SDFA: Statistical-Differential Fault Attack on Linear Structured SBox-Based Ciphers

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.*. proposed DEFAULT, the first block cipher which provides differential fault attack (DFA) resistance at the algorithm level, with 64-bit DFA security. Initially, the cipher employed a simple key schedule where a single key was XORed throughout the rounds, and the key schedule was updated by incorporating round-independent keys in a rotating fashion. However, at Eurocrypt 2022, Nageler et al. presented a DFA that compromised the claimed DFA security of DEFAULT, reducing it by up to 20 bits for the simple key schedule and allowing for unique key recovery in the case of rotating keys. In this work, we present an enhanced differential fault attack (DFA) on the DEFAULT cipher, showcasing its effectiveness in uniquely recovering the encryption key. We commence by determining the deterministic computation of differential trails for up to five rounds. Leveraging these computed trails, we apply the DFA to the simple key schedule, injecting faults at different rounds and estimating the minimum number of faults required for successful key retrieval. Our attack achieves key recovery with minimal faults compared to previous approaches. Additionally, we extend the DFA attack to rotating keys, first recovering equivalent keys with fewer faults in the DEFAULT-LAYER, and subsequently applying the DFA separately to the DEFAULT-CORE. Furthermore, we propose a generic DFA approach for round-independent keys in the DEFAULT cipher. Lastly, we introduce a new paradigm of fault attack that combines SFA and DFA for any linear structured SBox based cipher, enabling more efficient key recovery in the presence of both rotating and round-independent key configurations. We call this technique Statistical-Differential Fault Attack (SDFA). Our results shed light on the vulnerabilities of the DEFAULT cipher and highlight the challenges in achieving robust DFA protection for linear structure SBox-based ciphers.

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 [BS97], DFA 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 [SC16,JSP20] still persists in designs with a parallelism degree greater than 2. Additionally, DFA [SC15,Jan22,JP22] 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 [SBD$^+$20] and CRAFT [BLMR19] are efficient and provide error detection. DEFAULT is a more radical approach, aiming to prevent DFA attacks through cipher-level design. A brief survey on fault attacks and their countermeasures in symmetric key cryptography can be found in [BBB$^+$23].

DEFAULT is a block cipher design proposed by Baksi *et al.* [BBB$^+$21a] 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 of substitution boxes (SBoxes) with linear structures, 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 combination of DEFAULT-LAYER and DEFAULT-CORE (where rounds are used non-linear structured 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 linear structures in the SBox of DEFAULT result in certain inputs/outputs being differentially equivalent, including the corresponding 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 of out of 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 [BBB+21b], the authorrs first propose the simple key schedule function where the master key is used throughout each rounds in the cipher. Then in [BBB+21a] the authors update the simple key schedule by recomending to use the rotating key schedule function in the cipher to make it more DFA secure cipher.

In [NDE22], 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.

In this work, we significantly enhance the DFA attack on the DEFAULT cipher, improving both the key schedule function and the efficiency of key retrieval. We demonstrate that our proposed attack can uniquely and efficiently recover the key, requiring fewer injected faults. Additionally, we illustrate that by combining information from both differential and statistical fault attacks under the bit-set fault model, we can effectively recover the keys in a unique manner. Moreover, we extend the applicability of this attack to any linear-structured SBox-based DFA protected ciphers, establishing its effectiveness in a broader context.

1.1 Our Contributions

In this paper, we make several contributions in the field of fault attacks on the DEFAULT cipher. 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 simple 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.

To summarize our contributions, we provide a brief comparison of the performance of our improved attacks with previous attacks in Table 1. Overall, our work significantly advances the state-of-the-art in fault attacks on the DEFAULT cipher and provides effective strategies for key recovery in linear-structured SBox-based ciphers.

Key Schedule	Works	Attack Strategy	Results		References
			# of Faults	Key Space	
Simple	Nageler *et al.*	Enc-Dec IC-DFA	16	2^{39}	[NDE22, Section 6.1]
		Multi-round IC-DFA	16	2^{20}	[NDE22, Section 6.2]
	Ours	Second-to-Last Round Attack	32	2^{32}	Section 4.1.2
		Third-to-Last Round Attack	32	2^{0}	Section 4.1.3
		Fourth-to-Last Round Attack	16	2^{0}	Section 4.1.4
		Fifth-to-Last Round Attack	8	2^{0}	Section 4.1.5
		SDFA	$[64, 128]$	2^{0}	Section 5.2
Rotating	Nageler *et al.*	Generic NK-DFA	$1728 + x$	2^{0}	[NDE22, Section 6.3]
		Enc-Dec IC-NK-DFA	$288 + x$	2^{32}	[NDE22, Section 6.3]
		Multi-round IC-NK-DFA	$(84 \pm 15) + x$	2^{0}	[NDE22, Section 6.3]
	Ours	Third-to-Last Round Attack	$96 + x$	2^{0}	Section 4.2.2.1
		Fourth-to-Last Round Attack	$48 + x$	2^{0}	Section 4.2.2.2
		Fifth-to-Last Round Attack	$24 + x$	2^{0}	Section 4.2.2.3
		SDFA	$[64, 128]$	2^{0}	Section 5.3

*x represents the number of faults to retrieve the key at the DEFAULT-CORE

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

1.2 Outline of the Paper

The remaining sections of the paper are organized as follows: Section 2 provides a brief overview of the DEFAULT cipher. Section 3 presents the background on differential fault attacks and reviews previous attacks on the DEFAULT cipher. In Section 4, we introduce our novel techniques for deterministic trail computation up to five rounds and demonstrate improved DFA attacks on both the simple and rotating key schedules. Additionally, we propose a new attack strategy called SDFA under the bit-set fault model for key recovery in the DEFAULT cipher. Finally, Section 7 concludes the paper with closing remarks.

2 Preliminaries

In this section, we introduce the notations that will be used throughout the paper, followed by a description of the DEFAULT cipher.

2.1 Notations

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.

2.2 Description of DEFAULT Cipher

The DEFAULT cipher [BBB+21a] 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.

2.2.1 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: $0 \to 0$, $6 \to a$, $9 \to f$, and $f \to 5$. The definition of a linear structure can be found in Definition 2. 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.

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 $\forall x \in \mathbb{F}_2^n$.*

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

2.2.2 Permutation Bits The DEFAULT cipher incorporates the GIFT-128 permutation in each of its rounds, which is derived from the GIFT cipher [BBC+23]. 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.

2.2.3 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.

2.2.4 Add Round Key The round key for GIFT-128 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. 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.

3 DFA on DEFAULT

In this section, we will provide a brief overview of the design principles behind DEFAULT, a type of encryption that is resistant to Differential Fault Analysis (DFA) attacks. We will also revisit an attack described in the research paper by Maria et al. [NDE22] and examine how it exposes the limitations of using a linear structure like SBox to design a cipher-level DFA-protected cipher.

3.1 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, eestimating 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.

3.2 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. Matematically, it is defined as follows.

Definition 2 (Linear Structure). *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a permutation. 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 $\forall x \in \mathbb{F}_2^n$.*

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 $\mathcal{L}(S^{-1}) = \{0, 5, a, f\}$.

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 4.

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

Table 4: Input Bit Relations of Partition Correspond to S/S^{-1}

For example, consider the SBox S^{-1} (for encryption) with differential $7 \to 2$. Then, the number of inputs that satisfy $7 \to 2$ 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 5, we summarize the key bits information for both $\mathsf{enc}/\mathsf{dec}$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 5: Learned Key-Information when faulting at (S/S^{-1})

3.3 Designer's Claim

In [BBB+21a], Baksi *et al.* proposed the $\mathsf{DEFAULT}$ cipher as a means of achieving DFA protection. The main idea behind this design is to allow faults to propagate through the cipher and produce faulty ciphertexts. However, the amount of information an attacker can learn from these ciphertexts is limited, meaning that the search complexity required to recover the secret component cannot be further reduced, even if additional faults are injected. Suppose an attacker inject faults (including bit-faults) in each nibbles before the SBox at the last round of the $\mathsf{DEFAULT\text{-}LAYER}$. Then, the attacker can only learn two bit information of each key nibbles. Mathematically, let $K^i = (k_0^i, k_1^i, k_2^i, k_3^i), i = 0, \ldots, 31$ denote the key nibbles XORed after the SBoxes at the last round of $\mathsf{DEFAULT\text{-}LAYER}$.

For each faults at the nibbles, the attacker can learn an equation rearding the key bits in that nibble. Consider the 0^{th} SBox where the attacker inject the difference of 2 at the last round. Then, the attacker can learn information of $k_0 \oplus k_2 = b$ (see Table 5), where b is known if the attacker knows the corresponding faulty and non-faulty nibbles. According to the Table 5, the attacker can learn atmost two independent equations of involving the bits in the key nibbles, i.e., the key nibble space can be reduced from 2^4 to atmost 2^2 even though the attacker can inject more than two faults at each nibbles.

The goal of the DEFAULT cipher is to provide a lower bound on the search complexity required to retrieve the secret component, thus increasing the difficulty of a successful DFA attack. According to its designers, the DEFAULT cipher can protect against DFA and other fault attacks that rely on differential values to deduce information from cipher executions. Its design limits the information attackers can obtain from faulty ciphertexts, making it difficult to compromise the security and confidentiality of the encrypted data. An interesting observation about DFA resistance in SBoxes is that if an SBox has a non-trivial linear structure, an attacker cannot guess the actual input even after injecting multiple faults. The attacker has to search exhaustively among the set of possible inputs to find the actual one. The DEFAULT cipher uses a linear structured SBox for DEFAULT-LAYER, which reduces the key space to 2^{64} bits after performing a DFA attack. However, this is still impractical for successful attacks, making the cipher resistant to DFA attacks.

3.4 Nageler *et al.*'s Work

In their work [NDE22], the authors demonstrated that an attacker could gather more key bits than the designer's claimed level of DFA-security by utilizing information from multiple rounds. In this attack, the attacker chooses a fault model to induce single bit-flip faults on the nibble between rounds and uses only difference-based evaluation to learn key information. They first show how an attacker can combine information from multiple differential fault attacks on the simple key schedule to learn more than half of the key bits using two approaches: combining information from encryption and decryption, or combining information from multiple consecutive rounds. Also, the authors propose a generic attack strategy for the strong key schedule used in DEFAULT-LAYER. They identify equivalent keys that produce the same permutation and recover a normalized version of the correct key, which can be applied to all DEFAULT-like ciphers with linearly structured SBoxes. Further, they improved this attack by combining the equivalence class of keys with the multi-round attack. In their improvements, the authors consider a differential model that predicts fault propagation and provide a strategy for storing the probability distribution of nibbles round by round. For a given fault, the strategy involves taking all possible fault propagations through rounds and calculating the probability distribution of each nibble by observing all possible differences in a round.

Encrypt-Decrypt Attack on Simple Key Schedule. In this attack, the attacker induces a difference of 2 and 8 in each S-box at the last round of DEFAULT-LAYER and respectively learns the key information $k_0 \oplus k_2$ and $k_0 \oplus k_1 \oplus k_2 \oplus k_3$ for each nibble of the inversely permuted key. Then, at the final decryption round, the attacker induces a difference of 2 in each S-box to learn $k_0 \oplus k_3$. The key is recovered by solving these 96 independent equations and iterating the remaining 2^{32} key candidates. Furthermore, the authors improved the attack by inducing 16 faults at the fifth-to-last round and were able to retrieve approximately 89 to 95 bits of the key in 95% of the cases.

Multi-Round Attack on Simple Key Schedule. In their simple attack, the authors injected bit-faults separately at the last three rounds. Specifically, they injected two faults in each nibble at the last round, then a single fault in each nibble at the second-to-last round, and finally 16 faults at specific bit positions in the third-to-last round. As a result, the attacker would need to perform a total of 112 faults to reduce the keyspace to 2^{16}. Furthermore, the authors improved the attack by inducing 16 faults at the specific bit positions in the fifth-to-last round and were able to reduce the key around 20-bits of the key in 90% of the cases.

Generic Attack Strategy for Ciphers with Linear Structures. Let us consider a one-round DEFAULT-LAYER SBox $\mathcal{S}$ and key addition before and after, where the output is given by $y = \mathcal{S}(x \oplus k_0) \oplus k_1$, and $(k_0, k_1) \in \mathcal{F}_2^4 \times \mathcal{F}_2^4$. If $\mathcal{S}$ is a linear structured S-box, then for any , $y \in \mathcal{L}(S) - \{0\}$, we have $\mathcal{S}(x \oplus (k_0 \oplus y)) \oplus (k_1 \oplus \mathcal{S}(y)) = \mathcal{S}(x \oplus k_0) \oplus \mathcal{S}(y) \oplus (k_1 \oplus \mathcal{S}(y)) = \mathcal{S}(x \oplus k_0) \oplus k_1$. This shows that the keys (k_0, k_1) and $(k_0 \oplus y, k_1 \oplus S(y)), y \in \mathcal{L}(S) - \{0\}$ are equivalent, i.e., $(k_0, k_1) \equiv (k_0 \oplus 6, k_1 \oplus a) \equiv (k_0 \oplus 9, k_1 \oplus f) \equiv (k_0 \oplus f, k_1 \oplus 5)$. These classes of equivalent keys can be further re-defined to normalized-keys as $(\bar{k}_0, \bar{k}_1) \in \mathcal{F}_2^4 \times \mathcal{N}, \mathcal{N} = \{0, 1, 2, 3\}$. Since the number of equivalent keys is 2^2, the key space of (k_0, k_1) can be divided into 2^6 number of equivalent classes. For DEFAULT-LAYER, there are 32 SBoxes with 2^2 linear structures each and hence, one round DEFAULT-LAYER has a linear space of size $\mathcal{N} = 2^{64}$ and hence, $\mid \mathcal{L} \mid = 2^{64 \cdot (n-1)}$, where $\mathcal{L}$ denotes the linear space of $n-1$ rounds. Let n be the number of subkeys used in a cipher, i.e., it has $n-1$ number of rounds. For DEFAULT-LAYER, $n - 1 = 28$. Then, the normalized key $\bar{K} = (\bar{K}_0, \bar{K}_1, \ldots, \bar{K}_{n-1}) \in \mathcal{N}^n$ can be defined as follows.

$$\mathcal{N}^n = \begin{cases} \mathcal{N} \times \mathcal{N} \times \ldots \mathcal{N} \times \mathcal{K} & \text{if } n = 28 \\ \mathcal{N} \times \mathcal{N} \times \ldots \mathcal{N} \times \mathcal{N} & \text{if } n < 28, \end{cases}$$

, where $K_{n-1} \in \mathcal{K}$ $(= \mathcal{F}_2^{4n})$ is the subkey in which no SBoxes are involved there.

Suppose a cipher has $n-1$ rounds with n independent keys used inside it. In such ciphers, the general strategy is to inject 64 faults at each round in the DEFAULT-LAYER to obtain equivalent keys and then convert them to normalized keys $\bar{K} \in \mathcal{N}$. Next, faults are injected at the last round DEFAULT-CORE before the SBox operation, and the key is recovered uniquely. This is because the DEFAULT-CORE uses stronger SBoxes, allowing the attacker to perform DFA using the normalized keys $\bar{K}$ and retrieve the key used between the DEFAULT-CORE and DEFAULT-LAYER.

Encrypt-Decrypt Attack on Rotating Key Schedule. The attack involves obtaining equivalent keys and injecting faults in both the decryption and encryption of DEFAULT-LAYER. The first step for the attacker is to construct the normalized key $\bar{K} = (\bar{K}_0, \bar{K}_1, \bar{K}_2, \bar{K}_3)$ by inducing 64 faults in each of the four rounds in decryption query. Then, the attacker injects 32 faults just before the SBoxes in the encryption to retrieve an additional 32 bits of keys for K_3. This reduces the key space of K_3 to 2^{32}, thereby decreasing the brute-force key complexity to 2^{32}.

Multi-Round Attack on Rotating Key Schedule. This attack involves finding the equivalence class of keys using a multi-round attack. The simple approach requires around 384 faults to uniquely recover the key. However, the authors have improved the attack, and now it only requires around 84 ± 15 faults to recover the key uniquely.

3.5 Dey *et al.*'s Work.

This paper [DPRS21] describes a differential fault attack on the initial version of the DEFAULT cipher, which used the master key throughout the rounds. The authors showed that an attacker can reduce the key complexity to around 2^{16} by injecting 112 faults at the second last round. However, this attack is not effective against the modified version of the cipher, which was published at Asiacrypt 2021 and uses a key scheduling algorithm. They showed that the modification of the cipher makes the attack ineffective, as the key scheduling algorithm adds more complexity to the key generation process, making it harder to predict the key even with the presence of faults.

4 Ours Improvements on DFA

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 or fourth-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 aims to strengthen the security of the DEFAULT cipher against DFA attacks and provide insights on how to improve the security of future designs against this type of attack.

4.1 Attacks on Simple Key Schedule

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.

4.1.1 Faults at the Last Round Based on the information learned from Table 5, 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 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 four rounds. By inducing around 8 bit faults at the fourth-to-last round, we estimate that the key space can be reduced to 2^{64} with greater efficiency than the naive approach.

4.1.2 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 four 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's worth noting that if faults are injected in more than one nibble in $\mathcal{G}\mathrm{r}_i$ at the second-to-last round, the keyspace for that group can be reduced further, potentially up to 2^4. Thus, the overall keyspace is now reduced to $2^{4\cdot 8} = 2^{32}$ (for 8 groups $\mathcal{G}\mathrm{r}_i$). Further, we can improve this attack by injecting faults upto the fifth-to-last round during encryption, which we will describe in the following sections.

4.1.3 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 of the DEFAULT cipher. We introduce a fault before the last three rounds of the cipher, specifically 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 given. 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.

Overall, this approach employs Difference-based Analysis (DFA) in two phases: firstly, by inducing faults and analyzing input/output differences, we determine the trail of three rounds; secondly, we utilize this computed trail to efficiently reduce the key space of the cipher, taking advantage of the established correlations between input-output differences and key bits. This method offers a more targeted and efficient approach to reducing the key space compared to the previous approaches discussed earlier.

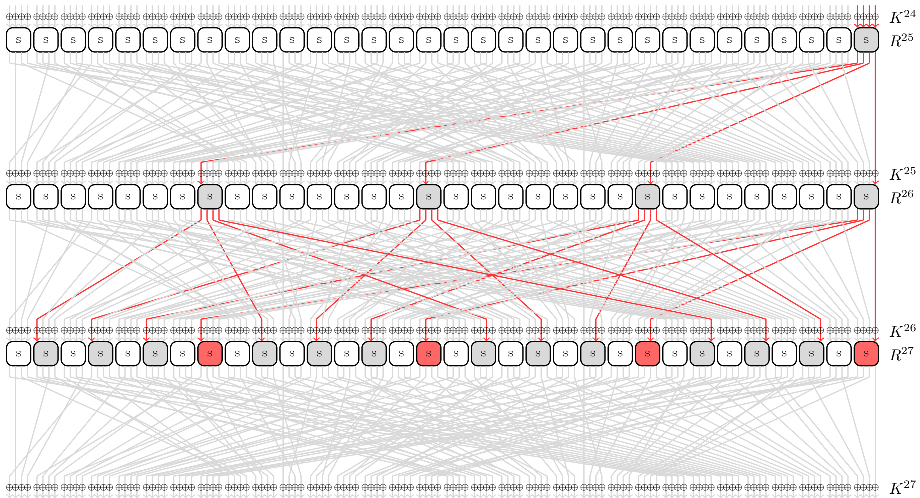

Fig. 1: Fault Propagation for the Three Rounds

Deterministic Trail Finding. We observe that a nibble difference at position $\mathcal{G}\mathrm{r}_i$ can activate the four nibbles at positions i, $i+8$, $i+16$, and $i+24$ after one round of the DEFAULT cipher. In other words, the nibble differences propagate to the groups $\mathcal{G}\mathrm{r}_{\frac{j}{4}}$, 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 $\mathcal{G}\mathrm{r}_{\frac{j}{4}}$, 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 1. 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 start by reducing the key space of the last round to 2^2 by comparing the non-faulty and faulty ciphertexts. By introducing two different bit differences at each nibble in the last round, we can effectively reduce the key space to 2^2. Next, we focus on each group $\mathcal{G}\mathrm{r}_i$, where i

Algorithm 1 Deterministic Computation of Three Rounds Differential Trail

Input: A list of ciphertext difference $\mathcal{L}_{\Delta C}$

Output: Lists of input-output differences $\mathcal{A}_{ID}^{25}, \mathcal{A}_{ID}^{26}$, & $\mathcal{A}_{ID}^{27}$

Initialize $\mathcal{L}_1 \leftarrow [\,], \mathcal{A}_{ID}^{25} \leftarrow [[\,],[\,]], \mathcal{A}_{ID}^{26} \leftarrow [[\,],[\,]], \mathcal{A}_{ID}^{27} \leftarrow [[\,],[\,]]$

$\mathcal{L}_1 = \mathcal{L}_{\Delta C}$

for $j = 0$ to 2 **do**

  $\mathcal{L}_1 = P^{-1}(\mathcal{L}_1)$ ▷ Invert through bit-permutation layer

  for $i = 0$ to 31 **do** ▷ At the round R^{27-j}

    $\mathcal{A}_{ID}^{27-j}[1][i] = \mathcal{L}_1[i]$

    if $\exists$ a difference $\beta \in \{1,2,4,8\} \ni \mathsf{DDT}^{-1}(\beta, \mathcal{L}_1[i]) \neq 0$ **then**

      $\mathcal{A}_{ID}^{27-j}[0][i] = \beta$

      $\mathcal{L}_1[i] = \beta$

return the lists $\mathcal{A}_{ID}^{27}, \mathcal{A}_{ID}^{26}$ and $\mathcal{A}_{ID}^{25}$

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 first/last fifteen nibbles of the third-to-last round can affect almost all the even/odd position nibbles in the last round.

By following this approach, we systematically reduce the key space by considering the differential trails and leveraging the relationships between input-output differences and key bits at different rounds of the cipher.

4.1.4 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 of the DEFAULT cipher. 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 (first 15 nibbles from the LSB) or right half (next 15 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 differences and key bits, we can significantly reduce the key space with a smaller number of injected faults compared to the previous attack.

Overall, this approach allows for the deterministic computation of the differential trail and presents a more efficient attack on the DEFAULT cipher by requiring fewer faults.

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 2.

Key Recovery. In the previous section, we discussed how to compute the unique trail from both non-faulty and faulty ciphertexts when faults are injected at 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.

By analyzing the input-output differences and iteratively refining the key space through the inversion of intermediate rounds, we can effectively narrow down the potential key candidates and approximate the correct key with a high level of confidence.

Experimental Verification. The question in this attack is how many faults are required to reduce the key space within practical limits? We estimate that approximately 20 bit faults are sufficient to recover the key uniquely.

4.1.5 Faults at the Fifth-to-Last Round In this section, we discuss how we can deterministically compute the differential trail when injecting faults during

Algorithm 2 Deterministic Computation of Four Rounds Differential Trail

Input: A list of ciphertext difference $\mathcal{L}_{\Delta C}$
Output: Lists of input-output differences $\mathcal{A}_{ID}^{24}$, $\mathcal{A}_{ID}^{25}, \mathcal{A}_{ID}^{26}$, & $\mathcal{A}_{ID}^{27}$
Initialize $\mathcal{L}_1 \leftarrow [\,], \mathcal{A}_{ID}^{24} \leftarrow [[\,],[\,]], \mathcal{A}_{ID}^{25} \leftarrow [[\,],[\,]], \mathcal{A}_{ID}^{26} \leftarrow [[\,],[\,]], \mathcal{A}_{ID}^{27} \leftarrow [[\,],[\,]]$
$\mathcal{L}_1 = \mathcal{L}_{\Delta C}$
$\mathcal{L}_1 = P^{-1}(\mathcal{L}_1)$ ▷ Invert through bit-permutation layer
for $i = 0$ to 31 **do** ▷ At the round R^{27}
    $\mathcal{A}_{ID}^{27}[1][i] = \mathcal{L}_1[i]$
for $i = 0$ to 8 **do** ▷ For each group $\mathcal{G}\mathrm{r}_i$ 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**
        $\mathcal{L}_1[i] = \Delta_0, \mathcal{L}_1[i+8] = \Delta_1, \mathcal{L}_1[i+16] = \Delta_2, \mathcal{L}_1[i+24] = \Delta_3$
        $\mathcal{L}_1[j] = 0, j \notin \{i, i+8, i+16, i+24\}$
        $\mathcal{L}_1 = P^{-1}(\mathcal{L}_1)$
        if $\mathcal{L}_1[j] = 0, \forall j \in \{0. \ldots, 31\} \setminus \{\alpha, \alpha+1, \alpha+2, \alpha+3\}$ **then** ▷ $\alpha \leftarrow 4 * i$
            if $j \in \{0, 1\}$ **then** ▷ $j = 0/1 \rightarrow$ injected faults at the left/right half of R^{24}
                if $S^{-1}(\mathcal{L}_1[\alpha + j]) \notin \mathcal{S}$ or $S^{-1}(\mathcal{L}_1[\alpha + j + 2]) \notin \mathcal{S}$ **then** ▷ $\mathcal{S} \leftarrow \{1, 2, 4, 8\}$
                    Break the for loop
        $\mathcal{A}_{ID}^{27}[0][i] = \Delta_0, \mathcal{A}_{ID}^{27}[0][i+8] = \Delta_1, \mathcal{A}_{ID}^{27}[0][i+16] = \Delta_2, \mathcal{A}_{ID}^{27}[0][i+24] = \Delta_3$
        $\mathcal{L}_{\Delta C}[i] = \Delta_0, \mathcal{L}_{\Delta C}[i+8] = \Delta_1, \mathcal{L}_{\Delta C}[i+16] = \Delta_2, \mathcal{L}_{\Delta C}[i+24] = \Delta_3$
Use Algorithm 1 to compute the unique trail for other three rounds and get the lists $\mathcal{A}_{ID}^{26}, \mathcal{A}_{ID}^{25}$ and $\mathcal{A}_{ID}^{24}$
return the lists $\mathcal{A}_{ID}^{27}, \mathcal{A}_{ID}^{26}, \mathcal{A}_{ID}^{25}$ and $\mathcal{A}_{ID}^{24}$

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 2.

Furthermore, the differences in even/odd nibbles at the second-to-last round activate all the nibbles in the fourth-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 second-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 second-to-last round follow a specific pattern. Specifically, they are either $0, 1, 4, 5$ or $0, 2, 8, 10$ (as shown in Figure 2).

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 3.

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 32 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.

Experimental Verification. In this attack we have observe that injecting around 8 bit-flip faults at the fifth-to-last round can reduce the key space almost uniquely.

4.2 Attacks on Rotating Key Schedule

In this section, we begin by explaining the computation of an equivalent key for the DEFAULT-LAYER layer. We outline the methodology to derive an equivalent key based on certain properties of the linear structured SBox S. Using this equivalent key, we propose a generalized attack that allows for the unique recovery of the DEFAULT cipher's key for different rounds in the presence of injected faults. Furthermore, we present a generic attack method that can be applied to retrieve the key when the cipher employs multiple round-independent keys.

4.2.1 Exploiting Equivalent Keys Due to the linear structured 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 $\mathcal{L}(S, S^{-1}) = \{(\alpha, \beta) : S(x \oplus \alpha) =$

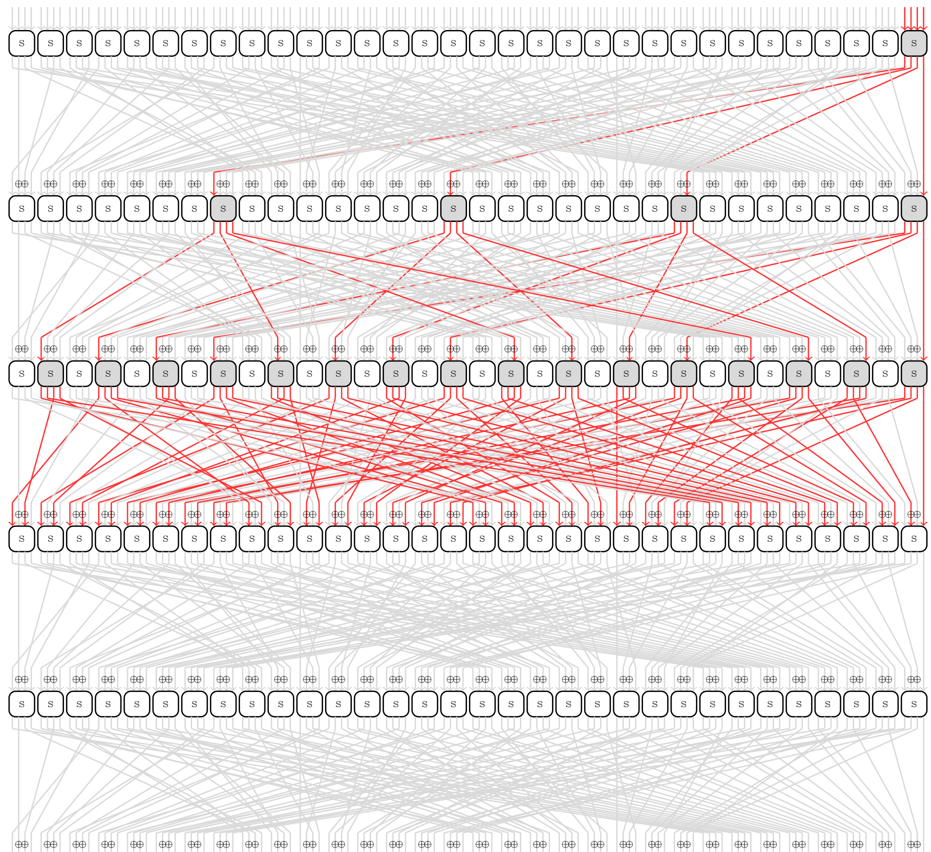

Fig. 2: Three Round Fault Propagation When injected at the Left Half in the Fifth-to-Last Round

$S(x) \oplus \beta\} = \{(0,0),(6,a),(9,f),(f,5)\}$. In another way, we can say that for any $(\alpha, \beta) \in \mathcal{L}(S, S^{-1})$, $\Pr[\alpha \to \beta] = 1$. 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 linear structured SBox, we have for any $(\alpha, \beta) \in \{(0,0),(6,a),(9,f),(f,5)\}$,

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

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

Algorithm 3 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 $\mathcal{G}\mathrm{r}_i$ 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**
          $\mathcal{L}_{\Delta C} = \mathcal{L}_2$
Use Algorithm 1 to compute the unique trail for other three rounds and get the lists $\mathcal{A}^{25}_{ID}, \mathcal{A}^{24}_{ID}$, 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 $\mathcal{A}^{23}_{ID}$

round function of DEFAULT cipher will be $2^{2\times 32} = 2^{64}$. The steps to generate an equivallent keys of DEFAULT-LAYER is given in Algorithm 4. 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 Figure 6 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

Algorithm 4 Compute Equivalent Round Keys for DEFAULT-LAYER

```
Input: key_seq[ ] = [[k_3], [k_2], [k_1], [k_0]]
Output: Return an equivalent key key_seq[]
for i = 0 to 2 do
    δ = [0, 0, ..., 0]                                  ▷ List of size 32
    for j = 0 to 31 do
        for any (α, β) ∈ L(S, S^{-1}) do
            key_seq[i] = key_seq[i] ⊕ α
            δ = δ ⊕ β
            break
    key_seq[i] = permute_bits(key_seq[i])
    for ℓ = 0 to 32 do
        key_seq[i + 1][ℓ] = key_seq[i + 1][ℓ] ⊕ δ
return key_seq[]
```

for any choosen key, we can further divide the keyspace into $2^{512-192} = 2^{320}$ number of different equivalent key classes.

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

(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 6: An Example of Different Sets of Equivalent Keys

4.2.2 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 fourth-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 above, we can uniquely retrieve the key $\hat{k}_0$ by injecting a suitable number of faults, such as around 16 or 8 faults at the seventh-to-last or eighth-to-last rounds, and deterministically computing the upper trails for four or five rounds using Algorithm 2 or Algorithm 1, 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 8 faults at the eight-to-last round. Consequently, around 264 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:

4.2.2.1 Retrieving Equivalent Key Using Three Round Trail Computation. It should be noted that a bit 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.

4.2.2.2 Retrieving Equivalent Key Using Four Round Trail Computation. By selecting the fault location at the fourth-to-last round, we can achieve the generation of at least two differences at each nibble in the second-to-last and last rounds with only around 16 faults. This enables the computation of $\hat{k}_3$ and $\hat{k}_2$. Additionally, by introducing 16 faults at the sixth-to-last round, we can recover $\hat{k}_1$ and consider the 2^{64} choices of k_0 by utilizing four-round trails computed using $\hat{k}_3$ and $\hat{k}_2$. Furthermore, approximately 16 faults at the seventh-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.

4.2.2.3 Retrieving Equivalent Key Using Five Round Trail Computation. To obtain equivalent keys $\hat{k}_3$ and $\hat{k}_2$ for DEFAULT-LAYER, we induce at least two differences in the second-to-last and last rounds using around 8 faults at the fifth-to-last round. By injecting 16 faults at the seventh-to-last round, we recover $\hat{k}_1$ and consider 2^{64} choices of k_0 using five-round trails computed with $\hat{k}_3$ and $\hat{k}_2$. Finally, around 8 faults at the eight-to-last round retrieve the equivalent key $(\hat{k}_0, \hat{k}_1, \hat{k}_2, \hat{k}_3)$. In total, about 24 faults are required.

4.2.3 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.

5 Introducing SDFA: Statistical-Differential Fault Attack

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 cases, the designers recommend using specialized countermeasures like Statistical Ineffective Fault Analysis (SIFA) and Fault Template Attack (FTA) to mitigate the risks associated with these 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. Without dedicated countermeasures against difference-based fault attacks, which exploit the propagation of differences through the algorithm, bit-set faults could potentially be exploited to reveal sensitive information or compromise the security of the system.

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.

5.1 Learned Information via SDFA

In Section 3.2, 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 α bit-set faults in a nibble, we can narrow down the key nibble space from 2^4 to $2^{4-\alpha}$. 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 $\alpha \to \beta$ 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 $\alpha \to \beta$ 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 7 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 8 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 7: Set of Outputs of SBox under Bit-Sets

Table 8: Learned Key-Information under Bit-Sets at SBox

Direction	Learned Expression			
	$u_0 = 1$	$u_1 = 1$	$u_2 = 1$	$u_3 = 1$
Enc (S^{-1})	$\sum_{i=0}^{3} k_i$ k_0	$k_0 \oplus k_2$ $k_2 \oplus k_3$	$k_0 \oplus k_2$ $k_2 \oplus k_3$	$\sum_{i=0}^{3} k_i$ $k_0 \oplus k_1 \oplus k_3$

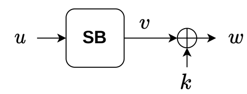

Fig. 3: Toy example of single SBox

5.2 Attack on Simple Key Schedule

By analyzing the SBox-based toy cipher (Figure 3), we have discovered that a single bit-set at the SBox can effectively extract atmost two bits of information from the key nibble. Additionally, from the insights provided in Table 8, 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.

5.3 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.

5.4 Generic Attack on Truely Independent Random Keys

In the scenario where the round keys in DEFAULT-LAYER are truly generated from random sources instead of being derived from a master key using recursive unkeyed round functions, the task of uniquely retrieving all the keys becomes significantly more challenging. In this case, both our DFA approach and the strategy presented in [NDE22] require injecting a considerably larger number of faults compared to our SDFA approach. The approach involves injecting approximately three bit-set faults at each round of DEFAULT-LAYER and leveraging the resulting faults to recover the key uniquely. However, when the round keys are truly independent and not derived from a master key, this strategy proves to be much effective compared to the DFA strategy. Consequently, a less number of faults need to be injected to achieve key recovery. To be more specific, if DEFAULT-LAYER utilizes a total of x truly independent round keys (where x is less than 28), then approximately $x \times y,\ y \in [64, 128]$ bit-set faults are required to recover all of its independent keys. This significant increase in the number of required faults emphasizes the increased difficulty in retrieving the keys when they are truly independent and not derived from a common source.

6 Discussion

In this section, we provide a brief discussion on the introduction of the Differential Fault Analysis (SDFA) technique for linear structured SBox-based ciphers. In the previous section, we demonstrated that by injecting two bit-set faults at the last round SBox in a specific manner, the corresponding key nibble can be uniquely recovered. This occurs when the actual bit value in the state changes from zero to one due to the fault injection. The resulting difference propagates through the ciphertext, allowing for the recovery of the key nibble. We classify bit-sets that generate differences in the ciphertext as active bit-sets, while those that do not are referred to as non-active bit-sets. Therefore, by performing two DFA + SFA operations individually on two active bit-sets in each SBox of the last round of the cipher, the key nibbles can be recovered uniquely. Additionally, we have observed that injecting four faults at each nibble is sufficient to recover the key uniquely using the SFA technique.

Recently, Baksi et al. proposed a new lightweight cipher called BAKSHEESH in their work [BBC+23]. BAKSHEESH is designed as a successor to the GIFT-128 cipher, with a smaller size (12.50% smaller) while maintaining the same security claims against classical attacks. The main differences between BAKSHEESH and GIFT-128 lie in the usage of a single non-trivial linear structured SBox in each round and the use of full-round key XOR instead of half-round key XOR in GIFT-128. In BAKSHEESH, each nibble key is reduced from 4 bits to 1 bit for the 1 non-trivial linear structured SBox, resulting in trivial DFA security with 2^{32}. However, it is claimed that the overall DFA security becomes 2^{64} when DEFAULT-CORE is replaced by the 10 rounds of the BAKSHEESH cipher in the DEFAULT design. So, in this case, the attack shown in [NDE22] cannot recover the key uniquely if the DEFAULT design use the rotating keys.

We have observed that in the case of the 1 non-trivial SBox in the BAKSHEESH cipher, injecting two active bit-set faults at each nibble reduces the key nibble from 4 bits to 1 bit, similar to the scenario of 3 non-trivial SBox in the DEFAULT cipher. As a result, the SDFA attack not only successfully recovers the key of the BAKSHEESH cipher but also efficiently recovers the keys of BAKSHEESH-based DEFAULT ciphers, regardless of whether they use simple or round-independent rotating keys. Furthermore, we claim that our DFA attack approach for the BAKSHEESH/BAKSHEESH-based DEFAULT cipher is capable of effectively and uniquely recovering the key for both simple and rotating keys.

Moreover, we have also demonstrated that both the DFA and SDFA attack approaches can efficiently recover the key for any linear structured SBox-based ciphers. This indicates that employing such linear structured SBox-based cipher designs may not be a good idea for achieving DFA protection.

7 Conclusion

In this work, we have presented an enhanced differential fault attack (DFA) on the DEFAULT cipher, which enables the effective and unique retrieval of the encryption key. Our approach involves determining the deterministic differential trails up to five rounds and then applying the DFA by injecting faults at various rounds, with a quantification of the required number of faults. Notably, our attack requires a significantly reduced number of faults compared to previous methods while achieving key recovery.

Furthermore, we have extended our DFA attack to handle rotating keys. By first recovering the equivalent keys using a smaller number of faults in the DEFAULT-LAYER and subsequently applying the DFA individually on the DEFAULT-CORE, we successfully retrieve the encryption key. Additionally, we have proposed a generic DFA approach for DEFAULT cipher instances utilizing round independent keys.

Moreover, we introduced a novel fault attack technique known as the statistical-differential fault attack (SDFA) that combines elements of both statistical fault analysis (SFA) and DFA. This attack demonstrates its efficacy in recovering encryption keys, not only for rotating keys but also for ciphers employing entirely round independent keys.

In conclusion, our work contributes 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 cipher and highlight the challenges in achieving effective DFA protection for linear structure Sbox-based ciphers. They emphasize the importance of implementing robust key protection mechanisms to address these vulnerabilities. Our findings underscore the difficulty in achieving DFA protection for such ciphers and reinforce the need for enhanced security measures to safeguard encryption keys.

References

[BBB+21a] Anubhab Baksi, Shivam Bhasin, Jakub Breier, Mustafa Khairallah, Thomas Peyrin, Sumanta Sarkar, and Siang Meng Sim. DEFAULT: cipher level resistance against differential fault attack. In Mehdi Tibouchi and Huaxiong Wang, editors, *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*, volume 13091 of *Lecture Notes in Computer Science*, pages 124–156. Springer, 2021.

[BBB+21b] Anubhab Baksi, Shivam Bhasin, Jakub Breier, Mustafa Khairallah, Thomas Peyrin, Sumanta Sarkar, and Siang Meng Sim. DEFAULT: cipher level resistance against differential fault attack. *IACR Cryptol. ePrint Arch.*, page 712, 2021.

[BBB+23] Anubhab Baksi, Shivam Bhasin, Jakub Breier, Dirmanto Jap, and Dhiman Saha. A survey on fault attacks on symmetric key cryptosystems. *ACM Comput. Surv.*, 55(4):86:1–86:34, 2023.

[BBC+23] Anubhab Baksi, Jakub Breier, Anupam Chattopadhyay, Tomas Gerlich, Sylvain Guilley, Naina Gupta, Kai Hu, Takanori Isobe, Arpan Jati, Petr Jedlicka, Hyunjun Kim, Fukang Liu, Zdenek Martinasek, Kosei Sakamoto, Hwajeong Seo, Rentaro Shiba, and Ritu Ranjan Shrivastwa. BAKSHEESH: similar yet different from GIFT. *IACR Cryptol. ePrint Arch.*, page 750, 2023.

[BLMR19] Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. CRAFT: lightweight tweakable block cipher with efficient protection against DFA attacks. *IACR Trans. Symmetric Cryptol.*, 2019(1):5–45, 2019.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[DPRS21] Chandan Dey, Sumit Kumar Pandey, Tapabrata Roy, and Santanu Sarkar. Differential fault attack on DEFAULT. *IACR Cryptol. ePrint Arch.*, page 1392, 2021.

[Jan22] Amit Jana. Differential fault attack on feistel-based sponge AE schemes. *J. Hardw. Syst. Secur.*, 6(1-2):1–16, 2022.

[JP22] Amit Jana and Goutam Paul. Differential fault attack on photon-beetle. In Chip-Hong Chang, Ulrich Rührmair, Debdeep Mukhopadhyay, and Domenic Forte, editors, *Proceedings of the 2022 Workshop on Attacks and Solutions in Hardware Security, ASHES 2022, Los Angeles, CA, USA, 11 November 2022*, pages 25–34. ACM, 2022.

[JSP20] Amit Jana, Dhiman Saha, and Goutam Paul. Differential fault analysis of NORX. In Chip-Hong Chang, Ulrich Rührmair, Stefan Katzenbeisser, and Patrick Schaumont, editors, *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security Workshop, ASHES@CCS 2020, Virtual Event, USA, November 13, 2020*, pages 67–79. ACM, 2020.

[NDE22] Marcel Nageler, Christoph Dobraunig, and Maria Eichlseder. Information-combining differential fault attacks on DEFAULT. In Orr Dunkelman and Stefan Dziembowski, editors, *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*, volume 13277 of *Lecture Notes in Computer Science*, pages 168–191. Springer, 2022.

[SBD⁺20] Thierry Simon, Lejla Batina, Joan Daemen, Vincent Grosso, Pedro Maat Costa Massolino, Kostas Papagiannopoulos, Francesco Regazzoni, and Niels Samwel. Friet: An authenticated encryption scheme with built-in fault detection. In Anne Canteaut and Yuval Ishai, editors, *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*, volume 12105 of *Lecture Notes in Computer Science*, pages 581–611. Springer, 2020.

[SC15] Dhiman Saha and Dipanwita Roy Chowdhury. Scope: On the side channel vulnerability of releasing unverified plaintexts. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, volume 9566 of *Lecture Notes in Computer Science*, pages 417–438. Springer, 2015.

[SC16] Dhiman Saha and Dipanwita Roy Chowdhury. Encounter: On breaking the nonce barrier in differential fault analysis with a case-study on PAEQ. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 581–601. Springer, 2016.