More Vulnerabilities of Linear Structure Sbox-Based Ciphers Reveal Their Inability to Protect DFA

Amit Jana, Anup Kumar Kundu, and Goutam Paul

Cryptology and Security Research Unit,
R. C. Bose Centre for Cryptology and Security
Indian Statistical Institute, Kolkata,
203, Barrackpore Trunk Road Kolkata 700108, INDIA.
janaamit001@gmail.com, anupkundumath@gmail.com, goutam.paul@isical.ac.in

Abstract. At Asiacrypt 2021, Baksi et al. introduced DEFAULT, the first block cipher designed to resist differential fault attacks (DFA) at the algorithm level, boasting of a 64-bit DFA security. The cipher initially employed a straightforward key schedule, where a single key was XORed in all rounds, and the key schedule was updated by incorporating round-independent keys in a rotating fashion. However, during Eurocrypt 2022, Nageler et al. presented a DFA attack that exposed vulnerabilities in the claimed DFA security of DEFAULT, reducing it by up to 20 bits in the case of the simple key schedule and even allowing for unique key recovery in the presence of rotating keys. In this work, we have significantly improved upon the existing differential fault attack (DFA) on the DEFAULT cipher. Our enhanced attack allows us to effectively recover the encryption key with minimal faults. We have accomplished this by computing deterministic differential trails for up to five rounds, injecting around 5 faults into the simple key schedule for key recovery, recovering equivalent keys with just 36 faults in the DEFAULT-LAYER, and introducing a generic DFA approach suitable for round-independent keys within the DEFAULT cipher. These results represent the most efficient key recovery achieved for the DEFAULT cipher under DFA attacks. Additionally, we have introduced a novel fault attack called the Statistical-Differential Fault Attack (SDFA), specifically tailored for linear-structured SBox-based ciphers like DEFAULT. This novel technique has been successfully applied to BAKSHEESH, resulting in a nearly unique key recovery. Our findings emphasize the vulnerabilities present in linear-structured SBox-based ciphers, including both DEFAULT and BAKSHEESH, and underscore the challenges in establishing robust DFA protection for such cipher designs. In summary, our research highlights the significant risks associated with designing linear-structured SBox-based block ciphers with the aim of achieving cipher-level DFA protection.

Keywords: Differential Fault Attack · Statistical Fault Attack · Statistical-Differential Fault Attack · DEFAULT · DFA Security

1 Introduction

The differential fault attack (DFA) is a powerful physical attack that poses a significant threat to symmetric key cryptography. Introduced in the field of block ciphers by Biham and Shamir [9], DFA [21,19,30] has proven to be capable of compromising the security of many block ciphers that were previously considered secure against classical attacks. While nonce-based encryption schemes can automatically prevent DFA attacks by incorporating nonces in encryption queries, the threat of DFA [24,17] still persists in designs with a parallelism degree greater than 2. Additionally, DFA [23,15,16] can pose a significant risk to nonce-based designs in the decryption query. In essence, DFA represents a significant challenge for cryptographic implementations whenever an attacker can induce physical faults. In response to this threat, the research community has focused on proposing countermeasures to enhance the DFA resistance of ciphers.

Countermeasures against fault injection attacks can be classified into two main categories. The first category focuses on preventing faults from occurring by utilizing specialized devices. The second category focuses on mitigating the impact of faults through redundancy or secure protocols. Countermeasures that mitigate the effects of fault injection attacks utilize redundancy for protection. These countermeasures can be classified into three categories based on where the redundancy is introduced: cipher level (no redundant computation), using a separate dedicated device, and incorporating redundancy in computation (commonly achieved through circuit duplication). Additionally, protocol-level techniques can also be employed to enhance fault protection.

Most of the countermeasures against attacks on cryptographic primitives, modes of operation, and protocols are focused on implementation-level defenses without requiring changes to the underlying cryptographic algorithms or protocols themselves. One effective countermeasure against DFA is to introduce redundancy into the system so that it can still function even if some faults or errors are introduced. Another countermeasure is to use error detection and correction codes. These codes can detect when errors or faults have occurred and correct them before they affect the output. Recent cryptographic designs propose primitives with built-in features to enable protected implementations against DFA attacks. For instance, FRIET [28] and CRAFT [8] are efficient and provide error detection. DEFAULT [4] is a more radical approach, aiming to prevent DFA attacks through cipher-level design. A brief survey on fault attacks and their countermeasures in symmetric key cryptography can be found in [3].

DEFAULT is a block cipher design proposed by Baksi *et al.* at Asiacrypt 2021 that provides protection against DFA attacks at the cipher level. The primary component of the DFA protection layer in DEFAULT (called the DEFAULT-LAYER) is a weak class linear structure (LS) based substitution boxes (SBox), which behave like linear functions in some aspects. The idea behind the DEFAULT design is that strong non-linear SBoxes are more resistant against classical differential attacks (DA), but weaker against DFA attacks. Conversely, weaker non-linear SBoxes are more resistant against DFA attacks but weaker against DA. Simply speaking, the DEFAULT cipher is a combination of DEFAULT-LAYER (where rounds are used LS SBoxes) and DEFAULT-CORE (where rounds are used non-LS SBoxes). To address this trade-off, DEFAULT maintains the main cipher, which is presumed secure against classical attacks, and adds two keyed permutations as additional layers before and after it. These keyed permutations have a unique structure that makes DFA non-trivial on them, resulting in a DFA-resistant construction. The SBox in DEFAULT-LAYER features three non-trivial LS elements, resulting in specific inputs/outputs becoming differentially equivalent, including the associated keys. As a result, attackers cannot learn more than half of the key bits by attacking the SBox layer. The designers claim that using DFA, an adversary can only recover 64 bits out of a 128-bit key, leaving a remaining keyspace of 2^{64} candidates that is difficult to brute-force. For even more security, the design approach can be scaled for a larger master key size. In their initial design [5], the authors first propose the simple key schedule function where the master key is used throughout each round in the cipher. Then in [4] the authors update the simple key schedule by recommending to use of the rotating key schedule function in the cipher to make it a more DFA secure cipher.

In [20], the authors initially demonstrate the vulnerability of the simple key schedule of the DEFAULT cipher to DFA attacks. They highlight that this attack can retrieve more key information than what the cipher's designers claimed, surpassing the 64-bit security level. The authors also present a method to retrieve the key in the case of a rotating key schedule by exploiting faults to create an equivalent key and then targeting the DEFAULT-CORE to recover the actual key. However, their attack on the simple key schedule does not achieve unique key recovery even with an increased number of injected faults. Moreover, as described in [11], this work presents a differential fault attack on the DEFAULT cipher under the simple key schedule, but it is worth noting that this attack is not applicable to the modified version of the cipher employing a key scheduling algorithm.

In recent times, Baksi et al. introduced a new lightweight block cipher based on linear structure (LS SBox) principles, as detailed in [6]. Similar to the DEFAULT-LAYER, which incorporates three non-trivial LS elements within its SBox, this newly introduced design features only one non-trivial LS element, resulting in a DFA security level of 2^{32}. Although the designers have not explicitly claimed any DFA security, we find it pertinent to conduct a comprehensive investigation into its DFA security, given its alignment with the LS SBox-based design paradigm.

1.1 Our Contributions

In this paper, we make several contributions in the field of fault attacks on LS SBox-based ciphers: DEFAULT and BAKSHEESH. Firstly, we demonstrate the vulnerability of the DEFAULT cipher to DFA attacks under bit-flip fault models, specifically targeting the simple key schedule. Our approach effectively reduces the key space with a minimal number of injected faults, surpassing the performance of previous attacks. To achieve this, we propose novel techniques for deterministic trail computation up to five rounds by analyzing the ciphertext differences. These techniques enable us to filter the intermediate rounds and further reduce the key space.

Furthermore, we extend our analysis to the rotating key schedule and showcase the efficiency of our approach in reducing the key space to a unique solution with a minimal number of faults. Additionally, we present a general framework for computing equivalent keys of the DEFAULT-LAYER cipher. By applying this framework, we demonstrate the efficacy of DFA attacks on rotating key schedules with significantly fewer injected faults.

Moreover, we introduce a new attack called the *Statistical-Differential Fault Attack* under the bit-set fault model. This attack efficiently recover the round keys of the DEFAULT cipher, even when the keys are independently chosen from random sources.

Finally, we applied our proposed DFA attack to another linear-structured SBox-based cipher, BAKSHEESH, efficiently recovering its master key uniquely. Likewise, under the bit-set fault model, the SDFA attack can be effectively applied to nearly retrieve its key uniquely.

To summarize our contributions, we offer a concise performance comparison between our enhanced attacks and previous attack methods in Table 1. Our work represents a substantial advancement in the field of fault attacks on LS SBox-based ciphers, notably the DEFAULT and BAKSHEESH ciphers, by introducing a highly effective key recovery strategy.

Cipher	Key Schedule	Relevant Works	Attack Strategy	Results		References
				# of Faults	Key Space	
DEFAULT	Simple	Nageler *et al.*	Enc-Dec IC-DFA	16	2^{39}	[20, Section 6.1]
			Multi-round IC-DFA	16	2^{20}	[20, Section 6.2]
		This Work	Second-to-Last Round Attack	64	2^{32}	Section 3.1.2
			Third-to-Last Round Attack	34	1	Section 3.1.3
			Fourth-to-Last Round Attack	16	1	Section 3.1.4
			Fifth-to-Last Round Attack	5	1	Section 3.1.5
			SDFA	$[64, 128]$	1	Section 4.2
	Rotating	Nageler *et al.*	Generic NK-DFA	$1728 + x$	1	[20, Section 4.3]
			Enc-Dec IC-NK-DFA	$288 + x$	2^{32}	[20, Section 5.1]
			Multi-round IC-NK-DFA	$(84 \pm 15) + x$	1	[20, Section 5.2, 6.3]
		This Work	Third-to-Last Round Attack	$96 + x$	1	Section 3.2.2.1
			Fourth-to-Last Round Attack	$48 + x$	1	Section 3.2.2.2
			Fifth-to-Last Round Attack	$36 + x$	1	Section 3.2.2.3
			SDFA	$[64, 128]$	1	Section 4.3
BAKSHEESH	Rotating	This Work	Second-to-Last Round Attack	40	1	Section 5.1.2
			Third-to-Last Round Attack	12	1	Section 5.1.3
			SDFA	128	1	Section 5.2

*x represents the number of faults to retrieve the key at the DEFAULT-CORE. We verified that 32 bit-faults at the second-to-last round in DEFAULT-CORE achieve unique key recovery.

Table 1: Differential Fault Attacks on DEFAULT with Different Key Schedules

2 Preliminaries

In this section, we will introduce the notations that will be utilized throughout the paper. Following that, we will provide descriptions of the DEFAULT and BAKSHEESH ciphers. Subsequently, we will offer a concise overview of DFA attacks, followed by an in-depth discussion of the linear structure (LS) SBox, a crucial element in designing a block cipher with DFA protection. The following notations are used throughout the paper.

- $a \oplus b$ denotes the bit-wise XOR of a and b.
- $+$ denotes the integer addition.
- $\cup, \cap$ denotes the set union and intersection respectively.
- ΔC denotes the ciphertext difference.

2.1 Description of DEFAULT Cipher

The DEFAULT cipher [4] is a lightweight block cipher with a 128-bit state and key size. It is designed to resist DFA attacks by limiting the amount of key information that can be learned by an attacker. The cipher incorporates two keyed permutations, known as DEFAULT-LAYER, as additional layers before and after the main cipher. These layers provide protection against DFA attacks and other classical attacks. The DEFAULT cipher consists of two main building blocks: DEFAULT-LAYER and DEFAULT-CORE. The DEFAULT-LAYER layer protects the cipher from DFA attacks, while the DEFAULT-CORE layer protects against classical attacks. The encryption function of the DEFAULT cipher can be expressed as $Enc = Enc_{\text{DEFAULT-LAYER}} \circ Enc_{CORE} \circ Enc_{\text{DEFAULT-LAYER}}$, indicating that the encryption process involves applying the DEFAULT-LAYER function before and after the DEFAULT-CORE function.

The DEFAULT cipher employs a total of 80 rounds, with the DEFAULT-LAYER function being applied 28 times and the DEFAULT-CORE function being applied 24 times. Each round function consists of a structured 4-bit SBox layer, a permutation layer, an add round constant layer, and an add round key layer. The DEFAULT-LAYER function utilizes a linear structured SBox, while the DEFAULT-CORE function utilizes a non-linear structured 4-bit SBox. In the following sections, we will discuss each component of the DEFAULT cipher in detail.

2.1.1 SBoxes The DEFAULT-LAYER layer of the DEFAULT cipher utilizes a 4-bit Linear Structured SBox, denoted as S. Table 2a shows the mapping of input and output values for this SBox, and it consists of four linear structures: $0 \rightarrow 0$, $6 \rightarrow a$, $9 \rightarrow f$, and $f \rightarrow 5$. The definition of a linear structure can be found in Definition 1. Similarly, the DEFAULT-CORE layer uses another SBox, denoted as S_c. Table 2b provides the input-output mapping for this SBox. To evaluate the differential behavior of S and S_c, the differential distribution tables are given in Table 3a and Table 3b respectively.

$x:$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x):$	0	3	7	e	d	4	a	9	c	f	1	8	b	2	6	5

(a) DEFAULT-LAYER SBox

$x:$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S_c(x):$	1	9	6	f	7	c	8	2	a	e	d	0	4	3	b	5

(b) DEFAULT-CORE SBox

Table 2: SBoxes for DEFAULT cipher

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16															
1				8						8						
2								8						8		
3					8										8	
4								8						8		
5					8										8	
6											16					
7				8						8						
8							8						8			
9																16
a		8										8				
b			8						8							
c		8										8				
d			8						8							
e							8						8			
f						16										

(a) DDT of S

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16															
1					2			2	2	2	2	2		2	2	
2							4	4							4	4
3		2		2	2	2			2		2				2	2
4						4	4							4	4	
5					2			2	2	2	2	2		2	2	
6		4		4						4		4				
7		2		2	2	2			2		2				2	2
8				4				4				4				4
9			2	2	2		2		2	2				2		2
a		4								4			8			
b		2	2		2	2	2	2	2			2				
c			4			4					4			4		
d			2	2	2		2		2	2				2		2
e			4								4		8			
f		2	2		2	2	2	2	2			2				

(b) DDT of S_{core}

Table 3: DDT of SBoxes used in DEFAULT

Definition 1 (Linear Structure). *For $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$, an element $a \in \mathbb{F}_2^n$ is called a linear structure of F if for some constant $c \in \mathbb{F}_2^n$, $F(x) \oplus F(x \oplus a) = c$ holds $\forall x \in \mathbb{F}_2^n$.*

2.1.2 Permutation Bits

The DEFAULT cipher incorporates the GIFT-128 permutation (P) in each of its rounds, which is derived from the GIFT [7] cipher. In the permutation layer of the GIFT cipher, there are two versions: one with 4 Quotient-Remainder groups for the 64-bit version, and another with 8 Quotient-Remainder groups for the 128-bit version. It is worth noting that these 8 Quotient-Remainder groups do not diffuse over themselves for 2 rounds.

2.1.3 Add Round Constants

For DEFAULT cipher, a round constant of 6-bits are XORed with the indices 23, 19, 15, 11, 7 and 3 respectively at each of the rounds. Along with this, the bit index 127 is flipped at each round to modify the state bits.

2.1.4 Add Round Key

The round key for DEFAULT cipher is 128 bits in length. In the first preprint version of DEFAULT, a simple key schedule was used where all the round keys were the same as the master key for each round. However, in a later version, a stronger key schedule was proposed to enhance security against DFA attacks. In this updated version, the authors introduced an idealized key schedule where each round key is independent of the others.

$x:$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x):$	3	0	6	d	b	5	8	e	c	f	9	2	4	a	7	1

(a) SBox

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16															
1				4			4					4			4	
2				4		4						4		4		
3						4	4							4	4	
4				4		4			4						4	
5						4	4		4			4				
6									4			4		4	4	
7				4			4		4					4		
8																16
9		4			4					4			4			
a			4		4						4		4			
b		4	4							4	4					
c		4						4			4		4			
d					4			4		4	4					
e		4	4		4			4								
f			4					4		4			4			

(b) DDT of SBox

Table 4: SBox and DDT of BAKSHEESH cipher

Although this idealized scheme requires 28×128 key bits to encrypt 128 bits of state using the DEFAULT cipher, it is not practical. To address this, the authors employed an unkeyed function R to generate four different round keys $K_0, \cdots, K_3$, where $K_0 = K$ and $K_i = R^4(K_{i-1})$ for $i \in 1, 2, 3$. These four round keys are then used periodically for each round to encrypt the plaintext.

2.2 Specification of BAKSHEESH

BAKSHEESH [6] is a lightweight block cipher designed to process 128-bit plaintexts. It is based on the GIFT-128 [7] cipher, featuring 35 rounds of encryption. Within its design, BAKSHEESH employs a 4-bit substitution-permutation box (SBox) with a non-linear LS element. The round function of BAKSHEESH comprises four operations: SubCells–applying a 4-bit linear structured SBox to the state, PermBits–permuting the bits of the state (similar to GIFT-128), AddRoundConstants–XORing a 6-bit constant and an additional bit to the state (similar to GIFT-128), and AddRoundKey–XORing the round key with the state. The SBox and its DDT are provided in Table 4a and Table 4b, respectively. BAKSHEESH exhibits a single linear structure at 8. Additionally, concerning the round keys, the first round key matches the master key, and subsequent round keys are generated with a 1-bit right rotation. More details about the specification of BAKSHEESH cipher cen be found in [6].

2.3 Differential Fault Attack

Differential Fault Attack (DFA) is a type of Differential Cryptanalysis that operates in the grey-box model. In this attack, the attacker deliberately introduces faults during the final stages of the cipher to extract the secret component effectively. In contrast, the security of a cipher against Differential Cryptanalysis in the black-box model depends on the probability of differential trails (fixed input/output difference) being as low as possible. However, in DFA, the attacker can introduce differences at the intermediate stages by inducing faults, increasing the trail probability for those rounds significantly. As a result, the attacker can extract the secret component more efficiently than in Differential Cryptanalysis in the black-box model. Finally, estimating the minimum number of faults is crucial in DFA to ensure the attack is both efficient and effective, keeping the search complexity within acceptable limits. To protect ciphers from DFA attacks, various state-of-the-art countermeasures have been proposed, including the use of dedicated devices or shields that prevent any potential sources of faults. Other countermeasures include the implicit/explicit detection of duplicated computations and mathematical solutions designed to render DFA ineffective or inefficient.

2.4 Revisiting Learned Information via the Linear Structure SBox

A linear structure SBox is a class of permutations that exhibit some properties of linear functions, making them weaker than non-linear permutations in certain aspects. The SBox S used in DEFAULT-LAYER has four linear structures as $\mathcal{L}(S) = \{0, 6, 9, f\}$. According to the DDT (Table 3a) of S, the non-trivial linear structures are $6, 9$ and f. Similarly, for the inverse SBox S^{-1}, the set of all linear structures of S^{-1} will be $\mathcal{L}(S^{-1}) = \{0, 5, a, f\}$. In their work [4], the designers demonstrate that inducing bit flips before the SBox can yield limited information to attackers, reducing key bits from 4 to 2 during encryption faults. However, in [20], Nageler et al. showed an improved DFA targeting the decryption algorithm, further reducing key bits to 1. This reduction to 2^{32} contradicts the initial claim of 2^{64} key space reduction. Learning key information from a linear structure SBox is non-trivial, and previous works lack detail on this aspect. This section revisits how attackers can glean key information from faults injected during both encryption and decryption queries at the SBox.

Learned Information from S/S^{-1}. Suppose that (x_0, x_1, x_2, x_3) and (y_0, y_1, y_2, y_3) are respectively the bit-level input and output of SBox S. Similarly, (y_0, y_1, y_2, y_3) and (x_0, x_1, x_2, x_3) are the input and output of S^{-1}. Note that, the output of S is same as the input to S^{-1} and vice-versa. Consider a set $\mathcal{A}$ of inputs which satisfy the differential $\alpha \to \beta$ for the SBox S, i.e., $\mathcal{A} = \{x : S(x) \oplus S(x \oplus \alpha) = \beta\}$. Then, for any $y \in \mathcal{L}(S)$, we have,

$$\begin{aligned} S(x \oplus y) \oplus S(x \oplus y \oplus \alpha) &= (S(x) \oplus S(x \oplus y)) \oplus (S(x \oplus \alpha) \oplus S(x \oplus y \oplus \alpha)) \oplus (S(x) \oplus S(x \oplus \alpha)) \\ &= \beta. \quad [\text{As, } (S(x) \oplus S(x \oplus y)) = (S(x \oplus \alpha) \oplus S(x \oplus \alpha \oplus y)).] \end{aligned}$$

This result shows that $x \in \mathcal{A} \implies x \oplus y \in \mathcal{A}, y \in \mathcal{L}(S)$. Thus, for any input $x \in \{0, 1, \ldots, f\}$, the attacker cannot uniquely identify which among $\{x, x \oplus 6, x \oplus 9, x \oplus f\}$ is the actual input to the SBox. Further, this can be partitioned into four subsets as $\{\{0, 6, 9, f\}, \{1, 7, 8, e\}, \{2, 4, b, d\}, \{3, 5, a, c\}\} = \{\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3\}$. Similarly, for S^{-1}, the partition will be $\{\{0, 5, a, f\}, \{1, 4, b, e\}, \{2, 7, 8, d\}, \{3, 6, 9, c\}\} = \{\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$. The input bit relations of $\mathcal{B}_i/\mathcal{D}_i$'s of S/S^{-1} are denoted by $\mathcal{B}_i^{eq}/\mathcal{D}_i^{eq}$ and given in Table 5. For example, consider the SBox S^{-1} (for encryption) with a differential $7 \to 2$. Then, the number of inputs that satisfy $7 \to 2$

$\mathcal{B}_0^{eq}$	$\mathcal{B}_1^{eq}$	$\mathcal{B}_2^{eq}$	$\mathcal{B}_3^{eq}$	$\mathcal{D}_0^{eq}$	$\mathcal{D}_1^{eq}$	$\mathcal{D}_2^{eq}$	$\mathcal{D}_3^{eq}$
$\sum_{i=0}^{3} x_i = 0$	$\sum_{i=0}^{3} x_i = 0$	$\sum_{i=0}^{3} x_i = 1$	$\sum_{i=0}^{3} x_i = 1$	$\sum_{i=0}^{3} y_i = 0$	$\sum_{i=0}^{3} y_i = 1$	$\sum_{i=0}^{3} y_i = 1$	$\sum_{i=0}^{3} y_i = 0$
$x_0 \oplus x_3 = 0$	$x_0 \oplus x_3 = 1$	$x_0 \oplus x_3 = 0$	$x_0 \oplus x_3 = 1$	$y_0 \oplus y_2 = 0$	$y_0 \oplus y_2 = 1$	$y_0 \oplus y_2 = 0$	$y_0 \oplus y_2 = 1$
$x_1 \oplus x_2 = 0$	$x_1 \oplus x_2 = 1$	$x_1 \oplus x_2 = 1$	$x_1 \oplus x_2 = 0$	$y_1 \oplus y_3 = 0$	$y_1 \oplus y_3 = 0$	$y_1 \oplus y_3 = 1$	$y_1 \oplus y_3 = 1$

Table 5: Input Bit Relations of Partition Correspond to S/S^{-1}

will be $\mathcal{D}_2 \cup \mathcal{D}_0 = \{0, 5, a, f, 2, 7, 8, d\}$ and hence, the attacker can learn the bit relation of this input set $\mathcal{D}_2 \cup \mathcal{D}_0$ as $\mathcal{D}_2^{eq} \cap \mathcal{D}_0^{eq} \implies y_0 \oplus y_2 = 0$. Similarly, if the differential $7 \to 4$ happens, then the attacker can learn the bit relation as $\mathcal{D}_1^{eq} \cap \mathcal{D}_3^{eq} \implies y_0 \oplus y_2 = 1$. In this way, for any differential $\alpha \to \beta$ of S^{-1}, the attacker can learn the bit relation of the inputs that satisfy $\alpha \to \beta$. Conversely, if we consider the SBox S (for decryption) with differential $\gamma \to \delta$, the attacker can learn the bit relation from the sets $\mathcal{B}_i, i \in \{0, 1, 2, 3\}$. For example, the inputs to satisfy the differential $2 \to 7$ will be $\mathcal{B}_2 \cup \mathcal{B}_0$ and thus, input bit relation will be $\mathcal{B}_2^{eq} \cap \mathcal{B}_0^{eq} \implies x_0 \oplus x_3 = 0$. Similarly, for $2 \to d$, the learned information will be $\mathcal{B}_1^{eq} \cap \mathcal{B}_3^{eq} \implies x_0 \oplus x_3 = 1$.

Consider an encryption query where difference is injected at the last round before the SBox operaration. Let (k_0, k_1, k_2, k_3) be the key XORed with the output of SBox and outputs the ciphertext (ignore the linear layer). Now, for each SBox, we are going to combine these learned information for the input/output of S/S^{-1} with the key to learn the corresponding key relation. For example, consider the learned information $y_0 \oplus y_2 = 0$ for a given differential $2 \to 7$ of S ($7 \to 2$ for S^{-1}). If c be the non-faulty ciphertext, then we have,

$$c_0 \oplus c_2 = (y_0 \oplus y_2) \oplus (k_0 \oplus k_2) \implies (k_0 \oplus k_2) = (c_0 \oplus c_2) \oplus (y_0 \oplus y_2) = c_0 \oplus c_2.$$

This relation shows that the attacker can learn the key information from the ciphertext relation. In the way, for both encryption and decryption, an attacker can learn key informations for each non-zero differential of S/S^{-1}. In Table 6, we summarize the key bits information for both enc/dec which can be learned based on the input difference of S/S^{-1}.

Direction	Learned expression															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Enc (S^{-1})	1	$\sum_{i=0}^{3} k_i$	$k_0 \oplus k_2$	$k_1 \oplus k_3$	$k_0 \oplus k_2$	$k_1 \oplus k_3$	1	$\sum_{i=0}^{3} k_i$	$\sum_{i=0}^{3} k_i$	1	$k_1 \oplus k_3$	$k_0 \oplus k_2$	$k_1 \oplus k_3$	$k_0 \oplus k_2$	$\sum_{i=0}^{3} k_i$	1
Dec (S)	1	$\sum_{i=0}^{3} k_i$	$k_0 \oplus k_3$	$k_1 \oplus k_2$	$\sum_{i=0}^{3} k_i$	$k_1 \oplus k_2$	1	$k_0 \oplus k_3$	$k_0 \oplus k_3$	1	$k_1 \oplus k_2$	$\sum_{i=0}^{3} k_i$	$k_1 \oplus k_2$	$k_0 \oplus k_3$	$\sum_{i=0}^{3} k_i$	1

Table 6: Learned Key-Information when faulting at (S/S^{-1})

3 Our Improvements of DFA on DEFAULT Cipher

In this work, we focus on improving the previously proposed differential fault analysis (DFA) attack on the DEFAULT cipher, specifically on both its simple and rotating key schedules. To enhance this attack, we first introduce a strategy that allows for the deterministic computation of the internal differential path when faults are injected up to the fifth-to-last rounds. We demonstrate the effectiveness of this method by applying it to the simple key schedule of the DEFAULT cipher and showing that an attacker can recover the key if faults are introduced during the third, fourth, or fifth-to-last rounds. Additionally, we improve the DFA attack on the rotating key schedule of the DEFAULT cipher. Throughout the paper, we use the encryption oracle to inject faults. Overall, our work is focused on fully breaking the DFA security of the DEFAULT cipher under difference-based fault attacks and providing insights into the challenges of using linear structure (LS) substitution boxes (SBox) in block ciphers to achieve cipher-level protection.

Fault Model. In this attack, we consider a fault model where the goal is to induce a precise single bit-flip faults in the cryptographic state nibble during the encryption query. For instance, an attacker might deliberately introduce a single bit-flip fault to alter a single bit in the nibble, located just before the input to the SBox operation, at the i^{th} last round of the state during encryption. Achieving this level of precision is feasible in practice, as attackers can employ techniques such as Laser fault injection [2,14,27]. These methods offer high accuracy in both space and time. Additionally, electromagnetic (EM) fault injection serves as an alternative method that does not require the de-packaging of the chip. Practical implementation of precise bit-level fault injections has been demonstrated through EM fault injection setups, as illustrated in [26].

3.1 Attacks on Simple Key Schedule

In their previous work, Nageler et al. [20] expanded their DFA attack by inducing bit-flip faults across multiple rounds to further reduce the key space. Their strategy involved injecting differences at certain rounds and exploring all possible differential paths through subsequent rounds based on the DDT. By analyzing the distribution of input/output differences at each SBox in subsequent rounds, they conducted differential analysis to recover key bits. However, this approach could not reduce the key space beyond 2^{16}, despite the potential for inducing additional faults. As a result, we delved deeper into this issue and devised a novel approach to achieve complete key recovery with significantly fewer faults. Moreover, our proposed attack enables key retrieval with significantly reduced offline computation time compared to previous approaches.

In this section, we present our strategy for deterministic computation of the differential trail up to five rounds in order to perform efficient DFA attacks. We describe how we compute the trail and utilize it to retrieve the key using bit-flip faults. Additionally, we analyze the number of faults required to uniquely recover the key for different rounds, providing an estimate of the fault complexity involved in the attack.

3.1.1 Faults at the Last Round In this attack scenario, the attacker needs to inject faults and analyze each of the 32 Substitution Boxes (SBox) independently. As per the designers' claim, injecting faults at each SBox nibble can reduce the search space from 2^4 to 2^2 at most, resulting in a total search complexity of $4^{32} = 2^{64}$. However, in [20], the authors further reduce the SBox nibble space to 2 by injecting faults at the decryption algorithm. Specifically, the authors demonstrate how to derive three linearly independent equations for each nibble of the key by inducing two and one faults in the encryption and decryption algorithms, respectively. It is worth noting that computing the table, which calculates the learned information involving the key nibbles for encryption and decryption algorithms, is not a straightforward process according to their work. Hence, we revisit the methodology for computing this table regarding the learned information involving the key nibbles and aim to provide a more detailed explanation.

Based on the information learned from Table 6, an attacker can learn two bits of information for each nibble in the last round of the DEFAULT-LAYER. One approach to reduce the key space is to inject two single bit-flip faults at each nibble in the last round before the SBox operation and reduce the key nibbles of 2^2 individually, resulting in a key space reduction to 2^{64} by inducing $2 \times 32 = 64$ number of bit-faults at the last round. However, a more efficient strategy is needed to induce faults further from the last rounds and deterministically obtain information about the input differences of each SBox in the last round. This requires developing a strategy that can deterministically guess the differential path from which the faults are injected. In the upcoming subsections, we will demonstrate that it is possible to deterministically guess the differential path of the DEFAULT-LAYER up to five rounds. By inducing around five bit-flip faults at the fifth-to-last round, we estimate that the key space can be reduced to 2^{64} with greater efficiency than the naive approach.

3.1.2 Faults at the Second-to-Last Round In this attack scenario, we assume that bit-faults are introduced at each nibble during the second-to-last round of the DEFAULT-LAYER. As a result, the fault propagation can affect at most four nibbles in the final round of the DEFAULT-LAYER. The DEFAULT-LAYER uses the GIFT-128 bit permutation internally, which has a useful property known as the Quotient-Remainder group structure. At round r, the 32 nibbles of a DEFAULT state are denoted as $S_i^r, i = 0, \ldots, 31$ and can be grouped into eight groups $\mathcal{G}\mathrm{r}_i = (S_{4i}^r, S_{4i+1}^r, S_{4i+2}^r, S_{4i+3}^r)$ for $i = 0, \ldots, 7$. This property states that any group at round r is permuted to a group of four nibbles at round $r+1$ through a 16-bit permutation, i.e.,

$$\mathcal{G}\mathrm{r}_i \xrightarrow{\text{16 bit permutation}} (S_i^r, S_{i+8}^r, S_{i+16}^r, S_{i+24}^r), i = 0, \ldots, 7.$$

The structure of the cipher allows for a nibble difference at the input of group $\mathcal{G}\mathrm{r}_i$ in the second-to-last round to induce a bit difference in four nibbles $S_i^{r+1}, S_{i+8}^{r+1}, S_{i+16}^{r+1}$, and S_{i+24}^{r+1} in the last round. This observation enables an attacker to deterministically determine the differential path by injecting bit-flip faults at the second-to-last round. Moreover, this observation allows for the deterministic computation of the differential paths up to five rounds, which we will discuss in the next subsections. This is possible because for each non-faulty and faulty ciphertext, the last round can be inverted by checking the input bit-difference at each nibble using the differential distribution table (DDT). The internal state difference can then be computed by checking the input bit-difference after the second-to-last round's inverse using the Quotient-Remainder group structure.

Attack Strategey. To attack the cipher in this scenario, a simple approach is to inject two bit-faults at each nibble in the last round, reducing the keyspace of each nibble to 2^2, i.e., the overall keyspace is thus reduced to 2^{64}. Then, inject one fault at each nibble in the second-to-last round, reducing the keyspace to 2^{32}. To accomplish this, we first group the 32 nibbles of the state into eight groups $\mathcal{G}\mathrm{r}_i$, each consisting of four nibbles, and consider the combined key space of nibble positions $i, i+8, i+16$, and $i+24$ for each group $\mathcal{G}\mathrm{r}_i$.

For each key in the combined key space of $\mathcal{G}\mathrm{r}_i$, we invert two rounds by considering the equivalent key classes of individual nibble positions at the second-to-last round and checking whether they satisfy $\mathcal{G}\mathrm{r}_i$'s input difference at the second-to-last round. By doing this, we can determine the internal state difference between the faulty and non-faulty ciphertexts. It is noteworthy that injecting faults in more than one nibble within $\mathcal{G}\mathrm{r}_i$ during encryptions at the second-to-last round can further reduce the keyspace for that group, potentially from 2^{16} to 2^4. The overall keyspace has now been effectively reduced to $2^{4 \cdot 8} = 2^{32}$, considering 8 groups denoted as $\mathcal{G}\mathrm{r}_i$. Initially, this approach necessitates approximately $2 \times 32 + 32 = 96$ bit-faults to achieve this reduction in the keyspace. However, we have enhanced this attack by introducing faults at the second-to-last round during encryption. Our practical verification shows that injecting faults at each SBox in the second-to-last round, specifically inducing faults at the least significant bits of the nibble (i.e., either at index 1 or index 2), notably reduces the keyspace to 2^{32}. This is because the output difference spread more differences if the input difference is either 2 or 4 (see DDT in Table 3a). The specific values representing the reduced keyspace for varying numbers of injected faults can be found in Table 10 (Appendix A).

3.1.3 Faults at the Third-to-Last Round

In this section, we focus on the key space reduction using Difference-based Analysis (DFA) for three rounds. We introduce a fault at the third-to-last round of the cipher, i.e., at round R^{25} in DEFAULT-LAYER. Throughout the attack, we induce bit-faults at the nibbles to generate input differences, and we assume that we know the nibble index where the input differences are injected. The attack consists of two phases. In the initial phase, we inject a bit fault at the input of the third-to-last round and determine the trail of three rounds deterministically. To achieve this, we compute the input and output differences of every nibble at each round, allowing us to trace the propagation of differences through the cipher. By carefully analyzing the trail, we can establish a deterministic relationship between the input differences and the output differences, enabling us to deduce the trail with high confidence. In the second phase, we utilize the computed trail to reduce the key space of the cipher. With knowledge of the trail, we can target specific nibbles and their corresponding input differences at the last round. By exploiting these input differences, we can perform DFA and significantly reduce the key space. This reduction is based on the fact that we now have knowledge of the correlations between the input-output differences and the key bits, allowing us to make informed guesses and narrow down the possible key values.

Deterministic Trail Finding. We know that a nibble difference at position $\mathcal{G}\text{r}_i$ can activate the four nibbles at positions i, $i+8$, $i+16$, and $i+24$ after one round of the DEFAULT cipher. Then, the nibble differences propagate to the groups $\mathcal{G}\text{r}_{\frac{j}{4}}$, where $j = i, i+8, i+16, i+24$, in the next round. By inducing an input difference at any nibble before the SBox operation in the third-to-last round R^{25} of DEFAULT-LAYER, we can activate the nibbles at positions i, $i+8$, $i+16$, and $i+24$ in the second-to-last round. Furthermore, the nibble differences in the groups $\mathcal{G}\text{r}_{\frac{j}{4}}$, where $j = i, i+8, i+16, i+24$, in the second-to-last round can activate at most all the even-positioned nibbles in the last round. This fault propagation property is illustrated in Figure 6 (Appendix A). This property of differential propagation allows us to determine the differential trail deterministically when an attacker injects bit-faults at the third-to-last round. The procedure for computing the differential trail is described in Algorithm 1. This algorithm takes advantage of the single bit differences in the input of each SBox at the last three rounds. By systematically analyzing the propagation of these single bit differences, we can construct the differential trail with certainty.

Key Recovery. For each differential trail, we begin by narrowing down the key nibble spaces associated with active SBox in the final round through a comparison of non-faulty and faulty ciphertexts. By introducing two distinct bit differences at each nibble in the final round, we can efficiently reduce the key space to 2^2. Next, we focus on each group $\mathcal{G}\text{r}_i$, where i ranges from 0 to 7, at the second-to-last round. We combine the key spaces from the nibble positions i, $i+8$, $i+16$, and $i+24$ based on the key nibbles of the last round. For each combined key, we perform the inverse of one round and check the corresponding trail list to determine the resulting differential. At this stage, we use the equivalent key nibble obtained from the reduction at the last round. If the computed differential matches the observed differential, we consider the combined key as a potential key combination. This filtering process is applied to each group at the second-to-last round. Finally, we create combined key spaces for each even/odd position based on the key reductions at the second-to-last round. These correspond to the left/right half of the nibbles at the third-to-last round. It is important to note that faults introduced at the sixteen least/most significant nibbles of the third-to-last round can affect almost all the even/odd position nibbles in the last round. Our practical verification demonstrates that injecting faults at each SBox in the third-to-last round, involving the flipping of the bit at either index 1 or index 2, significantly reduces the keyspace to nearly a unique key. The specific values representing the reduced keyspace for varying numbers

Algorithm 1 DETERMINISTIC COMPUTATION OF THREE ROUNDS DIFFERENTIAL TRAIL

Input: A list of ciphertext difference $\mathcal{L}_{\Delta C}$, faulty value δ, and faulty nibble position x
Output: Lists of input-output differences $\mathcal{A}^{25}_{iod}, \mathcal{A}^{26}_{iod}$, & $\mathcal{A}^{27}_{iod}$ at the third-to-last, second-to-last, and the last round respectively

Initialize $\mathcal{L}_1 \leftarrow [\,], \mathcal{A}^{25}_{iod} \leftarrow [[\,],[\,]], \mathcal{A}^{26}_{iod} \leftarrow [[\,],[\,]], \mathcal{A}^{27}_{iod} \leftarrow [[\,],[\,]], \mathcal{D}^{25}_{od} \leftarrow [\,], \mathcal{D}^{26}_{id} \leftarrow [\,]$
$\mathcal{A}^{25}_{iod}[0] = [0 \text{ for } i \text{ in range}(32)]$
$\mathcal{A}^{25}_{iod}[0][x] = delta$
$\mathcal{D}^{25}_{od} = \mathcal{D}^{26}_{id} = [0 \text{ for } i \text{ in range}(32)]$ ▷ Dummy output state difference list after the SBox layer
$\mathcal{D}^{25}_{od}[x] = 0xf$
$\mathcal{L}_1 = P(\mathcal{D}^{25}_{od})$'
$\mathcal{D}^{26}_{id} = findActiveSBox(\mathcal{L}_1)$ ▷ For each non-zero nibble value, this function assign 1 to this nibble index, otherwise it assign to 0
$\mathcal{A}^{27}_{iod}[1] = P^{-1}(\mathcal{L}_{\Delta C})$
$L_3 = [\,]$ ▷ Third layer possible input difference list
for $i = 0$ to $\mathcal{A}^{27}_{iod}[1]$ **do**
    Append $\mathsf{DDT}^{-1}[i]$ to the list L_3
$\mathcal{A}^{27}_{iod}[0] = [0 \text{ for } i \text{ in range}(32)]$
for $pos,\ i$ in enumerate(L_3) **do**
    if $i \neq [0]$ **then** $dList = [0 \text{ for } i \text{ in range}(32)]$
    for dif in i **do**
        $dList[pos] =$ dif
        if $list\_subset(findActiveSBox(P^{-1}(dList)), \mathcal{D}^{26}_{id}) == 1$ **then**
            $\mathcal{A}^{27}_{iod}[0][pos] =$ dif
$\mathcal{A}^{26}_{iod}[1] = P^{-1}(\mathcal{A}^{27}_{iod}[0])$
$L_2 = [\,]$ ▷ Second layer possible input difference list
for i in $P^{-1}(\mathcal{A}^{27}_{iod}[0])$ **do**
    Append $\mathsf{DDT}^{-1}[i]$ to the list L_2
$\mathcal{A}^{26}_{iod}[0] = [0 \text{ for } i \text{ in range}(32)]$
for $pos,\ i$ in enumerate(L_2) **do**
    if $i \neq [0]$ **then** $dList = [0 \text{ for } i \text{ in range}(32)]$
    for dif in i **do**
        $dList[pos] =$ dif
        if $list\_subset(findActiveSBox(P^{-1}(dList)), \mathcal{D}^{25}_{od}) == 1$ **then**
            $\mathcal{A}^{26}_{iod}[0][pos] =$ dif
$\mathcal{A}^{25}_{iod}[1] = P^{-1}(\mathcal{A}^{26}_{iod}[0])$
return the lists $\mathcal{A}^{27}_{ID}, \mathcal{A}^{26}_{ID}$ and $\mathcal{A}^{25}_{ID}$

of injected faults can be found in Table 10. Figure 1 shows the distribution of the size of the reduced keyspace after this attack.

3.1.4 Faults at the Fourth-to-Last Round In this section, we demonstrate the deterministic computation of the differential trail and propose an attack that requires fewer faults compared to the previous attack on three rounds. We introduce bit-flip nibble faults at the fourth-to-last round of the cipher, specifically at round R^{24} in DEFAULT-LAYER. These introduced bit-flip nibble faults at the fourth-to-last round cause the nibble differences in the left half (16 least significant nibbles) or right half (the next 16 nibbles) of the fourth-to-last round to propagate to almost all even or odd nibbles, respectively, at the second-to-last round. Furthermore, at the last round, the differences in even or odd nibbles activate all 32 nibbles in the state. In this attack, we first compute the trail deterministically and then based on the computed trail for each fault, we recover the key. By exploiting the known correlations between input-output

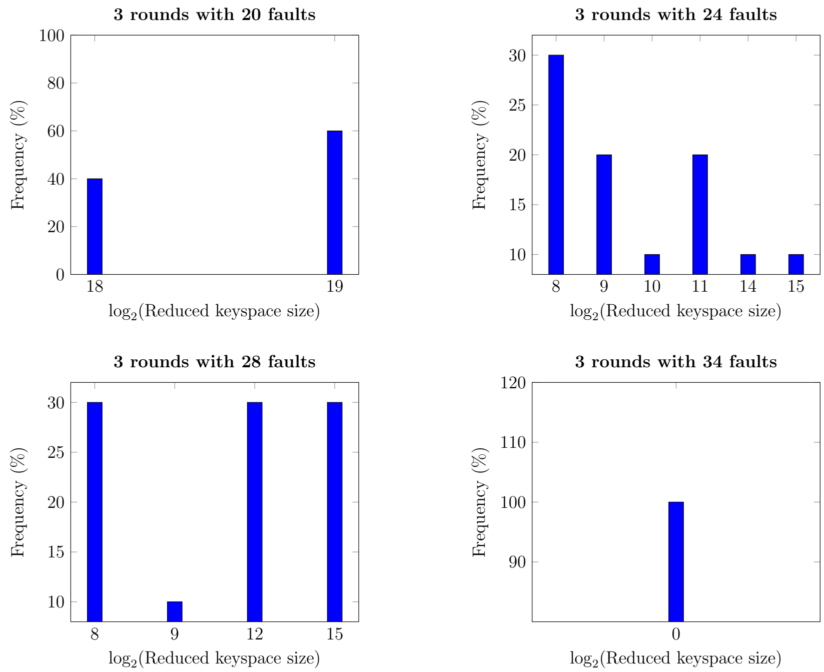

Fig. 1: Distribution of the Reduced Keyspace for the Third-to-Last Round Attack

differences and key bits, we can significantly reduce the key space with a smaller number of injected faults compared to the previous attack.

Deterministic Trail Finding. To compute the trail, we first determine the unique input-output nibble differences for each SBox at the last round. Once these differences are established, we can utilize Algorithm 1 to compute the trail for the remaining three rounds. Assuming that nibble differences arise at all even positions in the state at the second-to-last round before the SBox operations, we have exactly two active even nibbles in each group $\mathcal{G}\mathrm{r}_i$ at this round. Consequently, the input nibble difference at each SBox in the last round will no longer be a simple bit difference. Therefore, for each output of SBox at the last round, there are two possible choices of input differences, which may not be in the form of single-bit nibble differences.

To determine the output difference of SBoxes in $\mathcal{G}\mathrm{r}_i$ at the second-to-last round, we exhaustively consider all combined input differences corresponding to the positions i, $i{+}8$, $i{+}16$, and $i{+}24$ from the last round. We then check whether, after the bit permutation, these differences only go to the even nibble positions in $\mathcal{G}\mathrm{r}_i$, and their corresponding input differences are single-bit differences. This strategy allows us to uniquely identify the output difference of SBoxes in $\mathcal{G}\mathrm{r}_i$

at the second-to-last round. The process is described in detail in Algorithm 5 (Appendix A).

Key Recovery. Earlier, we explained the process of computing the unique trail based on both non-faulty and faulty ciphertexts when injecting faults during the fourth-to-last rounds. Once the trail is computed, we can proceed to reduce the key space by analyzing the last three rounds, as explained earlier. To achieve this, we iterate exhaustively through the entire keyspace at the last round for each input-output nibble difference at the fourth-to-last round. We invert the intermediate rounds by using the reduced keys at each round and filter out incorrect keys. By repeating this process for each input-output nibble difference in the last four rounds, we can significantly reduce the key space, approaching a nearly unique solution. Hence, through the analysis of input-output differences and the iterative refinement of the key space via intermediate round inversions, we can effectively narrow down the potential key candidates and approximate the correct key with a high level of confidence. Our practical validation confirms that the introduction of 8 bit-faults in each half of the SBox (both left and right) in the fourth-to-last, achieved by flipping a bit at either index 1 or index 2, substantially diminishes the keyspace, resulting in nearly unique keys. Detailed information on the reduced keyspace values corresponding to different fault injection counts is available in Table 10. Figure 2 shows the distribution of the size of the keyspace after this attack.

3.1.5 Faults at the Fifth-to-Last Round

In this section, we discuss how we can deterministically compute the differential trail when injecting faults during the fifth-to-last round (round R^{23}) in the DEFAULT-LAYER cipher. These faults can be injected either in the left half (from nibble positions 0 to 15) or the right half (from positions 16 to 31), affecting either all the even nibble positions or the odd nibble positions in the state at the third-to-last round. An example of fault propagation resulting from a nibble fault in the left half is illustrated in Figure 7 (Appendix A).

Furthermore, the differences in even/odd nibbles at the third-to-last round activate all the nibbles in the second-to-last round and subsequently in the last round as well. In this attack scenario, we compute the trail for five rounds uniquely and then estimate the number of faults required to recover the key. By doing so, we can significantly reduce the key space using a smaller number of faults compared to our previous approaches.

Deterministic Trail Finding. To compute the trail for five rounds when injecting faults at the fifth-to-last round, the approach involves inverting two rounds and then determining the upper three rounds' trails based on the possible differences at the third-to-last round. The objective is to check if these trails satisfy the input difference at the fifth-to-last round. When faults are injected at the left or right half during the fifth-to-last round, the nibble differences in each group $\mathcal{G}\mathrm{r}_i$, where $i \in 0, 1, \cdots, 7$, in the input to the third-to-last round follow a specific

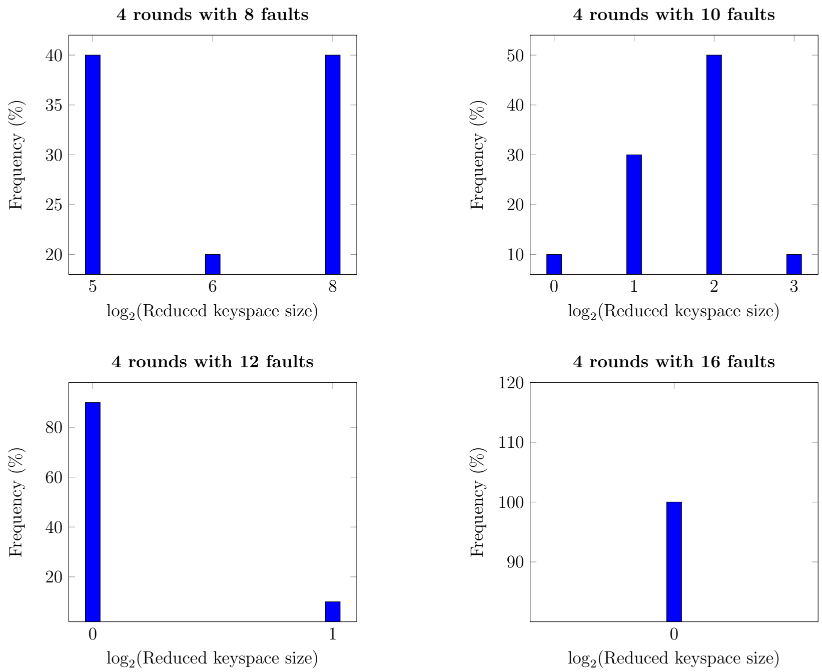

Fig. 2: Distribution of the Reduced Keyspace for the Fourth-to-Last Round Attack

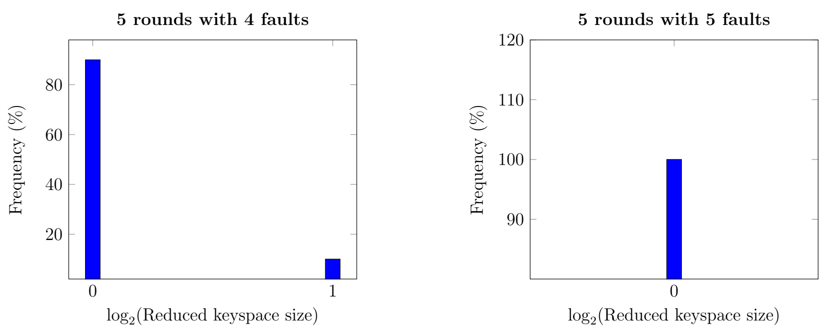

Fig. 3: Distribution of the Reduced Keyspace for the Fifth-to-Last Round Attack

pattern. Specifically, they are either $0, 1, 4, 5$ (faults at the left half) or $0, 2, 8, 10$ (faults at the right half) as shown in Figure 7.

This nibble difference pattern at the second-to-last round helps filter the ciphertext difference and trace it back to the input of the second-to-last round. Subsequently, the last three rounds of the computation trail (as described in

Algorithm 1) are applied to identify the unique differential trail. The process for computing the five rounds trail is presented in Algorithm 2.

Key Recovery. The deterministic computation of the five-round trail enables us to reduce the key space by evaluating each round individually based on the ciphertext difference. To recover the key, the initial step is to exhaustively evaluate each key nibble at the last round individually, effectively reducing the entire key space by up to 64 bits at the last round. Subsequently, we proceed to perform key space reduction for each group individually at the second-to-last round. This iterative process continues up to the fifth-to-last round, where we repetitively analyze and reduce the key space. By applying this method, we progressively narrow down the key space at each round, taking into account the induced faults, until we ultimately arrive at a unique solution based on the number of injected faults. In summary, by analyzing each round and reducing the key space iteratively, we can effectively narrow down the potential key candidates based on the induced faults in the differential trail computation. Our empirical validation strongly supports the notion that introducing a single bit-fault within each of the 8 groups $\mathcal{G}\mathrm{r}_i, i \in 0, 1, \cdots, 7$ of the SBox, achieved by flipping a bit at either index 1 or index 2, substantially reduces the keyspace, resulting in unique keys. For comprehensive details regarding the reduced keyspace values associated with varying fault injection counts, we refer to Table 10. Figure 3 shows the distribution of the size of the keyspace after this attack.

3.2 Attacks on Rotating Key Schedule

In the study presented in [20], the authors introduce the concept of computing an equivalent key, which generates the same ciphertext as the original key for a given plaintext. Building on this notion, the attacker's strategy involves computing an equivalent key for the DEFAULT-LAYER layer by injecting faults at various rounds. Subsequently, the attacker aims to recover the master key by executing a Differential Fault Analysis (DFA) on the DEFAULT-CORE.

This section begins by explaining how to derive an equivalent key for the DEFAULT-LAYER. We introduce additional methodologies for calculating an equivalent key based on specific properties of the linear structured SBox S. Using this equivalent key, we propose a comprehensive attack strategy based on our deterministic trail computation approach, facilitating the unique recovery of the DEFAULT cipher's key for different rounds amidst injected faults. This method not only boasts efficient offline computation capabilities but also requires significantly fewer faults compared to previous attacks. Additionally, we present a versatile attack approach applicable in scenarios where the cipher utilizes multiple round-independent keys.

3.2.1 Exploiting Equivalent Keys Due to the LS SBox, we know that for any $\alpha \in \mathcal{L}(S)$ $\exists \beta \in \mathcal{L}(S^{-1})$ such that $S(x \oplus \alpha) = S(x) \oplus S(\alpha) = S(x) \oplus \beta$, $\forall x \in \mathcal{F}_2^4$. Let us define $\mathcal{L}(S, S^{-1}) = \{(\alpha, \beta) : S(x \oplus \alpha) = S(x) \oplus \beta\} =$

Algorithm 2 DETERMINISTIC COMPUTATION OF FIVE ROUNDS DIFFERENTIAL TRAIL

Input: A list of ciphertext difference $\mathcal{L}_{\Delta C}$
Output: Lists of input-output differences $\mathcal{A}^{23}_{ID}, \mathcal{A}^{24}_{ID}, \mathcal{A}^{25}_{ID}, \mathcal{A}^{26}_{ID}$, & $\mathcal{A}^{27}_{ID}$

$\mathcal{L}_1 \leftarrow [\ ], \mathcal{L}_2 \leftarrow [\ ], \mathcal{A}^{23}_{ID} \leftarrow [[\ ], \mathcal{A}^{24}_{ID} \leftarrow [[\ ],[\ ]], \mathcal{A}^{25}_{ID} \leftarrow [[\ ],[\ ]], \mathcal{A}^{26}_{ID} \leftarrow [[\ ],[\ ]], \mathcal{A}^{27}_{ID} \leftarrow [[\ ],[\ ]]$
$T_1 = [0, 1, 4, 5], T_2 = [0, 2, 8, 10]$
$\mathcal{T} = [\ [(T_1)^8, (T_2)^8, (T_1)^8, (T_2)^8],\ [(T_2)^8, (T_1)^8, (T_2)^8, (T_1)^8]\ ]$ ▷ Input nibble differences at the second-to-last round correspond to faults at the left/right half
$\mathcal{L}_1 = \mathcal{L}_{\Delta C}$
$\mathcal{L}_1 = P^{-1}(\mathcal{L}_1)$ ▷ Invert through bit-permutation layer
for $i = 0$ to 31 **do** ▷ At the round R^{27}
  $\mathcal{A}^{27}_{ID}[1][i] = \mathcal{L}_1[i]$
for $j = 0$ to 1 **do** ▷ For each fault at the left/right half in the fifth-to-last round
  for $i = 0$ to 8 **do** ▷ For each group $\mathcal{G}r_i$ at R^{26}
    for $(\Delta_0, \Delta_1, \Delta_2, \Delta_3) \in S^{-1}(\mathcal{L}_1[i]) \times S^{-1}(\mathcal{L}_1[i+8]) \times S^{-1}(\mathcal{L}_1[i+16]) \times S^{-1}(\mathcal{L}_1[i+24])$ at round R^{27} **do**
      $\mathcal{L}_1[i] = \Delta_0, \mathcal{L}_1[i+8] = \Delta_1, \mathcal{L}_1[i+16] = \Delta_2, \mathcal{L}_1[i+24] = \Delta_3$
      $\mathcal{L}_1[j] = 0, j \notin \{i, i+8, i+16, i+24\}$
      $\mathcal{A}^{27}_{ID}[0] = \mathcal{L}_1$
      $\mathcal{L}_1 = P^{-1}(\mathcal{L}_1)$
      $\mathcal{A}^{26}_{ID}[1] = \mathcal{L}_1$
      for $(\Delta_0, \Delta_1, \Delta_2, \Delta_3) \in S^{-1}(\mathcal{L}_1[0+\alpha]) \times S^{-1}(\mathcal{L}_1[1+\alpha]) \times S^{-1}(\mathcal{L}_1[2+\alpha]) \times S^{-1}(\mathcal{L}_1[3+\alpha])$ at round R^{26} **do** ▷ $\alpha \leftarrow 4 * i$
        $\mathcal{L}_2[\alpha] = \Delta_0, \mathcal{L}_2[1+\alpha] = \Delta_1, \mathcal{L}_2[2+\alpha] = \Delta_2, \mathcal{L}_2[3+\alpha] = \Delta_3$
        $\mathcal{L}_2[j] = 0, j \notin \{\alpha, 1+\alpha, 2+\alpha, 3+\alpha\}$
        $\mathcal{A}^{26}_{ID}[0] = \mathcal{L}_2$
        if $(\Delta_0 \in \mathcal{T}[j][\alpha])$ & $(\Delta_1 \in \mathcal{T}[j][1+\alpha])$ & $(\Delta_2 \in \mathcal{T}[j][2+\alpha])$ & $(\Delta_3 \in \mathcal{T}[j][3+\alpha])$ **then**
          $\mathcal{L}_{\Delta C} = \mathcal{L}_2$
Compute the trail for other three rounds using Algorithm 1 and get $\mathcal{A}^{25}_{ID}, \mathcal{A}^{24}_{ID}$, and $\mathcal{A}^{23}_{ID}$
return the lists $\mathcal{A}^{27}_{ID}, \mathcal{A}^{26}_{ID}, \mathcal{A}^{25}_{ID}, \mathcal{A}^{24}_{ID}$ and $\mathcal{A}^{23}_{ID}$

$\{(0,0),(6,a),(9,f),(f,5)\}$. In another way, we can say that for any $(\alpha,\beta) \in \mathcal{L}(S,S^{-1})$, $\Pr[\alpha \rightarrow \beta] = 1$. Consider a toy cipher consisting of one DEFAULT-LAYER SBox with a key addition before and after: $y = S(x \oplus k_0) \oplus k_1$, where $k_0, k_1 \in \mathcal{F}_2^4$. Due to the LS SBox, we have for any $(\alpha,\beta) \in \mathcal{L}(S,S^{-1})$,

$$y = S(x \oplus (k_0 \oplus \alpha)) \oplus (k_1 \oplus \beta) = S(x \oplus k_0) \oplus \beta \oplus (k_1 \oplus \beta) = S(x \oplus k_0) \oplus k_1, \forall x \in \mathcal{F}_2^4.$$

This means that if (k_0, k_1) be the actual key used in the toy cipher, then for any $(\alpha,\beta) \in \mathcal{L}(S,S^{-1})$, $(\hat{k}_0, \hat{k}_1) = (k_0 \oplus \alpha, k_1 \oplus \beta)$ will also be an equivalent key of the toy cipher, i.e., the number of equivalent keys of this toy cipher will be 2^2. Similarly, any round function of DEFAULT cipher can be think of parallel execution of 32 toy ciphers. Let $k_0 = (k_0^0, k_0^1, \ldots, k_0^{31})$ and $k_1 = (k_1^0, k_1^1, \ldots, k_1^{31})$ denote the two keys before and after the SBox layer respectively. Then, $\forall$ linear structures $(\alpha^i, \beta^i), i \in \{0, 1, \ldots, 31\}$, the number of equivalent keys for the round function of DEFAULT cipher will be $2^{2\times 32} = 2^{64}$. The various methods for generating equivalent keys of the DEFAULT-LAYER are outlined in Algorithm 3 and Algorithm 4. The practical verification to compute the equivalent keys can be found in [1]. Thus, for the DEFAULT-LAYER with four keys (k_0, k_1, k_2, k_3) used in the three round functions, the number of equivalent keys $(\hat{k}_0, \hat{k}_1, \hat{k}_2, \hat{k}_3)$ will be $2^{3\times 64} = 2^{192}$. For example, the keys in Table 7 are equivalent keys and hence, generate the same ciphertext c corresponds to the message m. Since the keyspace of (k_0, k_1, k_2, k_3) used in the DEFAULT-LAYER is 2^{512} and it has 2^{192} number of equivalent keys for any choosen key, we can further divide the keyspace into $2^{512-192} = 2^{320}$ number of different equivalent key classes.

3.2.2 Generalized Attack Strategy In this approach, we exploit the fact that injecting two faults at each nibble position in the last round of the encryption process reduces the key nibble space from 2^4 to 2^2. We iteratively select one key nibble from each reduced set of key nibble values to obtain keys $\hat{k}_3$, $\hat{k}_2$, and $\hat{k}_1$. However, at the fourth-to-last round, the key nibbles of k_0 still have 2^2 possible choices. To compute $\hat{k}_0$, our strategy involves introducing additional faults at higher rounds and using the other keys $\hat{k}_3$, $\hat{k}_2$, and $\hat{k}_1$ in conjunction with the deterministic trail computation up to the fifth-to-last round. For instance, if we inject 32 faults at each nibble in the sixth-to-last round of DEFAULT-LAYER, we can trace back from the ciphertext difference to the fourth-to-last round output difference by applying the equivalent round keys $\hat{k}_3$, $\hat{k}_2$, and $\hat{k}_1$. Based on this fourth-to-last round difference, we can compute the trail for the upper three rounds (from fourth to sixth last rounds) using Algorithm 1.

In the case of the simple key schedule, we have demonstrated that around 32 faults at each nibble in the third-to-last round are adequate for unique key recovery. Similarly, in the scenario described in the previous section, we can uniquely retrieve the key $\hat{k}_0$ by injecting a suitable number of faults, such as around 12 or 5 faults at the seventh-to-last or eighth-to-last rounds, and deterministically computing the upper trails for four or five rounds using Algorithm 5 or Algorithm 2, respectively. To summarize, the first step requires approximately 256 faults to uniquely select $\hat{k}_3$, $\hat{k}_2$, and $\hat{k}_1$ from 2^{64} choices, along with k_0 having 2^{64} possibilities. The recovery of $\hat{k}_0$ can be accomplished by injecting just 5 extra single bit-flip faults at the eight-to-last round. Consequently, around 261 faults are needed to recover an equivalent key of DEFAULT-LAYER. Once the equivalent key is obtained, the original key can be recovered by injecting faults in the DEFAULT-CORE.

The aim is to explore alternative strategies that can effectively reduce the number of faults required, as opposed to the initial approach of injecting two faults at each nibble in the last four rounds. By leveraging deterministic trail computations, several strategies can be employed to achieve this reduction. These strategies are as follows:

Algorithm 3 COMPUTATION OF EQUIVALENT ROUND KEYS ACCORDING TO [20]

```
Input: k_seq[ ] = [[k3], [k2], [k1], [k0]]
Output: Return an equivalent key k_seq[]
for i = 0 to 2 do
    δ = [0, 0, . . . , 0]
    for j = 0 to 31 do
        for any (α, β) ∈ L(S, S^-1) do
            if ((α > 0) and (β > 0)) then
                k_seq[i][j] = k_seq[i][j] ⊕ α
                δ[j] = δ[j] ⊕ β
                break
    k_seq[i] = permute_bits(k_seq[i])
    for ℓ = 0 to 32 do
        k_seq[i + 1][ℓ] = k_seq[i + 1][ℓ] ⊕ δ[ℓ]
return key_seq[]
```

Algorithm 4 OTHER WAYS TO COMPUTE EQUIVALENT ROUND KEYS FOR DEFAULT-LAYER

```
Input: k_seq[ ] = [[k3], [k2], [k1], [k0]] and choose one element (p, q) from the Set S = {(0, 3), (4, 7), (8, 11), (12, 15)}
Output: Return an equivalent key k_seq[]
for i = 0 to 2 do
    δ = [0, 0, . . . , 0]
    for j = 0 to 31 do
        for any (α, β) ∈ L(S, S^-1) do
            x = k_seq[i][j] ⊕ α
            if p ≤ x ≤ q then
                k_seq[i][j] = k_seq[i][j] ⊕ α
                δ[j] = δ[j] ⊕ β
                break
    k_seq[i] = permute_bits(k_seq[i])
    for ℓ = 0 to 32 do
        k_seq[i + 1][ℓ] = k_seq[i + 1][ℓ] ⊕ δ[ℓ]
return key_seq[]
```

k_0	$1a5f01b35ef5deea60361f4df591c654$
k_1	$5a66c55f3847aed3025023785542a124$
k_2	$85cb6b4f87f44ed160d20d713c86144f$
k_3	$84c302e5cb1539af59d623e9acdae09d$

(a) Original Keys

$\hat{k}_0$	$7c3967d53893b88c0650792b93f7a032$
$\hat{k}_1$	$96aa0993f48b621fce9cefb4998e6de8$
$\hat{k}_2$	$4907a7834b38821dac1ec1bdf04ad883$
$\hat{k}_3$	$2e69a84f61bf9305f37c894306704a37$

(b) An Equivalent Keys

$\hat{k}_0$	$153f98d5310a481a0930e0bdfc61c95d$
$\hat{k}_1$	31210031003333000322101301033031
$\hat{k}_2$	12120210330102210232022122130120
$\hat{k}_3$	12320213202301003022231012132232

(c) An Equivalent Keys

$\hat{k}_0$	$153f98d5310a481a0930e0bdfc61c95d$
$\hat{k}_1$	57476657665555666544767567655657
$\hat{k}_2$	47475745665457745767577477465475
$\hat{k}_3$	47675746757654556577764547467767

(d) An Equivalent Keys

Table 7: An Example of Different Sets of Equivalent Keys

3.2.2.1 Retrieving Equivalent Key Using Three Round Trail Computation. It should be noted that a single bit-flip fault at any nibble can activate at least two nibbles in the next round. By injecting 32 faults at each nibble in the third-to-last round, we can generate at least two differences at each nibble in the second-to-last and last rounds. This allows us to compute $\hat{k}_3$ and $\hat{k}_2$. Then, by injecting another 32 faults at the fifth-to-last round, we can recover $\hat{k}_1$ and consider the 2^{64} choices of k_0 by computing three-round trails using $\hat{k}_3$ and $\hat{k}_2$. Finally, inducing another 32 faults at the sixth-to-last round, we obtain an equivalent key $(\hat{k}_0, \hat{k}_1, \hat{k}_2, \hat{k}_3)$. In summary, approximately 96 faults are required to recover an equivalent key for DEFAULT-LAYER.

3.2.2.2 Retrieving Equivalent Key Using Four Round Trail Computation. By injecting 32 single bit-flip fauts at each nibbles in the fourth-to-last round, we can achieve the generation of at least two different input differences at each nibble in the third-to-last, second-to-last and last rounds which can able to reduce the key nibble space to 2^2 individually. This enables the computation of $\hat{k}_3$, $\hat{k}_2$ and $\hat{k}_1$. Additionally, by introducing 8 faults at the seventh-to-last round, we can recover the 2^{64} choices of k_0 by utilizing four-round trails computed using $\hat{k}_3, \hat{k}_2$ and $\hat{k}_1$. Furthermore, approximately 8 faults at the eighth-to-last round are sufficient to obtain an equivalent key $(\hat{k}_0, \hat{k}_1, \hat{k}_2, \hat{k}_3)$. To summarize, a total of around 48 faults are required to recover an equivalent key for DEFAULT-LAYER.

3.2.2.3 Retrieving Equivalent Key Using Five Round Trail Computation. Like the previous approach, we inject 32 single bit-flip faults at each nibbles in the fifth-to-last round. This ensures the generation of at least two different input differences at each nibble in the fourth-to-last, third-to-last, second-to-last and last rounds respectively and then compute $\hat{k}_3, \hat{k}_2$, $\hat{k}_1$ and 2^{64} choices of k_0. Moreover, approximately 4 faults at the tenth-to-last round are sufficient to obtain an equivalent key $(\hat{k}_0, \hat{k}_1, \hat{k}_2, \hat{k}_3)$. As a result, a total of around 36 faults are required to recover an equivalent key for DEFAULT-LAYER. Figure 4 shows the distribution of the size of the keyspace after this attack.

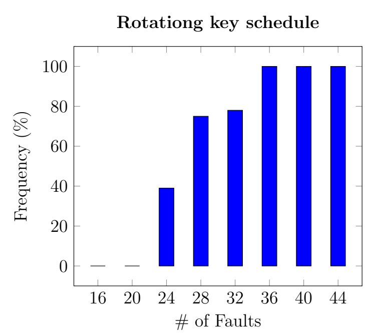

Fig. 4: Distribution of recovering an equivalent key for the rotating key schedule

3.2.3 Generic Attack Strategy for More Round Keys In the scenario where an DEFAULT-LAYER encryption consists of r rounds with $r+1$ round keys $k_0, k_1, \ldots, k_r$, a simple approach involves injecting two faults at each nibble in the encryption process for each of the r rounds. This allows us to compute r equivalent keys: $\hat{k}_r, \hat{k}_{r-1}, \ldots, k_1$. However, the initial key k_0 remains unknown due to the lack of input knowledge and the unavailability of additional DEFAULT-LAYER SBox to be faulted.

To recover the unknown key k_0, we target the last round of the DEFAULT-CORE and introduce faults individually to each SBox. This technique enables the unique retrieval of the key k_0. Once an equivalent key is determined, the original key can be obtained by applying the DFA to the DEFAULT-CORE.

To minimize the number of required faults, an efficient strategy involves injecting 8 faults at the fifth-to-last round, allowing the unique determination of $\hat{k}_r$ and k_{r-1}. This strategy is repeated iteratively until only three rounds remain. At this point, injecting 32 faults at the initial round of DEFAULT-LAYER facilitates the unique recovery of $\hat{k}_3$ and $\hat{k}_2$. Finally, injecting two faults at each nibble in the initial round yields the unique choice of $\hat{k}_1$. Subsequently, the DFA is applied to the DEFAULT-CORE to uniquely retrieve k_0.

3.3 Experimental Results on DEFAULT under DFA

In this attack scenario, we have conducted a comprehensive analysis for both the simple key schedule and the rotating key schedule of DEFAULT. For the simple key schedule, our estimations indicate that approximately 32, 34, 16, and 5 bit-faults are required to effectively reduce the key spaces to 2^{32}, 1, 1, and 1, respectively, under a differential fault attack (DFA). These faults are introduced at the second-to-last, third-to-last, fourth-to-last, and fifth-to-last rounds, respectively. Likewise, for the rotating key schedule, our estimates suggest that approximately 96, 48, and 36 bit-faults are necessary to recover the equivalent key for the DEFAULT-LAYER using DFA techniques when the faults are injected at the third-to-last, fourth-to-last, and fifth-to-last rounds, respectively. We have also rigorously validated the efficacy of Algorithm 3, 4 in computing equivalent keys for the DEFAULT-LAYER. Furthermore, we have determined that around 32 bit-faults at each SBox in the second-to-last round are sufficient to uniquely recover the key of DEFAULT-CORE. It is important to emphasize that all our findings and estimations have undergone rigorous practical experiments to ensure their validity and reliability. Detailed implementations of these attacks can be found in [1]. Our experiments were conducted on an Intel® Core™ i5-8250U computer. It is worth noting that employing more powerful computing hardware could potentially yield more accurate fault estimation results.

4 Introducing SDFA: Statistical-Differential Fault Attack on DEFAULT Cipher

In addition to Difference-based Fault Analysis (DFA), Statistical Fault Attack (SFA) is another powerful attack in the context of fault attacks and their analysis. SFA leverages the statistical bias introduced by injected faults and differs from previous attacks is that it only requires faulty ciphertexts, making it applicable in various scenarios compared to difference-based fault attacks. While the designers of the DEFAULT cipher claim that their proposed design can protect against DFA and any form of difference-based fault attacks, but they do not assert security against other fault attacks that exploit statistical biases in the execution. In such scenarios, the designers recommend for the adoption of specialized countermeasures designed to thwart Statistical Ineffective Fault Analysis (SIFA) [13,12] and Fault Template Attack (FTA) [10,25]. These countermeasures are recommended to mitigate the inherent risks associated with these specific types of attacks.

Although countermeasures against statistical ineffective fault attacks and fault template attacks can enhance the resilience of a cryptographic system, the absence of specific countermeasures against difference-based fault attacks leaves a potential vulnerability to bit-set faults. Bit-set faults involve intentional manipulations of individual or groups of bits, allowing attackers to strategically modify intermediate values or ciphertexts. Practical experiments [22,18] on a microcontroller demonstrated successful induction of bit-set faults using laser beams, with higher occurrence rates than bit-flip faults. Despite requiring expensive equipment, this method allows for precise fault injection in target location and timing, as shown in [29]. Without targeted countermeasures against difference-based fault attacks exploiting the propagation of differences through the algorithm, bit-set faults pose a potential risk of revealing sensitive information or compromising system security.

In this section, we introduce a new fault attack called SDFA, which combines DFA with SFA by inducing bit-set faults. The SDFA attack enables us to further reduce the number of faults required to recover the key compared to our proposed improved attacks for both simple and rotating key schedules. Additionally, we demonstrate the effectiveness of this attack in retrieving subkeys for rotating key schedules, even when all the subkeys are generated from a random source.

4.1 Learned Information via SDFA

In Section 2.4, we discussed the information learned from DFA and its relation to input-output differences in an SBox. In this section, we delve deeper into the connection between DFA and SFA when bit-set faults are introduced into the state. Specifically, we examine the scenario where four bit-set faults are applied to positions in the last round SBox, resulting in the unique recovery of the key nibble using SFA. Alternatively, by introducing a bit-set fault in a nibble, we can narrow down the key nibble space from 2^4 to 2^{4-t}, $1 \leq t \leq 2$. Our objective is to combine the power of SFA and DFA to uniquely recover the key nibble with fewer faults in a nibble.

Consider an SBox with inputs (u_0, u_1, u_2, u_3) and outputs (v_0, v_1, v_2, v_3). Given an input-output difference $\alpha \rightarrow \beta$ in the SBox, the set of possible output nibbles that satisfy the given differential can be represented as $\mathcal{D}_i \cup \mathcal{D}_j$, where $i, j \in 0, 1, 2, 3$. Now, let us assume an attacker injects a bit-set fault at the 0-th bit of the SBox, resulting in $u_0 = 1$, and the input difference $\alpha = 1$. Depending on the DDT table, this leads to either $\beta = 3$ or $\beta = 9$. Consequently, the set $(\mathcal{D})$ of outputs that satisfy the differential $\alpha \rightarrow \beta$ will be either $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_3$ for $\beta = 3$, or $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2$ for $\beta = 9$. Simultaneously, for SFA, the attacker can compute the set of outputs $\mathcal{I}$ that satisfy $u_i = 1$ by inverting the SBox using the faulty outputs, i.e., $\mathcal{I} = \{x : S^{-1}(x) \;\&\; 2^i = 2^i\}$.

To determine the intersecting nibbles between DFA and SFA, our objective is to identify the common nibble values from each of the four partition sets $\mathcal{D}_i$ for DFA. These sets are denoted as $\mathcal{H}_i$ and defined as $\mathcal{H}_i = \{x \in \mathcal{D} : S^{-1}(x) \;\&\; 2^i =$

$2^i\}$. Table 8 provides the sets $\mathcal{H}_i$ corresponding to different bit-sets at the i^{th} position. These sets $\mathcal{H}_i$ are obtained by identifying the common values found within the intersecting sets of $\mathcal{D}$ for DFA and $\mathcal{I}$ for SFA.

Finally, for each bit-set u_i in the SBox, if $\mathcal{D} = \mathcal{D}_p \cup \mathcal{D}_q, p, q \in \{0, \ldots, 3\}$ represents the set of outputs that satisfy the differential $\alpha \to \beta$, then the SDFA (Statistical-Differential Fault Attack) is defined as the set $\mathcal{Z}$ of possible outputs that satisfy the differential $\alpha \to \beta$, given by $\mathcal{Z} = \mathcal{D} \cap \mathcal{I} = \mathcal{H}_p \cup \mathcal{H}_q$. An example of the intersecting outputs obtained by performing SDFA under a bit-set fault at the second bit position in the SBox is presented in Example 1.

Now consider a toy cipher where given a message m, the ciphertext c is produced by $c = S(m) \oplus k$. From the above example, the attacker can learn the following two independent equations involving the key bits as follows:

$$k_0 \oplus k_2 = (c_0 \oplus c_2) \oplus (v_0 \oplus v_2) = c_0 \oplus c_2,$$
$$k_2 \oplus k_3 = (c_2 \oplus c_3) \oplus (v_2 \oplus v_3) = c_2 \oplus c_3 \oplus 1.$$

Likewise, for any S-box differential $\alpha \to \beta$ involving bit-sets in the SBox, the attacker can extract two independent equations that involve the key bits, thereby revealing two bits of information about that key nibble. Table 9 provides a comprehensive list of possible differentials under nibble bit-sets, along with their corresponding independent equations that can be derived through the SDFA attack. It is important to note that in the case of bit-set faults, if the targeted bit is already set to 1, no difference will be generated. In such cases, the DFA attack cannot be performed. However, the SFA attack can still be applied to reduce the key information by one bit. Therefore, even if bit-set faults fail to generate a difference, they can still contribute to the reduction of one key bit information.

Example 1. Let us consider the input-output difference $2 \to 7$ corresponding to the bit-set $u_1 = 1$ in an S-box. In this case, the set $\mathcal{D}$ of output differences corresponding to the DFA will be $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_2 = \{0, 5, a, f, 2, 7, 8, d\}$. Similarly, for SFA, the set $\mathcal{I}$ will be $\mathcal{I} = \{1, 5, 6, 7, 8, 9, a, e\}$. Therefore, the intersecting set $\mathcal{Z}$ is obtained as $\mathcal{Z} = \mathcal{D} \cap \mathcal{I} = \{5, a, 7, 8\}$. Alternatively, we can compute $\mathcal{H}_0 = \{5, a\}$ and $\mathcal{H}_2 = \{7, 8\}$, which are the sets of output differences in $\mathcal{D}$ that satisfy the condition $(S^{-1}(x)\ \&\ 2^i) = 2^i$. Then, the set $\mathcal{Z}$ can be expressed as $\mathcal{Z} = \mathcal{H}_0 \cup \mathcal{H}_2 = \{5, a, 7, 8\}$.

Bit-Set	$\mathcal{H}_0$	$\mathcal{H}_1$	$\mathcal{H}_2$	$\mathcal{H}_3$
$u_0 = 1$	$\{5, f\}$	$\{4, e\}$	$\{2, 8\}$	$\{3, 9\}$
$u_1 = 1$	$\{5, a\}$	$\{1, e\}$	$\{7, 8\}$	$\{6, 9\}$
$u_2 = 1$	$\{5, a\}$	$\{4, b\}$	$\{2, d\}$	$\{6, 9\}$
$u_3 = 1$	$\{5, f\}$	$\{1, b\}$	$\{2, 8\}$	$\{6, c\}$

Table 8: Set of Outputs of SBox under Bit-Sets

4.2 Attack on Simple Key Schedule

By analyzing the SBox-based toy cipher (Figure 5), we have discovered that a single bit-set at the SBox can effectively extract atmost two bits of information

Direction	Learned Expression							
	$u_0 = 1$		$u_1 = 1$		$u_2 = 1$		$u_3 = 1$	
	$1 \to 3$	$1 \to 9$	$2 \to 7$	$2 \to d$	$4 \to 7$	$4 \to d$	$8 \to 6$	$8 \to c$
Enc (S^{-1})	$\sum_{i=0}^{3} k_i$	$\sum_{i=0}^{3} k_i$	$k_0 \oplus k_2$	$k_0 \oplus k_1$	$k_0 \oplus k_1$	$k_0 \oplus k_2$	k_0	k_0
	$k_1 \oplus k_2 \oplus k_3$	k_0	$k_2 \oplus k_3$	$k_1 \oplus k_2 \oplus k_3$	$k_0 \oplus k_2$	$k_0 \oplus k_3$	$k_1 \oplus k_3$	$k_1 \oplus k_3$

Table 9: Learned Key-Information under Bit-Sets at SBox

from the key nibble. Additionally, from the insights provided in Table 9, we observe that any two bit-sets at the SBox can reduce atmost four bits of information, i.e., to generate four independent equations involving the key bits. This enables us to uniquely recover the key nibble. In the worst case, it can reduce atleast two bits of information for two bit-sets in a nibble.

If our focus is on the last round of the DEFAULT-LAYER, in the best case scenario we can achieve the unique recovery of each key nibble by injecting 2 faults (active bit-set faults). In the worst case, 4 bit-set faults ensure the unique key recovery of each key nibbles. This shows that around 64 active bit-set faults (in the best case) are required to retrieve the key uniquely. Whereas in the worst case scenario 128 active bit-set faults are sufficient to recover the key. However, to minimize the number of faults required, the attacker can strategically inject bit-set faults in the upper rounds.

4.3 Attack on Rotating Key Schedule

The rotating key schedule in DEFAULT-LAYER involves four keys, namely k_0, k_1, k_2, and k_3, which are used for each round in a rotating fashion. The master key k_0 serves as the initial key, and the other three keys are derived by applying the four unkeyed round function of DEFAULT-LAYER recursively. From the perspective of an attacker, if any one of the round keys is successfully recovered, it becomes possible to derive the remaining three keys using the key schedule function. In the case of DEFAULT-LAYER, the key k_3 is used in the last round. By injecting approximately three bit-set faults at each nibble in the last round, it is feasible to effectively retrieve the key k_3.

To summarize, a total of around 64 to 128 faults are required to recover the complete set of keys in DEFAULT-LAYER. This attack strategy leverages the relationship between the round keys and the rotating key schedule, allowing for the recovery of the master key and subsequent derivation of the other keys.

4.4 Generic Attack on Truely Independent Random Keys

In the scenario where the round keys in the DEFAULT cipher are genuinely generated from random sources rather than derived from a master key using recursive unkeyed round functions, the task of uniquely retrieving all the keys becomes

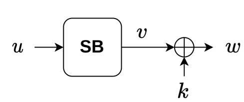

Fig. 5: Toy example of single SBox

considerably more challenging. In this case, both our DFA approach and the strategy presented in [20] face significant challenges in recovering keys uniquely and may require injecting a substantially larger number of faults compared to our SDFA approach.

Simply speaking, the SDFA approach involves injecting approximately three bit-set faults at each round of DEFAULT-LAYER and utilizing these faults to achieve unique key recovery. Thus, when the round keys are genuinely independent and not derived from a master key, this strategy proves to be much more effective than the DFA strategy. To provide a more concrete perspective, if DEFAULT employs a total of x $(x > 29)$ truly independent round keys, then approximately $x \times y,\ y \in [64, 128]$ bit-set faults are needed to recover all of its independent keys. This substantial increase in the number of required faults underscores the heightened difficulty of retrieving the keys when they are genuinely independent and not derived from a common source.

4.5 Experimental Results on DEFAULT under SDFA

We have performed an extensive analysis utilizing our novel attack strategy, SDFA, on both the simple key schedule and the rotating key schedule, considering the bit-set fault scenario. In the most favorable scenario for both key schedules, our estimations indicate that 64 active bit-set faults, with two faults introduced at each SBox, are adequate to uniquely recover the encryption key. Conversely, in the most challenging scenario, injecting 128 active bit-set faults at each SBox guarantees the unique key recovery. For complete implementation details of these attacks, we refer to [1]. The experiment was conducted on an Intel® Core™ i5-8250U computer.

5 Attacks on BAKSHEESH

For the BAKSHEESH cipher, despite the absence of any claimed DFA security by the designer, we conducted a thorough examination of its susceptibility to both Differential Fault Analysis (DFA) and Statistical-Differential Fault Analysis (SDFA) under bit-flip and bit-set fault scenarios, respectively. In this section, we will begin by outlining the differential fault attack, wherein we introduce faults at various rounds and determine the minimum number of faults required to achieve unique key recovery. Subsequently, we will present the SDFA attack and provide an estimate of the number of faults necessary to successfully retrieve the key in a unique manner.

5.1 DFA on BAKSHEESH

In this section, we outline our strategy for efficiently determining the differential trail up to three rounds to facilitate DFA attacks. We explain the trail computation process, its application in key retrieval via bit-flip faults, and estimate the fault complexity for key recovery in various rounds.

5.1.1 Faults at the Last Round

In our observations, injecting two faults at each nibble in the last round of BAKSHEESH yields three bits of information. Additionally, it is worth noting that the two key values corresponding to any two injected faults at the SBox are complementary to each other. The initial approach to reduce the key space involves inducing two bit-flip faults at each nibble in the last round before the SBox operation, individually affecting key nibbles, thus reducing the key space to 2^{32} with 64 faults in the last round. However, a more efficient strategy is required, inducing faults further from the last rounds, and deterministically obtaining information about the input differences for each SBox in the last round. This necessitates the development of a deterministic strategy capable of guessing the differential path from which the faults originate. In the upcoming subsections, we will demonstrate the feasibility of deterministically computing the differential path in BAKSHEESH for up to three rounds.

5.1.2 Faults at the Second-to-Last Round

The GIFT-128 permutation structure of the cipher permits a nibble difference at the input of group $\mathcal{G}\mathrm{r}_i$ in the second-to-last round to induce a bit difference in four nibbles in the last round. This observation allows an attacker to deterministically ascertain the differential path by introducing bit-flip faults at the second-to-last round. Furthermore, this insight enables the deterministic computation of differential paths for up to three rounds, as discussed in the next subsection. This is achievable because, for both non-faulty and faulty ciphertexts, the last round can be inverted by assessing input bit-differences at each nibble using DDT. The internal state difference can then be calculated by examining input bit-differences after the inverse operation of the second-to-last round, leveraging the Quotient-Remainder group structure.

A straightforward approach to attacking the cipher involves injecting two bit-faults at each nibble in the last round, thereby reducing the keyspace for each nibble to 2, resulting in an overall keyspace of 2^{32}. Subsequently, injecting one fault at each nibble in the second-to-last round uniquely reduces the keyspace. This naive approach necessitates approximately 96 faults for key recovery. However, we can enhance this attack by introducing faults at the second-to-last round during encryption. Our practical validation confirms that the introduction of one bit-faults at the second bit position in each SBox and two bit-faults at the third bit position in two different SBox at each group $\mathcal{G}\mathrm{r}_i$ at the second-to-last round, substantially diminishes the keyspace to nearly unique key. Detailed information on the reduced keyspace values corresponding to different fault injection counts is available in Table 11 (Appendix A).

5.1.3 Faults at the Third-to-Last Round

In this attack, we introduce bit-faults into a nibble during the third-to-last round of the cipher. Similar to the previous attack in DEFAULT-LAYER, we follow a deterministic process to calculate the input and output differences for each nibble at every round. This allows us to track how differences propagate throughout the cipher, as illustrated in Figure 6. Also, the three rounds trail computation is similar to Algorithm 1. We then leverage the computed trail to reduce the cipher's key space. By introducing two distinct bit differences in each nibble during the last round, we effectively reduce the key space to 2^{32}. Next, our focus narrows down to nibble positions 0, 1, 2, 3, 8, 9, 10, and 11 during the second-to-last round. We filter these nibble positions by iteratively inverting two rounds relative to combining the key spaces from nibble positions 0, 1, 2, 3, 8, 9, 10, 11, 16, 17, 18, 19, 24, 25, 26, and 27, all based on the key nibbles of the last round. Similarly, we filter nibble positions 20, 21, 22, 23, 28, 29, 30, and 31 by inverting two rounds with respect to combining the key spaces from nibble positions 4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, and 31, again based on the key nibbles of the last round. We subsequently perform further filtering on remaining nibble differences at the second-to-last round, considering the reduced key space for all 32 key nibble positions. Finally, we conduct additional filtering on nibble differences at the third-to-last round based on the further reduced key space. Our practical verification demonstrates that introducing two bit-faults at the third bit position in two different SBox within each group $\mathcal{G}\mathrm{r}_i$ during the third-to-last round significantly reduces the keyspace to a unique key. Comprehensive details regarding the reduced keyspace values for various fault injection counts can be found in Table 11.

5.2 SDFA on BAKSHEESH

The SBox employed in the BAKSHEESH cipher features a single non-zero LS element, denoted as 8. In the context of DFA, the key nibbles can be effectively reduced to one bit by introducing a minimum of two faults in each nibble. Notably, only introducing any two out of the three possible input differences (1, 2, and 4) at each SBox is sufficient to reduce the key nibbles to 2, given that 8 is a LS point.

Regarding SFA, our observations indicate that performing four SFA operations using bit-set faults can reduce the key nibbles to a minimum of 2. We have verified that introducing bit-set faults at each position within the SBox nibbles, with one active fault at the first three positions, is capable of uniquely reducing the key nibble space. Therefore, approximately 128 bit-set faults are sufficient for a nearly unique key recovery.

5.3 Experimental Results on BAKSHEESH

In this attack scenario, we have applied both our DFA and SDFA attack techniques to BAKSHEESH, achieving successful key recovery. In the DFA approach, our estimations suggest that approximately 48 and 16 bit-faults are needed to reduce the key spaces to $2^{0.2}$ and 1, respectively. These faults are strategically introduced at the second-to-last and third-to-last rounds. When it comes to the SDFA approach, our most favorable estimations indicate that 96 active bit-set faults, with three faults introduced at each SBox, are sufficient for a unique key recovery. In the worst-case scenario, injecting 128 active bit-set faults at each SBox guarantees a unique key recovery. For detailed implementations of these attacks, we refer to [1]. The experiments were performed on an Intel® Core™ i5-8250U computer. It is important to mention that employing more powerful computing hardware could potentially lead to more precise fault estimation results.

6 Discussion

This work presents enhanced DFA attacks on both LS SBox-based ciphers, DEFAULT and BAKSHEESH. The DEFAULT-LAYER SBox incorporates three non-trivial LS elements, while BAKSHEESH has only one non-trivial LS element. In our attack, we leverage deterministic trail computation for five rounds in the case of DEFAULT and three rounds for BAKSHEESH. This deterministic trail computation significantly reduces the number of required faults for key recovery. Therefore, exploring deterministic trail computation for a larger number of rounds could be an interesting avenue for future research.

Regarding the DEFAULT cipher, the designers claimed its DFA security to be 2^{64} under any difference-based fault analysis. In response, we introduce a new fault attack, the Statistical-Differential Fault Attack (SDFA), under the bit-set fault model. Our attack successfully recovers the unique keys of both DEFAULT and BAKSHEESH ciphers, even when the keys are independently drawn from random sources. This research highlights that without specific DFA protection, such ciphers are vulnerable to our proposed attacks. Furthermore, any difference-based countermeasures implemented against these ciphers contradict the design principles of cipher-level DFA protection. This suggests that employing linear-structured SBox-based cipher designs may not be advisable for achieving cipher-level DFA protection. Additionally, it would be interesting to investigate whether other attacks exploiting information leakages from statistical biases, such as Statistical Ineffective Fault Attack (SIFA) or Fault Template Attack (FTA), require fewer faults compared to difference-based fault analysis.

7 Conclusion

In light of the practical significance of Differential Fault Analysis (DFA) style attacks, the development of effective cipher protection strategies holds substantial relevance. Over recent years, various approaches and strategies have been explored to mitigate such vulnerabilities. Notably, the authors of the DEFAULT cipher have introduced a compelling design strategy aimed at intrinsically constraining the extent of information accessible to potential attackers. This innovative approach represents a notable contribution to the ongoing efforts to enhance cipher's DFA security.

In this study, we have presented an enhanced Differential Fault Attack (DFA) on the DEFAULT cipher, enabling the effective and unique retrieval of the encryption key. Our approach involves determining deterministic differential trails spanning up to five rounds and applying DFA by injecting faults at various rounds while quantifying the required number of faults. Specifically, for the simple key schedule, we demonstrate that approximately 5 bit-faults are sufficient to uniquely recover the key of DEFAULT. In contrast, for systems utilizing rotating keys, we show that approximately 20 bit-faults are required to recover the equivalent key of DEFAULT-LAYER. Remarkably, our attack achieves key recovery with a significantly reduced number of faults compared to previous methods.

Furthermore, we introduced a novel fault attack technique known as the Statistical-Differential Fault Attack (SDFA), which combines elements of both Statistical Fault Analysis (SFA) and DFA. In this attack, we demonstrate that at most 128 bit-set faults are sufficient to recover the key for both the key schedule configurations of the DEFAULT cipher. This attack highlights its efficacy in recovering encryption keys, not only for systems employing rotating keys but also for ciphers utilizing entirely round-independent keys.

Finally, we applied our proposed DFA attack to another linear-structured SBox-based cipher, BAKSHEESH, and efficiently recovered its master key uniquely. We show that approximately 16 bit-faults are required to achieve unique key recovery for BAKSHEESH. Similarly, under the bit-set fault model, the SDFA attack can be effectively applied to nearly retrieve its key uniquely by inducing 128 bit-set faults in the worst case.

In conclusion, our work makes significant contributions to the field of fault attacks by presenting enhanced DFA techniques, extending their applicability to rotating and round-independent keys, and introducing the SDFA approach. These advancements provide valuable insights into the vulnerabilities of the DEFAULT and BAKSHEESH ciphers and highlight the challenges in achieving effective DFA protection for linear-structured SBox-based ciphers. Our findings underscore the difficulty in achieving DFA protection for such ciphers and emphasize the need for enhanced security measures to safeguard encryption keys.

References

1. Attack Verification of DEFAULT and BAKSHEESH. https://drive.google.com/drive/folders/1aUOK_EnrOf5ICGOan9NkL3GIUbdiQanZ?usp=sharing
2. Agoyan, M., Dutertre, J., Mirbaha, A., Naccache, D., Ribotta, A., Tria, A.: How to flip a bit? In: 16th IEEE International On-Line Testing Symposium (IOLTS 2010), 5-7 July, 2010, Corfu, Greece. pp. 235–239. IEEE Computer Society (2010). https://doi.org/10.1109/IOLTS.2010.5560194, https://doi.org/10.1109/IOLTS.2010.5560194
3. Baksi, A., Bhasin, S., Breier, J., Jap, D., Saha, D.: A survey on fault attacks on symmetric key cryptosystems. ACM Comput. Surv. **55**(4), 86:1–86:34 (2023). https://doi.org/10.1145/3530054, https://doi.org/10.1145/3530054
4. Baksi, A., Bhasin, S., Breier, J., Khairallah, M., Peyrin, T., Sarkar, S., Sim, S.M.: DEFAULT: cipher level resistance against differential fault attack. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13091, pp. 124–156. Springer

(2021). https://doi.org/10.1007/978-3-030-92075-3_5, https://doi.org/10.1007/978-3-030-92075-3_5

5. Baksi, A., Bhasin, S., Breier, J., Khairallah, M., Peyrin, T., Sarkar, S., Sim, S.M.: DEFAULT: cipher level resistance against differential fault attack. IACR Cryptol. ePrint Arch. p. 712 (2021), https://eprint.iacr.org/archive/2021/712/1622193888.pdf
6. Baksi, A., Breier, J., Chattopadhyay, A., Gerlich, T., Guilley, S., Gupta, N., Hu, K., Isobe, T., Jati, A., Jedlicka, P., Kim, H., Liu, F., Martinasek, Z., Sakamoto, K., Seo, H., Shiba, R., Shrivastwa, R.R.: BAKSHEESH: similar yet different from GIFT. IACR Cryptol. ePrint Arch. p. 750 (2023), https://eprint.iacr.org/2023/750
7. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: A small present - towards reaching the limit of lightweight encryption. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10529, pp. 321–345. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_16, https://doi.org/10.1007/978-3-319-66787-4_16
8. Beierle, C., Leander, G., Moradi, A., Rasoolzadeh, S.: CRAFT: lightweight tweakable block cipher with efficient protection against DFA attacks. IACR Trans. Symmetric Cryptol. **2019**(1), 5–45 (2019). https://doi.org/10.13154/tosc.v2019.i1.5-45, https://doi.org/10.13154/tosc.v2019.i1.5-45
9. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Jr., B.S.K. (ed.) Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1294, pp. 513–525. Springer (1997). https://doi.org/10.1007/BFb0052259, https://doi.org/10.1007/BFb0052259
10. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2523, pp. 13–28. Springer (2002). https://doi.org/10.1007/3-540-36400-5_3, https://doi.org/10.1007/3-540-36400-5_3
11. Dey, C., Pandey, S.K., Roy, T., Sarkar, S.: Differential fault attack on DEFAULT. IACR Cryptol. ePrint Arch. p. 1392 (2021), https://eprint.iacr.org/2021/1392
12. Dobraunig, C., Eichlseder, M., Groß, H., Mangard, S., Mendel, F., Primas, R.: Statistical ineffective fault attacks on masked AES with fault countermeasures. In: Peyrin, T., Galbraith, S.D. (eds.) Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11273, pp. 315–342. Springer (2018). https://doi.org/10.1007/978-3-030-03329-3_11, https://doi.org/10.1007/978-3-030-03329-3_11
13. Dobraunig, C., Eichlseder, M., Korak, T., Mangard, S., Mendel, F., Primas, R.: SIFA: exploiting ineffective fault inductions on symmetric cryptography. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 547–572 (2018). https://doi.org/10.13154/tches.v2018.i3.547-572, https://doi.org/10.13154/tches.v2018.i3.547-572
14. Dutertre, J., Mirbaha, A., Naccache, D., Ribotta, A., Tria, A., Vaschalde, T.: Fault round modification analysis of the advanced encryption standard. In: 2012

IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2012, San Francisco, CA, USA, June 3-4, 2012. pp. 140–145. IEEE Computer Society (2012). https://doi.org/10.1109/HST.2012.6224334, `https://doi.org/10.1109/HST.2012.6224334`

15. Jana, A.: Differential fault attack on feistel-based sponge AE schemes. J. Hardw. Syst. Secur. **6**(1-2), 1–16 (2022). https://doi.org/10.1007/s41635-022-00124-w, `https://doi.org/10.1007/s41635-022-00124-w`
16. Jana, A., Paul, G.: Differential fault attack on photon-beetle. In: Chang, C., Rührmair, U., Mukhopadhyay, D., Forte, D. (eds.) Proceedings of the 2022 Workshop on Attacks and Solutions in Hardware Security, ASHES 2022, Los Angeles, CA, USA, 11 November 2022. pp. 25–34. ACM (2022). https://doi.org/10.1145/3560834.3563824, `https://doi.org/10.1145/3560834.3563824`
17. Jana, A., Saha, D., Paul, G.: Differential fault analysis of NORX. In: Chang, C., Rührmair, U., Katzenbeisser, S., Schaumont, P. (eds.) Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security Workshop, ASHES@CCS 2020, Virtual Event, USA, November 13, 2020. pp. 67–79. ACM (2020). https://doi.org/10.1145/3411504.3421213, `https://doi.org/10.1145/3411504.3421213`
18. Menu, A., Dutertre, J., Rigaud, J., Colombier, B., Moëllic, P., Danger, J.: Single-bit laser fault model in NOR flash memories: Analysis and exploitation. In: 17th Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2020, Milan, Italy, September 13, 2020. pp. 41–48. IEEE (2020). https://doi.org/10.1109/FDTC51366.2020.00013, `https://doi.org/10.1109/FDTC51366.2020.00013`
19. Moradi, A., Shalmani, M.T.M., Salmasizadeh, M.: A generalized method of differential fault attack against AES cryptosystem. In: Goubin, L., Matsui, M. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4249, pp. 91–100. Springer (2006). https://doi.org/10.1007/11894063_8, `https://doi.org/10.1007/11894063_8`
20. Nageler, M., Dobraunig, C., Eichlseder, M.: Information-combining differential fault attacks on DEFAULT. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13277, pp. 168–191. Springer (2022). https://doi.org/10.1007/978-3-031-07082-2_7, `https://doi.org/10.1007/978-3-031-07082-2_7`
21. Piret, G., Quisquater, J.: A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2779, pp. 77–88. Springer (2003). https://doi.org/10.1007/978-3-540-45238-6_7, `https://doi.org/10.1007/978-3-540-45238-6_7`
22. Roscian, C., Sarafianos, A., Dutertre, J., Tria, A.: Fault model analysis of laser-induced faults in SRAM memory cells. In: Fischer, W., Schmidt, J. (eds.) 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013. pp. 89–98. IEEE Computer Soci-

ety (2013). https://doi.org/10.1109/FDTC.2013.17, https://doi.org/10.1109/FDTC.2013.17

23. Saha, D., Chowdhury, D.R.: Scope: On the side channel vulnerability of releasing unverified plaintexts. In: Dunkelman, O., Keliher, L. (eds.) Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9566, pp. 417–438. Springer (2015). https://doi.org/10.1007/978-3-319-31301-6_24, https://doi.org/10.1007/978-3-319-31301-6_24
24. Saha, D., Chowdhury, D.R.: Encounter: On breaking the nonce barrier in differential fault analysis with a case-study on PAEQ. In: Gierlichs, B., Poschmann, A.Y. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9813, pp. 581–601. Springer (2016). https://doi.org/10.1007/978-3-662-53140-2_28, https://doi.org/10.1007/978-3-662-53140-2_28
25. Saha, S., Bag, A., Roy, D.B., Patranabis, S., Mukhopadhyay, D.: Fault template attacks on block ciphers exploiting fault propagation. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12105, pp. 612–643. Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_22, https://doi.org/10.1007/978-3-030-45721-1_22
26. Saha, S., Chakraborty, R.S., Nuthakki, S.S., Anshul, Mukhopadhyay, D.: Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability. In: Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings. pp. 577–596. Springer (2015). https://doi.org/10.1007/978-3-662-48324-4_29
27. Selmke, B., Brummer, S., Heyszl, J., Sigl, G.: Precise laser fault injections into 90 nm and 45 nm sram-cells. In: Homma, N., Medwed, M. (eds.) Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers. Lecture Notes in Computer Science, vol. 9514, pp. 193–205. Springer (2015). https://doi.org/10.1007/978-3-319-31271-2_12, https://doi.org/10.1007/978-3-319-31271-2_12
28. 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: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12105, pp. 581–611. Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_21, https://doi.org/10.1007/978-3-030-45721-1_21
29. Skorobogatov, S.: Optical fault masking attacks. In: Breveglieri, L., Joye, M., Koren, I., Naccache, D., Verbauwhede, I. (eds.) 2010 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2010, Santa Barbara, California, USA, 21 August 2010. pp. 23–29. IEEE Computer Society (2010). https://doi.org/10.1109/FDTC.2010.18, https://doi.org/10.1109/FDTC.2010.18

30. Tunstall, M., Mukhopadhyay, D.: Differential fault analysis of the advanced encryption standard using a single fault. IACR Cryptol. ePrint Arch. p. 575 (2009), http://eprint.iacr.org/2009/575

A Appendix

Attack Strategy	Results	
	Number of Faults	Reduced Key Space
Faults at the Second-to-Last Round	64	2^{32}
	48	2^{39}
	32	2^{46}
Faults at the Third-to-Last Round	32	$2^{0.2}$
	28	2^{7}
	24	2^{14}
Faults at the Fourth-to-Last Round	16	1
	12	1
	8	2^{7}
Faults at the Fifth-to-Last Round	8	1
	6	1
	5	1

Table 10: Keyspace Reduction with Varying Injected Faults in DEFAULT's Simple Key Schedule under Differential Fault Attacks

Attack Strategy	Results	
	Number of Faults	Reduced Key Space
Faults at the Second-to-Last Round	48	1
	40	1
	32	2^{32}
Faults at the Third-to-Last Round	16	1
	12	1
	10	2

Table 11: Keyspace Reduction with Varying Injected Faults in BAKSHEESH under Differential Fault Attacks

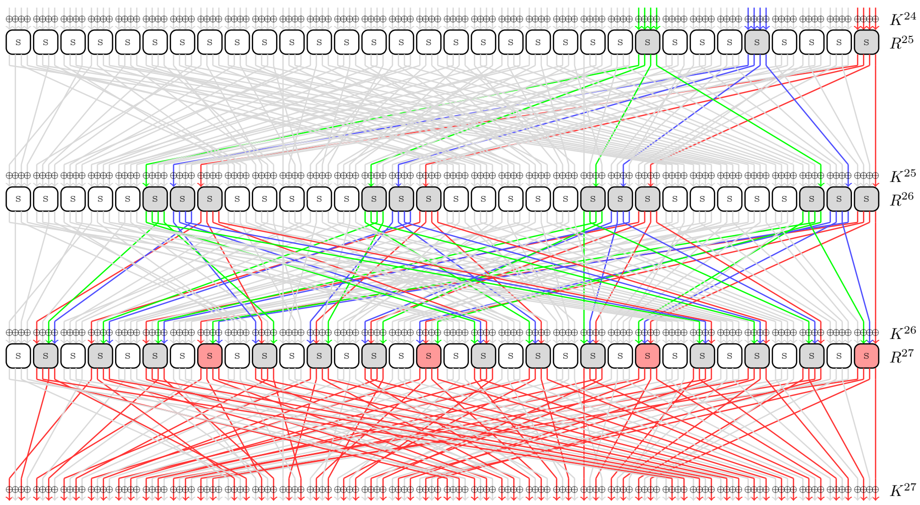

Fig. 6: Fault Propagation for Three Rounds

Algorithm 5 DETERMINISTIC COMPUTATION OF FOUR ROUNDS DIFFERENTIAL TRAIL

Input: A list of ciphertext difference $\mathcal{L}_{\Delta C}$
Output: Lists of input-output differences $\mathcal{A}^{24}_{ID}$, $\mathcal{A}^{25}_{ID}, \mathcal{A}^{26}_{ID}$, & $\mathcal{A}^{27}_{ID}$

```
Initialize 𝓛1 ← [ ], 𝓐24_ID ← [[ ], [ ]], 𝓐25_ID ← [[ ], [ ]], 𝓐26_ID ← [[ ], [ ]], 𝓐27_ID ← [[ ], [ ]]
𝓛1 = 𝓛ΔC
𝓛1 = P^-1(𝓛1)                                         ▷ Invert through bit-permutation layer
for i = 0 to 31 do                                     ▷ At the round R^27
    𝓐27_ID[1][i] = 𝓛1[i]
for i = 0 to 8 do                                      ▷ For each group 𝒢r_i at R^26
    for (Δ0, Δ1, Δ2, Δ3) ∈ S^-1(𝓛1[i]) × S^-1(𝓛1[i+8]) × S^-1(𝓛1[i+16]) × S^-1(𝓛1[i+24]) at round R^27 do
        𝓛1[i] = Δ0, 𝓛1[i+8] = Δ1, 𝓛1[i+16] = Δ2, 𝓛1[i+24] = Δ3
        𝓛1[j] = 0, j ∉ {i, i+8, i+16, i+24}
        𝓛1 = P^-1(𝓛1)
        if 𝓛1[j] = 0, ∀j ∈ {0. . . . , 31} \ {α, α+1, α+2, α+3} then      ▷ α ← 4 ∗ i
            if j ∈ {0, 1} then        ▷ j = 0/1 → injected faults at the left/right half of R^24
                if S^-1(𝓛1[α+j]) ∉ 𝒮 or S^-1(𝓛1[α+j+2]) ∉ 𝒮 then       ▷ 𝒮 ← {1, 2, 4, 8}
                    Break the for loop
        𝓐27_ID[0][i] = Δ0, 𝓐27_ID[0][i+8] = Δ1, 𝓐27_ID[0][i+16] = Δ2, 𝓐27_ID[0][i+24] = Δ3
        𝓛ΔC[i] = Δ0, 𝓛ΔC[i+8] = Δ1, 𝓛ΔC[i+16] = Δ2, 𝓛ΔC[i+24] = Δ3
Compute the trail for other three rounds using Algorithm 1 and get 𝓐26_ID, 𝓐25_ID and 𝓐24_ID
return the lists 𝓐27_ID, 𝓐26_ID, 𝓐25_ID and 𝓐24_ID
```

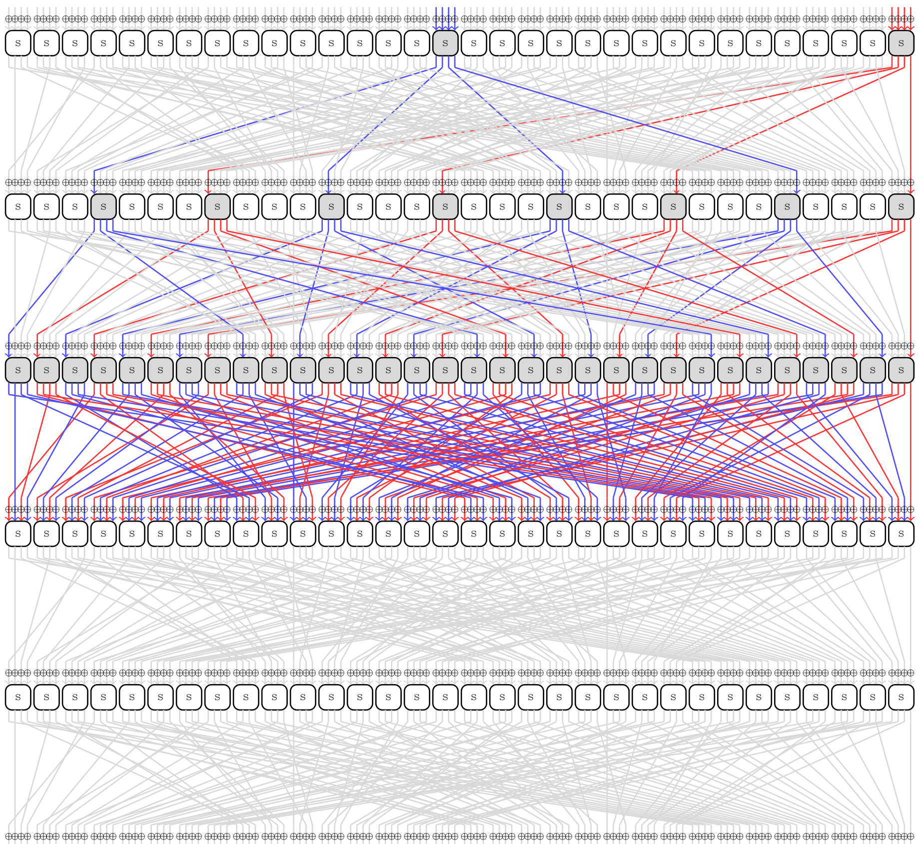

Fig. 7: Fault Propagation for Five Rounds