows: Section 2 gives a brief description of ITUbee and reviews the essential property of S-box S and F-function F which is useful to our schemes. In Section 3, we review the DFA proposed in the previous work. In addition, the detailed attack procedure is described in Section 4.

Table 1. Summary of our results.

	Number of Faults	Computational Complexity	Memory Complexity	Precomputational Complexity
Scheme 1	1	2^{48}	2^{40}	2^{40}
Scheme 2	4	2^{40}	2^{40}	-
Scheme 3	3	2^{32}	2^{40}	2^{40}

2. Preliminaries

2.1. Description of the Block Cipher ITUBee

ITUbee [37] is a software-oriented lightweight Feistel-like block cipher. Both the key length and block size are 80-bit. It consists of 20 rounds, and the whitening key was added before the first round and after the last round.

The definitions of the components and the notations used in the encryption and decryption procedure are as follows:

- (P_L, P_R) and (K_L, K_R) are, respectively, two 40-bit parts of the 80-bit plaintexts and the 80-bit key.
- RC_i is a 16-bit round constant which is added to the rightmost 16-bit.
- $S(a||b||c||d||e) = s[a]||s[b]||s[c]||s[d]||s[e]$, where a, b, c, d, and e are 8-bit values and S is the S-box same as that in AES [40].
- $L(a||b||c||d||e) = (e \oplus a \oplus b)||(a \oplus b \oplus c)||(b \oplus c \oplus d)||(c \oplus d \oplus e)||(d \oplus e \oplus a)$, where a, b, c, d, and e are 8-bit values.
- The F-function used in the round is defined as $F(X) = S(L(S(X)))$.

The encryption procedure is shown in Algorithm 1 and Figure 1, and the decryption procedure is the same as the encryption except for the order of the keys and the round constants.

Algorithm 1: Encryption Procedure of ITUbee ($Enc(P_L, P_R, K_L, K_R)$)

Initialize $X_1 \leftarrow P_L \oplus K_L$ and $X_0 \leftarrow P_R \oplus K_R$
for *i = 1 to 20* **do**
  if $i \ mod \ 2 = 1$ **then**
    $RK \leftarrow K_R$
  else
    $RK \leftarrow K_L$
  end
  $X_{i+1} \leftarrow X_{i-1} \oplus F(L(RK \oplus RC_i \oplus F(X_i)))$
end
$C_L \leftarrow X_{20} \oplus K_R$ and $C_R \leftarrow X_{21} \oplus K_L$
Output (C_L, C_R)

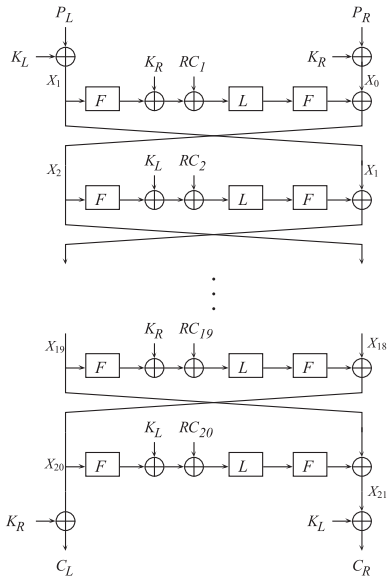

Figure 1. Schematic diagram of ITUbee.

For more details, please refer to [37].

2.2. Obervations on ITUBee Block Cipher

In this section, we review several observations of ITUbee, which are the bases of our schemes.

2.2.1. Differential Property of S-Box

Let $S(\cdot)$ be the S-box used in ITUbee, and consider the following equation:

$$S(x) \oplus S(x \oplus \alpha) = \beta \tag{1}$$

With α and β standing for input and output differentials, respectively, Equation (1) can be tackled by examining all conceivable x values. Based on the specific α and β, the solution counts to Equation (1) can only be 0, 2, or 4. To elaborate, the likelihoods stand at $\frac{129}{256}$ for 0, $\frac{126}{256}$ for 2, and $\frac{1}{256}$ for 4. Typically, for a randomly chosen input–output differential pair, one would expect to locate 1 x solution. A lookup table structured around α and β indices could simplify this process.

2.2.2. Property of F-Function

Consider a 40-bit input differential exhibiting a singular nonzero byte; the resulting output differential from $F(\cdot)$ will consistently incorporate three nonzero bytes. Given the input differential α and output differential β, on average, there exists a single solution for the equation $F(x) \oplus F(x \oplus \alpha) = \beta$. To streamline the equation-solving process, it is practical to construct a lookup table, with α and β as guiding indices, bringing computational complexity down to 1. However, the prerequisite calculation for this lookup table necessitates nearly 2^{40} iterations of $F(\cdot)$ operations along with a matching memory requirement.

3. Previous DFA Scheme

In [39], Fu et al. proposed two fault injection schemes on ITUbee and suggested countermeasures to protect the encryption devices. In this section, we review their analysis schemes and countermeasures, and we will prove their countermeasures are not strong enough by proposing our improved DFA schemes in Section 4.

3.1. Fault Model

Fu et al.'s DFA on ITUbee is based on the byte-oriented model, which has the following assumptions:

- The adversary can inject a byte fault to a selected state of the block cipher; for example, the adversary could inject a random byte fault to the output of the last $S(\cdot)$ in the last but two rounds.
- The location of the fault in the word is known to the adversary. Moreover, the case of unknown location of injected fault is also discussed.
- The adversary could obtain ciphertexts of both correct and faulty execution.

3.2. DFA Schemes

3.2.1. Attack 1

Randomly choose a plaintext P and ask for the corresponding ciphertext C, inject a random byte fault to a certain position of the second $S(\cdot)$ layer of the last round, and obtain the faulty ciphertext C^*. For every possible difference generated from the injection, compute the input difference and output difference of the last $S(\cdot)$ operation in the last round and filter the values of input and output states. For the remaining candidates, compute the pairs backwards and filter with the injection position. If there is more than one candidate remaining, repeat the steps and recover the secret key.

3.2.2. Scheme 2

A plaintext P is picked at random and its corresponding ciphertext C is computed using the unknown key. Subsequently, a random byte fault is inserted at a designated spot within the output of the final $S(\cdot)$ layer of the round preceding the last, leading to a faulty ciphertext referred to as C^*. For the input difference and output difference of the second F-function of the last round, deduce the possible values of the internal state before and after the F-function according to the property in Section 2.2.2. Further filter the candidates

according to the differential in the last round. If there is more than one candidate remaining, repeat the steps and recover the secret key.

3.3. Countermeasures

Considering the efficiency, Fu et al. propose to protect the encryption devices by running the double-check mechanism in the last two rounds. However, if we can achieve key recovery attacks by injecting fault before the last two rounds, the countermeasure is invalid.

4. Improved DFA Schemes on ITUbee

In this section, DFA on ITUbee is described in detail. Under the same assumptions as defined in Section 3.1, we give three schemes whose fault is injected before the last two rounds. For a better understanding of our method, we first introduce some notations used in the section. X_i denotes the input state of round i, and C_L and C_R are the ciphertexts. For each internal state noted in Figure 2, for example, e_0, use $e_0[i]$ to denote the i-th byte of the state. $e_0^*[i]$ denotes the corresponding byte of the state with fault, and $\Delta e_0^*[i]$ is the difference between the correct and fault value of the i-th byte of the state.

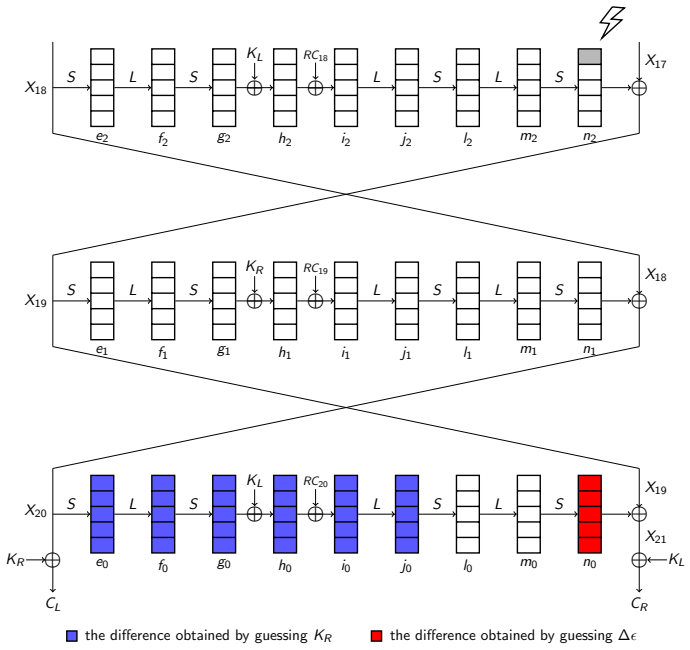

Figure 2. Fault injection Scheme 1.

4.1. Scheme 1: Differential Fault Attack with Exhaustive Search

We assume that the fault $\Delta\epsilon$ is injected at a selected byte of state n_2. The major steps of the attack are as follows:

Step 1. Obtain the correct and faulty ciphertexts. Initiate by randomly picking a plaintext P, and calculate the equivalent ciphertext C with the undisclosed key. As presented in Figure 2, infuse a random byte fault $\Delta\epsilon$ into the state n_2 to procure the erroneous ciphertext C^*.

Step 2. Deduce the difference of the internal state. For each 8-bit value in $\Delta\epsilon$, the corresponding difference in Δn_0 can be determined in reverse order. Note that

$$\Delta n_0 = n_0 \oplus n_0^* = (X_{19} \oplus K_L \oplus C_R) \oplus (X_{19} \oplus K_L \oplus C_R^* \oplus \Delta\epsilon) = C_R \oplus C_R^* \oplus \Delta\epsilon.$$

Further, using 40-bit K_R, we can forward compute the corresponding difference of Δj_0, namely,

$$\Delta j_0 = L(\Delta i_0) = L(\Delta g_0) = L(F(e_0) \oplus F(e_0^*)) = L(F(K_R \oplus C_L) \oplus F(K_R \oplus C_L^*)).$$

$\Delta n_0, \Delta j_0$ can be deduced from the ciphertexts, which means both the differences before and after the last $F(\cdot)$ operation are known. According to the observations in Section 2.2.2,

for every fixed input difference and output difference pair of $F(\cdot)$, we can obtain one solution that matches the input and output difference pair on average. Thus, for each of the 2^{48} possible values of the pair of Δn_0 and Δj_0, there is one corresponding value of n_0 and j_0 that exists on average.

Step 3. Exhaustively search to recover the whole key. Looking up the table storing the values indexed by input and output difference of $F(\cdot)$, we obtain 2^{48} possible values of (n_0, j_0). Computing $K_L = n_0 \oplus C_R$ and $K_R = F^{-1}(L^{-1}(j_0)) \oplus C_L$, we obtain 2^{48} possible keys of (K_L, K_R). Exhaustively search for all the possible values and recover the whole key.

Complexity. As $F(\cdot)$ consists of $L(\cdot)$ and $S(\cdot)$ operations only, it is a 40-bit permutation. This kind of permutation can be viewed as a Super S-box [41]. To build a look-up table indexed by input difference α and output difference β, we need 2^{40} precomputation time complexity and 2^{40} bits of memory. To recover the whole key, we need 2^{48} forward and backward computing operations and 2^{48} decryption time complexity.

4.2. Scheme 2: Meet-in-the-Middle Fault Attack with 2^{40} Complexity

Assume that a random fault $\Delta\epsilon$ is injected at a selected byte of state n_2. Without loss of generality, we assume that $n_2[0]$ is the position where the fault is injected. Note that

$$j_0 = F^{-1}(n_0) = F^{-1}(X_{19} \oplus K_L \oplus C_R)$$

The corrupted value of the internal state j_0 can be obtained in the same way. According to the property of $F(\cdot)$, we can always compute two bytes in the input of $F(\cdot)$, though one byte in the output is unknown. For example, if $n_0[1,2,3,4]$ is known, only the 0-th byte of n_0 is unknown, so we have

$$j_0[1] = F^{-1}(n_0[1,3,4]) = F^{-1}(X_{19}[1,3,4] \oplus K_L[1,3,4] \oplus C_R[1,3,4]) \tag{2}$$

$$j_0[4] = F^{-1}(n_0[1,2,4]) = F^{-1}(X_{19}[1,2,4] \oplus K_L[1,2,4] \oplus C_R[1,2,4]) \tag{3}$$

Utilizing this property, and noting that $X_{19}[1,2,3,4] = X^*_{19}[1,2,3,4]$, $X_{19}[0] \neq X^*_{19}[0]$, we can obtain the following equation:

$$\begin{aligned} \Delta j_0[1] &= F^{-1}(n_0[1,3,4]) \oplus F^{-1}(n^*_0[1,3,4]) \\ &= F^{-1}(X_{19}[1,3,4] \oplus K_L[1,3,4] \oplus C_R[1,3,4]) \oplus F^{-1}(X_{19}[1,3,4] \oplus K_L[1,3,4] \oplus C^*_R[1,3,4]) \end{aligned} \tag{4}$$

Furthermore, $\Delta j_0[4]$ can be computed in the same way. Moreover, we can obtain the state Δj_0 for all possible values of K_R in such a computational path:

$$\Delta j_0 = L(\Delta i_0) = L(\Delta g_0) = L(F(K_R \oplus C_L) \oplus F(K_R \oplus C^*_L)) \tag{5}$$

With the obtained C and C^*, as we can see, $\Delta j_0[1,4]$ can be obtained in two computational paths, meaning each computing direction involving several uncorrelated key bytes. Thus, we can carry out an MITM attack. The major steps of an attack making use of $\Delta j_0[1]$ are shown below:

Step 1. Obtain the correct and faulty ciphertexts with the same plaintext. Randomly choose a plaintext P and obtain the corresponding ciphertext C under the unknown key. As depicted in Figure 3, inject a random difference $\Delta\epsilon$ to the 0-th byte of the state n_2 and ask for the corresponding corrupt ciphertext four times. Then, we obtain four pairs of different correct and faulty ciphertext pairs $(C, (C^*)_i)$ and six pairs of faulty ciphertext pairs $((C^*)_i, (C^*)_j)$, where $0 \leq i, j \leq 3, i \neq j$. Under the correct key guess, these 10 pairs that satisfy the values in $X_{19}[1,2,3,4]$ are the same, only the values in $X_{19}[0]$ have differences.

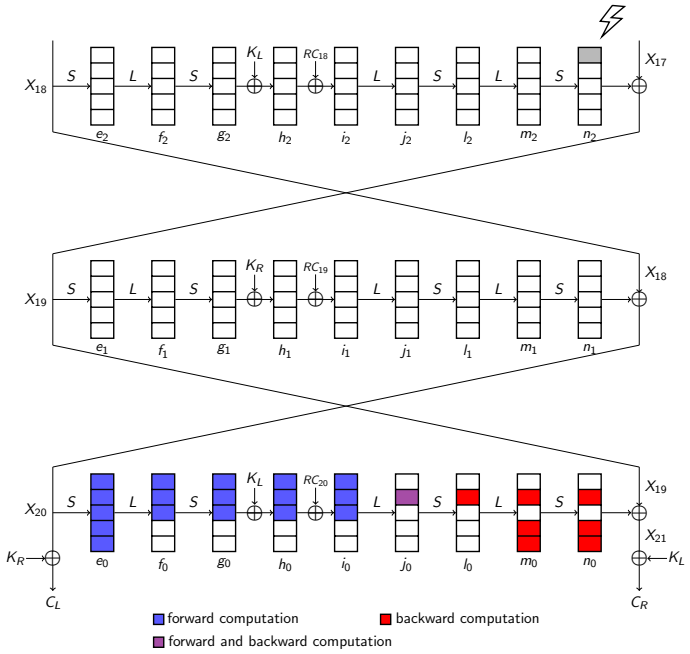

Figure 3. Fault injection Scheme 2.

Step 2. Compute $\Delta j_0[1]$ in two computational directions and filter them. To obtain a valid candidate, execute the following:

1. For each pair, guess all 2^{24} candidates of $X_{19} \oplus K_L[1,3,4]$, compute the values of $\Delta j_0[1]$ according to Equation (4), store the value of $\Delta j_0[1]$ for each $X_{19} \oplus K_L[1,3,4]$.
2. For all 2^{40} candidates of K_R, forward compute the value of $\Delta j_0[1]$ for each pair, store the values of $\Delta j_0[1]$ indexed by K_R.
3. For each pair, if the value $\Delta j_0[1]$ of two computational directions is equal, mark the K_R and $X_{19} \oplus K_L[1,3,4]$ as a valid candidate. This phase can be seen as a 2^{-8} filter, with eight pairs filtered, the number of candidates of $(K_R, X_{19} \oplus K_L[1,3,4])$ decreased to close to 1.

Step 3. Recover the correct key. We consider the computation procedure without injected fault, namely, (P, C). With K_R filtered by the above process, we can compute the state g_0. Moreover, for every 2^{16} of possible values of $X_{19} \oplus K_L[0,2]$, we can compute all possible states h_0 with the $X_{19} \oplus K_L[1,3,4]$ obtained before. Finally, we can deduce all possible K_L to recover and validate the whole key using exhaustive searching.

$$K_L = g_0 \oplus h_0 = F(K_R \oplus C_L) \oplus \left(RC_{20} \oplus L^{-1}\left(F^{-1}(C_R \oplus K_L \oplus X_{19})\right)\right) \tag{6}$$

The above is the attack procedure when the fault is injected in the state $n_2[0]$. Likewise, for other certain locations of the injected fault, we can carry out an MITM attack in a similar way. However, if the location of the fault is unknown, we can also carry out three MITM attacks on any given three bytes on Δj_0, respectively. Because of the Pigeonhole Principle, there exists at least one byte whose computation is unaffected by the fault. As a result, in this case, we need three times the computational complexity of the known location's case.

Complexity. For a known injected location's fault, we have only one MITM episode, and, in each episode, 2^{40+24} different elements in $\{K_R, X_{19} \oplus K_L[1,3,4]\}$ are tested. Only the corrected one will pass the filter in theory. Next, we exhaustively search the corresponding key for every possible value of $X_{19} \oplus K_L[0,2]$ which costs 2^{16} in complexity. Therefore, the overall time complexity can be estimated as $2^{40} + 2^{24} + 2^{16}$ times the encryption or decryption, which is approximately 2^{40}. In addition, we need 2^{40} memory complexity to store the candidates. However, if the injected location is unknown, we suppose that the same byte is injected by different faults. For consequences, the complexity will be multiplied by three.

4.3. Scheme 3: Meet-in-the-Middle Attack with 2^{32} Complexity

Assume that the fault $\Delta\epsilon$ is injected at a selected byte of state n_2. Without loss of generality, we assume that $n_2[0]$ is the position where the fault is injected. Consequently, the difference is not totally diffused in state i_1, namely, $\Delta i_1[2,3] = 0$. Note that

$$\Delta j_1[0] \oplus \Delta j_1[2] \oplus \Delta j_1[4] = \Delta\Big(L^{-1}(j_1[0,2,4])\Big) = \Delta i_1[2] = 0. \tag{7}$$

Thus, we have $\Delta j_1[0] \oplus \Delta j_1[2] = \Delta j_1[4]$, and

$$\begin{aligned} &F^{-1}((K_R \oplus X_{18})[0,2,3,4] \oplus C_L[0,2,3,4]) \oplus F^{-1}((K_R \oplus X_{18})[0,2,3,4] \oplus C_L^*[0,2,3,4]) \\ &\quad = F^{-1}((K_R \oplus X_{18})[1,2,4] \oplus C_L[1,2,4]) \oplus F^{-1}((K_R \oplus X_{18})[1,2,4] \oplus C_L^*[1,2,4]). \end{aligned} \tag{8}$$

As we can see, $K_R \oplus X_{18}$ can be computed in two parts, respectively, in Equation (8) so that we can carry out an MITM attack as described in the following.

Step 1. Obtain the correct and faulty ciphertexts with the same plaintext. Randomly choose a plaintext P and obtain the corresponding ciphertext C under the unknown key. As shown in Figure 4, inject at random byte faults $\Delta\epsilon$ to a certain byte of the state n_2 in the last two rounds and store the faulty ciphertext. Inject different faults three times, so we can obtain six ciphertext pairs, which contain three pairs of correct and different faulty ciphertext $(C, (C^*)_i)$ and three pairs of faulty and faulty ciphertext $((C^*)_i, (C^*)_j)$, where $0 \le i, j \le 2$ and $i \ne j$.

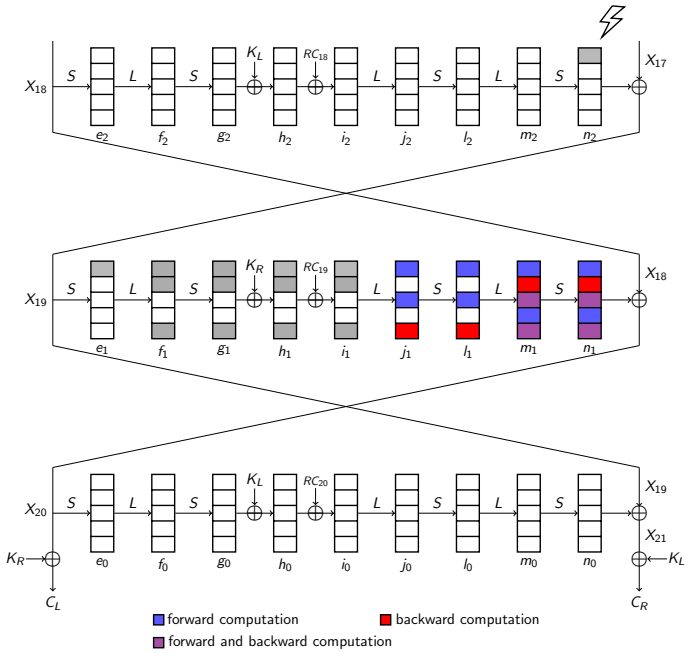

Figure 4. Fault injection Scheme 3.

Step 2. Compute the value for all candidates and filter them. For all 2^{16} candidates of $K_R \oplus X_{18}[2,4]$, perform the following operations:

1. Compute the values of $\Delta j_1[0] \oplus \Delta j_1[2]$ according to the left side of Equation (8) for all candidates of $K_R \oplus X_{18}[0,3]$ for each that we obtained before, and store the values of $K_R \oplus X_{18}[0,3]$ indexed by the vector of $\Delta j_1[0] \oplus \Delta j_1[2]$ of five pairs in the six in a table.
2. Compute the vector $\Delta j_1[4]$ following the right side of Equation (8) for all candidates of $K_R \oplus X_{18}[1]$ of the five pairs we obtained before, and store the values in another table.
3. We sort the two tables and find collisions for the index values. If there is a collision between the two tables, the corresponding $K_R \oplus X_{18}$ is stored as a valid candidate. With the five-pair filter, there will exist just one candidate in theory.

Step 3. Recover the remaining key bits. We consider the computational procedure of one correct and faulty pair $\{(P, C), (P, C^*)\}$. With $K_R \oplus X_{18}$ filtered by the above process, we can compute the state h_1 and the difference Δg_1. Moreover, for every 2^8 possible difference $\Delta\epsilon$, which is the input difference of the first function $F(\cdot)$ in the last but one round, all possible states g_1 can be deduced with the property of function $F(\cdot)$. Consequently, we can recover the possible K_R and the state X_{20}, g_0. Similarly, we can also obtain the difference Δj_0 and Δn_0, which is the input and output difference of the last function in the last round. Likewise, we can obtain the state j_0 to recover the state h_0 and the key K_L. Finally, we can obtain all possible keys and validate the correct key using an exhaustive search.

The above is the attack procedure when the fault is injected in the state $n_2[0]$. Likewise, for every known location of the injected fault, we can carry out an MITM attack in a similar way. However, if the location of the fault is unknown, we can also reanalyze a byte in state i_1 where the fault has not been affected and carry out MITM attacks. Because the fault can be injected into any one byte of the total five bytes in state n_2, we need five times the computational complexity of the known location's case.

Complexity. For a known injected location's fault, we have 2^{16} MITM episodes for $K_R \oplus X_{18}[2, 4]$, and, in each episode, elements in $K_R \oplus X_{18}[0, 1, 3]$ are tested. Only one element will pass the filter with a high probability. Next, we calculate the corresponding key for every possible value of $\Delta\epsilon$ which costs 2^8 in complexity. Therefore, the overall time complexity can be estimated as $2^{16}(2^{16} + 2^8) + 2^8$, which is approximately 2^{32}; meanwhile, 2^{40} precomputational complexity and 2^{40} memory complexity for storing the values and difference of the F-function is needed. If the injected location is unknown, we suppose that the same byte is injected by different faults. As a consequence, the computational complexity will be multiplied by five.

4.4. Simulation Results

In this section, we give some simulation results of our schemes. As Scheme 1 requires too much time and memory, only the simulations of Schemes 2 and 3 are given. We implemented our schemes on a PC with an Intel Core i7 processor whose frequency is 2.5 GHz. In the simulation of Schemes 2 and 3, respectively, 500 and 1000 samples were recorded with randomly selected keys.

Scheme 2. Owing to the huge computational complexity of Scheme 2, we chose 500 samples simulated in Figure 5, where the y-axis represents the number of key candidates filtered after Step 2 in Scheme 2 and the x-axis represents the sample number. A data point with a value above 1 on the y-axis indicates that the key candidates for that sample have not been completely filtered. In that case, we can filter one more time using the remaining pairs, or we have to endure extra computational complexity in Step 3. As depicted in Figure 5, with the injection of 4 faults, the average number of remaining key candidates is 1.546. This result confirms that 4 faults are sufficient for Scheme 2 to be effective.

Scheme 3. As shown in Figure 5, we simulated Scheme 3 with 1000 samples and collected the number of candidates filtered after Step 2 in Scheme 3. Similarly, we could still filter again in the case that the number of candidates was over one. Moreover, in Step 3, the exhaustive search has a really low probability of causing a situation in which we cannot recover the correct key due to the property of function F. A simple solution to this problem is just changing the correct and faulty pair used in Step 3. When injecting 3 faults, the average number of remaining key candidates is 1.397. This result provides evidence that 3 faults are sufficient for the effectiveness of Scheme 3.

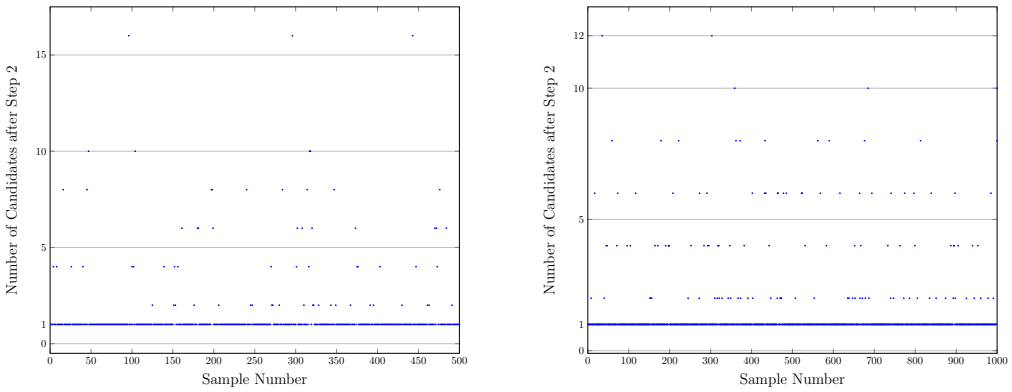

Figure 5. Key candidates after Step 2 in Scheme 2 (**left**) and after Step 2 in Scheme 3 (**right**).

4.5. Further Countermeasures

In [39], the authors proposed a countermeasure based on the double-check mechanism. For efficiency, they ran the crucial operation twice to check if the two executions matched each other. However, they only ran the last two rounds twice according to their analysis. The countermeasure is invalid for our attack scheme, so we suggest extending the number of rounds involved in the double-check mechanism by at least one. Namely, run the ITUbee as follows in Algorithm 2:

Algorithm 2: ITUbee-Cipher (Message, Key)

$State1 \leftarrow Round1 - 17(Message, Key)$
$State2 \leftarrow State1$
$Res1 \leftarrow Round18 - 20(State1, Key)$
$Res2 \leftarrow Round18 - 20(State2, Key)$
$RandomDelay()$
if $Res1 = Res2$ **then**
  $Output(Res1)$
else
  $Reset()$
end

Thus, the countermeasure could protect the devices from our attacks. The random delay was introduced to avoid the adversary injecting twice in one execution to run a successful attack.

4.6. Discussions

In this section, we present a comparison between the work conducted by Fu et al. [39] and the work conducted by ourselves on the DFA of the same ITUbee algorithm. Fu et al. induced single-byte random faults on the state located in the second to the last round, and they suggested implementing ITUbee with a double-check mechanism in the last two rounds to go against fault analysis. In this paper, we proposed three schemes to achieve DFA attacks, with faults induced at the third to the last round. Our attacks prove that their countermeasures do not protect against all DFA attacks. These distinctions are comprehensively summarized in Table 2 .

Table 2. The comparisons with Fu et al.'s work.

	Number of Faults	Faults Injection Round	Computational Complexity	Memory Complexity	Precomputational Complexity
Our Scheme 3	3	third to last	2^{32}	2^{40}	2^{40}
Fu et al.'s Scheme 2	4	second to last	$2^{25.2}$	2^{40}	2^{40}

5. Conclusions

This article presents several differential fault analyses on ITUbee based on the meet-in-the-middle idea. Our attacks make use of the property of faulty values and differences in faulty and correct intermediate values. Our attack schemes combine the differential fault analysis and meet-in-the-middle methods, which can also be extended to other block ciphers. In addition, we overrode the security of the countermeasures given in previous works and revisited the protection schemes for ITUbee block cipher on devices.

Author Contributions: Conceptualization, Y.K. and Q.Y.; validation, Y.K.; formal analysis, Q.Y. and L.Q.; writing—original draft preparation, Y.K. and Q.Y.; writing—review and editing, Q.Y. and L.Q.; supervision, G.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data is contained within the article.

Acknowledgments: The authors would like to thank the anonymous reviewers for their helpful comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zakaria, A.A.; Halim, A.H.A.; Ridzuan, F.; Zakaria, N.H.; Daud, M. LAO-3D: A Symmetric Lightweight Block Cipher Based on 3D Permutation for Mobile Encryption Application. *Symmetry* **2022**, *14*, 2042. [CrossRef]
2. Alshammari, B.; Guesmi, R.; Guesmi, T.; Alsaif, H.; Alzamil, A. Implementing a Symmetric Lightweight Cryptosystem in Highly Constrained IoT Devices by Using a Chaotic S-Box. *Symmetry* **2021**, *13*, 129. [CrossRef]
3. Aumüller, C.; Bier, P.; Fischer, W.; Hofreiter, P.; Seifert, J. Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures. In *Proceedings of the Cryptographic Hardware and Embedded Systems—CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, 13–15 August 2002*; Revised Paper; Lecture Notes in Computer Science; Kaliski, B.S., Jr., Koç, Ç.K., Paar, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2523, pp. 260–275. [CrossRef]
4. Biham, E.; Shamir, A. Differential Fault Analysis of Secret Key Cryptosystems. In *Proceedings of the Advances in Cryptology—CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, CA, USA, 17–21 August 1997*; Proceedings; Lecture Notes in Computer Science; Kaliski, B.S., Jr., Ed.; Springer: Berlin/Heidelberg, Germany, 1997; Volume 1294, pp. 513–525. [CrossRef]
5. Hemme, L. A Differential Fault Attack against Early Rounds of (Triple-)DES. In *Proceedings of the Cryptographic Hardware and Embedded Systems—CHES 2004: 6th International Workshop Cambridge, MA, USA, 11–13 August 2004*; Proceedings; Lecture Notes in Computer Science; Joye, M., Quisquater, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3156, pp. 254–267. [CrossRef]
6. Rivain, M. Differential Fault Analysis on DES Middle Rounds. In *Proceedings of the Cryptographic Hardware and Embedded Systems—CHES 2009, 11th International Workshop, Lausanne, Switzerland, 6–9 September 2009*; Proceedings; Lecture Notes in Computer Science; Clavier, C., Gaj, K., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5747, pp. 457–469. [CrossRef]
7. Piret, G.; Quisquater, J. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In *Proceedings of the Cryptographic Hardware and Embedded Systems—CHES 2003, 5th International Workshop, Cologne, Germany, 8–10 September 2003*; Proceedings; Lecture Notes in Computer Science; Walter, C.D., Koç, Ç.K., Paar, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2779, pp. 77–88. [CrossRef]
8. Dusart, P.; Letourneux, G.; Vivolo, O. Differential Fault Analysis on A.E.S. In *Proceedings of the Applied Cryptography and Network Security, First International Conference, ACNS 2003, Kunming, China, 16–19 October 2003*; Proceedings; Lecture Notes in Computer Science; Zhou, J., Yung, M., Han, Y., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2846, pp. 293–306. [CrossRef]
9. Blömer, J.; Seifert, J. Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). In *Proceedings of the Financial Cryptography, 7th International Conference, FC 2003, Guadeloupe, French West Indies, 27–30 January 2003*; Revised Papers; Lecture Notes in Computer Science; Wright, R.N., Ed.; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2742, pp. 162–181. [CrossRef]
10. Giraud, C. DFA on AES. In *Proceedings of the Advanced Encryption Standard—AES, 4th International Conference, AES 2004, Bonn, Germany, 10–12 May 2004*; Revised Selected and Invited Papers; Lecture Notes in Computer Science; Dobbertin, H., Rijmen, V., Sowa, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3373, pp. 27–41. [CrossRef]
11. Moradi, A.; Shalmani, M.T.M.; Salmasizadeh, M. A Generalized Method of Differential Fault Attack Against AES Cryptosystem. In *Proceedings of the Cryptographic Hardware and Embedded Systems—CHES 2006, 8th International Workshop, Yokohama, Japan, 10–13 October 2006*; Proceedings; Lecture Notes in Computer Science; Goubin, L., Matsui, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4249, pp. 91–100. [CrossRef]

12. Derbez, P.; Fouque, P.; Leresteux, D. Meet-in-the-Middle and Impossible Differential Fault Analysis on AES. In *Proceedings of the Cryptographic Hardware and Embedded Systems—CHES 2011—13th International Workshop, Nara, Japan, 28 September–1 October 2011*; Proceedings; Lecture Notes in Computer Science; Preneel, B., Takagi, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6917, pp. 274–291. [CrossRef]
13. Yuce, B.; Schaumont, P.; Witteman, M. Fault Attacks on Secure Embedded Software: Threats, Design, and Evaluation. *J. Hardw. Syst. Secur.* **2018**, *2*, 111–130. [CrossRef]
14. Selmke, B.; Heyszl, J.; Sigl, G. Attack on a DFA Protected AES by Simultaneous Laser Fault Injections. In *Proceedings of the 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, 16 August 2016*; IEEE Computer Society: Piscataway, NJ, USA, 2016; pp. 36–46. [CrossRef]
15. Santis, F.D.; Guillen, O.M.; Sakic, E.; Sigl, G. Ciphertext-Only Fault Attacks on PRESENT. In *Proceedings of the Lightweight Cryptography for Security and Privacy—Third International Workshop, LightSec 2014, Istanbul, Turkey, 1–2 September 2014*; Revised Selected Papers; Lecture Notes in Computer Science; Eisenbarth, T., Öztürk, E., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8898, pp. 85–108. [CrossRef]
16. Ghalaty, N.F.; Yuce, B.; Schaumont, P. Differential Fault Intensity Analysis on PRESENT and LED Block Ciphers. In *Proceedings of the Constructive Side-Channel Analysis and Secure Design—6th International Workshop, COSADE 2015, Berlin, Germany, 13–14 April 2015*; Revised Selected Papers; Lecture Notes in Computer Science; Mangard, S., Poschmann, A.Y., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9064, pp. 174–188. [CrossRef]
17. Patranabis, S.; Breier, J.; Mukhopadhyay, D.; Bhasin, S. One Plus One is More than Two: A Practical Combination of Power and Fault Analysis Attacks on PRESENT and PRESENT-Like Block Ciphers. In *Proceedings of the 2017 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2017, Taipei, Taiwan, 25 September 2017*; IEEE Computer Society: Piscataway, NJ, USA, 2017; pp. 25–32. [CrossRef]
18. Fukunaga, T.; Takahashi, J. Practical Fault Attack on a Cryptographic LSI with ISO/IEC 18033-3 Block Ciphers. In *Proceedings of the Sixth International Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2009, Lausanne, Switzerland, 6 September 2009*; Breveglieri, L., Koren, I., Naccache, D., Oswald, E., Seifert, J., Eds.; IEEE Computer Society: Piscataway, NJ, USA, 2009; pp. 84–92. [CrossRef]
19. Tupsamudre, H.; Bisht, S.; Mukhopadhyay, D. Differential Fault Analysis on the Families of SIMON and SPECK Ciphers. In *Proceedings of the 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2014, Busan, Republic of Korea Korea, 23 September 2014*; Tria, A., Choi, D., Eds.; IEEE Computer Society: Piscataway, NJ, USA, 2014; pp. 40–48. [CrossRef]
20. Simon, T.; Batina, L.; Daemen, J.; Grosso, V.; Massolino, P.M.C.; Papagiannopoulos, K.; Regazzoni, F.; Samwel, N. Friet: An Authenticated Encryption Scheme with Built-in Fault Detection. In *Proceedings of the Advances in Cryptology—EUROCRYPT 2020—39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, 10–14 May 2020*; Proceedings, Part I; Lecture Notes in Computer Science; Canteaut, A., Ishai, Y., Eds.; Springer: Berlin/Heidelberg, Germany, 2020; Volume 12105, pp. 581–611. [CrossRef]
21. Beierle, C.; Leander, G.; Moradi, A.; Rasoolzadeh, S. CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks. *IACR Trans. Symmetric Cryptol.* **2019**, *2019*, 5–45. [CrossRef]
22. Baksi, A.; Bhasin, S.; Breier, J.; Khairallah, M.; Peyrin, T.; Sarkar, S.; Sim, S.M. DEFAULT: Cipher Level Resistance against Differential Fault Attack. In *Proceedings of the Advances in Cryptology—ASIACRYPT 2021—27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, 6–10 December 2021*; Proceedings, Part II; Lecture Notes in Computer Science; Tibouchi, M., Wang, H., Eds.; Springer: Berlin/Heidelberg, Germany, 2021; Volume 13091, pp. 124–156. [CrossRef]
23. Medwed, M.; Standaert, F.; Großschädl, J.; Regazzoni, F. Fresh Re-keying: Security against Side-Channel and Fault Attacks for Low-Cost Devices. In *Proceedings of the Progress in Cryptology—AFRICACRYPT 2010, Third International Conference on Cryptology in Africa, Stellenbosch, South Africa, 3–6 May 2010*; Proceedings; Lecture Notes in Computer Science; Bernstein, D.J., Lange, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6055, pp. 279–296. [CrossRef]
24. Medwed, M.; Petit, C.; Regazzoni, F.; Renauld, M.; Standaert, F. Fresh Re-keying II: Securing Multiple Parties against Side-Channel and Fault Attacks. In *Proceedings of the Smart Card Research and Advanced Applications—10th IFIP WG 8.8/11.2 International Conference, CARDIS 2011, Leuven, Belgium, 14–16 September 2011*; Revised Selected Papers; Lecture Notes in Computer Science; Prouff, E., Ed.; Springer: Berlin/Heidelberg, Germany, 2011; Volume 7079, pp. 115–132. [CrossRef]
25. Dobraunig, C.; Koeune, F.; Mangard, S.; Mendel, F.; Standaert, F. Towards Fresh and Hybrid Re-Keying Schemes with Beyond Birthday Security. In *Proceedings of the Smart Card Research and Advanced Applications—14th International Conference, CARDIS 2015, Bochum, Germany, 4–6 November 2015*; Revised Selected Papers; Lecture Notes in Computer Science; Homma, N., Medwed, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9514, pp. 225–241. [CrossRef]
26. Lomné, 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, Leuven, Belgium, 9 September 2012*; Bertoni, G., Gierlichs, B., Eds.; IEEE Computer Society: Piscataway, NJ, USA, 2012; pp. 85–94. [CrossRef]
27. Malkin, T.; Standaert, F.; Yung, M. A Comparative Cost/Security Analysis of Fault Attack Countermeasures. In *Proceedings of the Fault Diagnosis and Tolerance in Cryptography, Third International Workshop, FDTC 2006, Yokohama, Japan, 10 October 2006*; Proceedings; Lecture Notes in Computer Science; Breveglieri, L., Koren, I., Naccache, D., Seifert, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4236, pp. 159–172. [CrossRef]

28. Maistri, P.; Leveugle, R. Double-Data-Rate Computation as a Countermeasure against Fault Analysis. *IEEE Trans. Comput.* **2008**, *57*, 1528–1539. [CrossRef]
29. Joye, M.; Manet, P.; Rigaud, J. Strengthening hardware AES implementations against fault attacks. *IET Inf. Secur.* **2007**, *1*, 106–110. [CrossRef]
30. Barenghi, A.; Breveglieri, L.; Koren, I.; Pelosi, G.; Regazzoni, F. Countermeasures against fault attacks on software implemented AES: Effectiveness and cost. In Proceedings of the 5th Workshop on Embedded Systems Security, WESS 2010, Scottsdale, AZ, USA, 24 October 2010; ACM: New York, NY, USA, 2010; p. 7. [CrossRef]
31. Diffie, W.; Hellman, M.E. Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *Computer* **1977**, *10*, 74–84. [CrossRef]
32. Dong, X.; Hua, J.; Sun, S.; Li, Z.; Wang, X.; Hu, L. Meet-in-the-Middle Attacks Revisited: Key-Recovery, Collision, and Preimage Attacks. In *Proceedings of the Advances in Cryptology—CRYPTO 2021—41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, 16–20 August 2021*; Proceedings, Part III; Lecture Notes in Computer Science; Malkin, T., Peikert, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2021; Volume 12827, pp. 278–308. [CrossRef]
33. Hua, J.; Dong, X.; Sun, S.; Zhang, Z.; Hu, L.; Wang, X. Improved MITM Cryptanalysis on Streebog. *IACR Trans. Symmetric Cryptol.* **2022**, *2022*, 63–91. [CrossRef]
34. Dinur, I.; Dunkelman, O.; Keller, N.; Shamir, A. New Attacks on Feistel Structures with Improved Memory Complexities. In *Proceedings of the Advances in Cryptology—CRYPTO 2015—35th Annual Cryptology Conference, Santa Barbara, CA, USA, 16–20 August 2015*; Proceedings, Part I; Lecture Notes in Computer Science; Gennaro, R., Robshaw, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9215, pp. 433–454. [CrossRef]
35. Dinur, I.; Dunkelman, O.; Keller, N.; Shamir, A. Cryptanalysis of Iterated Even-Mansour Schemes with Two Keys. In *Proceedings of the Advances in Cryptology—ASIACRYPT 2014—20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, 7–11 December 2014*; Proceedings, Part I; Lecture Notes in Computer Science; Sarkar, P., Iwata, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8873, pp. 439–457. [CrossRef]
36. Dunkelman, O.; Sekar, G.; Preneel, B. Improved Meet-in-the-Middle Attacks on Reduced-Round DES. In *Proceedings of the Progress in Cryptology—INDOCRYPT 2007, 8th International Conference on Cryptology in India, Chennai, India, 9–13 December 2007*; Proceedings; Lecture Notes in Computer Science; Srinathan, K., Rangan, C.P., Yung, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4859, pp. 86–100. [CrossRef]
37. Karakoç, F.; Demirci, H.; Harmanci, A.E. ITUbee: A Software Oriented Lightweight Block Cipher. In *Proceedings of the Lightweight Cryptography for Security and Privacy—Second International Workshop, LightSec 2013, Gebze, Turkey, 6–7 May 2013*; Revised Selected Papers; Lecture Notes in Computer Science; Avoine, G., Kara, O., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; Volume 8162, pp. 16–27. [CrossRef]
38. Soleimany, H. Self-similarity cryptanalysis of the block cipher ITUbee. *IET Inf. Secur.* **2015**, *9*, 179–184. [CrossRef]
39. Fu, S.; Xu, G.; Pan, J.; Wang, Z.; Wang, A. Differential Fault Attack on ITUbee Block Cipher. *ACM Trans. Embed. Comput. Syst.* **2016**, *16*, 1–10. [CrossRef]
40. Daemen, J.; Rijmen, V. *The Design of Rijndael: AES—The Advanced Encryption Standard*; Information Security and Cryptography; Springer: Berlin/Heidelberg, Germany, 2002. [CrossRef]
41. Gilbert, H.; Peyrin, T. Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In *Proceedings of the Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Republic of Korea, 7–10 February 2010*; Revised Selected Papers; Lecture Notes in Computer Science; Hong, S., Iwata, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6147, pp. 365–383. [CrossRef]