Differential Fault Attacks on TFHE-friendly cipher FRAST

Weizhe Wang and Deng Tang(✉)

Shanghai Jiao Tong University, Shanghai 200240, China
{SJTUwwz, dengtang}@sjtu.edu.cn

Abstract. Differential Fault Attacks (DFAs) have recently emerged as a significant threat against stream ciphers specifically designed for Hybrid Homomorphic Encryption (HHE). In this work, we propose DFAs on the FRAST cipher, which is a cipher specifically tailored for Torus-based Fully Homomorphic Encryption (TFHE). The round function of FRAST employs random S-boxes to minimize the number of rounds, and can be efficiently evaluated in TFHE. With our specific key recovery strategy, we can mount the DFA with a few faults. Under the assumption of precise fault injection, our DFA can recover the key within one second using just 4 or 6 faults. When discarding the assumption and considering a more practical fault model, we can still achieve key recovery in a few minutes without increasing the number of faults. To the best of our knowledge, this is the first third-party cryptanalysis on FRAST. We also explored countermeasures to protect FRAST. Our analysis revealed that negacyclic S-boxes, a key component of TFHE-friendly ciphers, are unsuitable for incorporating linear structures to resist DFA. Consequently, we recommend removing the negacyclic restriction in the penultimate round of FRAST and introducing non-zero linear structures into the S-boxes of the last two rounds. We believe that our work will provide valuable insights for the design of TFHE-friendly ciphers.

Keywords: Differential Fault Attack · Hybrid Homomorphic Encryption · Torus-based Fully Homomorphic Encryption · FRAST.

1 Introduction

Homomorphic Encryption (HE), a crucial cryptographic tool for protecting sensitive data in cloud computing, was first introduced by Rivest et al. [33] in 1978 and has been extensively studied since then. In traditional HE schemes, the bottlenecks are the heavy communication overhead caused by ciphertext expansion and the excessive computational burden on the client side. To address these issues, the transciphering framework, also known as Hybrid Homomorphic Encryption (HHE), was proposed. In HHE, the client only needs to encrypt data using a symmetric key cipher, and the server can then homomorphically decrypt it to obtain the HE ciphertext of the original data. To further enhance the performance of HHE, a wide range of HE-friendly ciphers have been developed over the past decade, such as LowMC [2], Kreyvium [11], FLIP [28], and Rasta [17], etc.

Torus-based Fully Homomorphic Encryption (TFHE), introduced by Chillotti et al. [12,13], boasts the fastest bootstrapping among Fully Homomorphic Encryption (FHE) schemes. A key feature of TFHE is its ability to homomorphically perform table lookup operations via Programmable Bootstrapping (PBS). This feature removes the limitation of the algebraic simplicity of nonlinear layers, enabling the design of new TFHE-friendly ciphers such as Elisabeth [16] and its patches [20], FRAST [15], and Transistor [7]. Proposed by Cho et al. at ToSC 2024, FRAST employs an effective unbalanced Feistel structure and uses random S-boxes in its round function. Compared to Kreyvium (resp. Elisabeth), FRAST achieves 2.768 (resp. 10.57) times higher throughput for TFHE keystream evaluation in the offline phase of the transciphering framework.

Despite their outstanding performance in HHE application, the novel and aggressive design strategies make HHE-friendly ciphers vulnerable to specific attacks, such as Differential Fault Attacks (DFA). DFA is a powerful cryptanalysis tool first proposed by Boneh et al. [9] in 1997. Unlike traditional cryptanalysis techniques, DFA can bypass the block-box assumption and exploit the cipher's weakness in the physical implementation. DFAs are often tailored to specific cryptographic primitives, requiring intricate knowledge of their structure and state propagation. Since its birth, DFA has been widely applied to analyze symmetric key ciphers such as AES [31,39,24], Trivium [21], and others [43,27,26,25].

Recent advancements in DFA research have extended its scope to HHE-friendly symmetric key ciphers. Roy et al. [35] pioneered the first DFA on Kreyvium and FLIP, later followed by Radheshwar et al. [32] targeting RASTA and FiLIP-DSM. Subsequent studies have expanded to attacks on RAIN and HERA [23], FLIP and FiLIP [29]. Most recently, Aikata et al. [1] introduced the SASTA-DFA framework, enabling DFA through high-level protocols and successfully applying it to Rubato, HERA, Pasta and Masta. Although TFHE-friendly ciphers have only been proposed recently, there has already been some related DFA research. Wang et al. [42] proposed a DFA on Elisabeth with a merge-and-intersect procedure. This attack was later optimized and generalized to a broad sub-family of ciphers following the group filter permutator paradigm, and subsequently applied to the patched versions of Elisabeth [41].

Since the introduction of DFA, how to protect ciphers against DFA has become a significant topic in cryptography. Common DFA countermeasures can be broadly categorized into four types: prevention, detection, infection and cipher-level protection. The first three approaches often lie outside the domain of cipher design, leaning more towards engineering-level solutions. Among these, the fault detection mechanisms based on error correction codes have gained considerable attention, with notable examples including CRAFT [8] and FRIET [40]. At ASIACRYPT 2021, Baksi et al. [5] proposed the first cipher-level DFA countermeasure, using linear structures in S-boxes to ensure exponential DFA search complexity. As a result, they introduced DEFAULT, a DFA-resistant cipher. Although DEFAULT was later broken by information-combined DFA [30,22], the attack required a strong fault model and a large number of faults. Hence, introducing linear structures in S-boxes remains a promising countermeasure in practical applications.

1.1 Our contributions and Organization of the Article

In this work, we analyze the security of the TFHE-friendly cipher FRAST against DFA under nibble fault model. We successfully mounted DFAs on FRAST with a few number of faults and discussed potential improvements to enhance its resistance against DFA. The main contributions of this paper are as follows:

– First, we investigate the propagation of single-nibble faults in one round of the FRAST. The 1-round differentials are classified into two categories (illustrated as Figure 3): faults occurring in the leftmost nibble ($X_1^{(i)}$) and faults occurring in other nibble ($X_{\geq 2}^{(i)}$). Using the first type of differentials, we can significantly reduce the size of candidate round key set. Under the assumption that the adversary can control the fault location, we propose the first DFA against FRAST. By injecting 2 faults (resp. 3 faults) into rounds 39 and 40, we are able to recover the master key with a probability of 85.1% (resp. 99.1%) in 0.27 s (resp. 3.49 s).
– Second, we discard the assumption of precise fault injection and consider a more practical random fault model. In the case of random fault locations, we increase the number of faults to obtain a sufficient number of first type of differentials, leading to our second DFA. To reduce the number of faults, we shift the fault injection one round earlier, increasing the effective fault ratio from $\frac{1}{32}$ to $\frac{31}{32}$. Based on the new differential equations, we develop the third DFA. By injecting 2 faults (resp. 3 faults) into rounds 38 and 49, we are able to recover the master key with a probability of 52% (resp. 75.2%) in 88.82 s (resp. 377.95 s).
– Finally, we explored how to improve FRAST to resist DFA. We focus on DEFAULT-like countermeasures and provide the definition of linear structures over groups. We prove that if a negacyclic S-box possesses a non-zero linear structure, it will degenerate into a binary function or a constant function (shown as Proposition 1). Hence, removing the negacyclic limitation in the penultimate round and introducing non-zero linear structures into the S-boxes of the last two rounds are the recommended countermeasures for FRAST. Since negacyclic S-boxes are a critical component of TFHE-friendly ciphers, this finding also provides valuable insights for designing DFA-resistant TFHE-friendly ciphers.

The summary of our results is provided in Table 1. We simulate DFAs using Python 3.9, and all the experiments are conducted on our workstation (2× Intel(R) Xeon(R) 5220R CPUs with 24 cores, running Ubuntu 20.04). Our implementation codes are given in Github repository.

The structure of the article is as follows. In Section 2, we introduce the notations, the background information of FRAST and DFA techniques. In Section 3, we analyze the 1-round differentials caused by a nibble fault, and present the

Table 1: Our DFA results on FRAST.

Cipher	Precise Injection?	Injected Faults		Success Rate	Average Success Time	Reference
		Numbers	Rounds			
FRAST	✓	(2,2)	(39,40)	85.1%	0.27 s	Section 3.2
		(3,3)		99.1%	3.49 s	
	✗	(96,96)	(39,40)	61.9%	1.44 s	Section 4.1
		(128,128)		81.9%	1.32 s	
		(2,2)	(38,39)	52.0%	88.82 s	Section 4.2
		(3,3)		75.2%	377.95 s	

first DFA under the precise fault injection assumption. In Section 4, we discard the assumption, give a naive extension from the first DFA, and reduce the number of faults by injecting earlier. Some possible countermeasures that help FRAST resist DFA are discussed in Section 5. Finally, we conclude the paper in Section 6.

2 Preliminaries

In this section, we will introduce the notations that will be utilized throughout the paper. Following that, we will provide a description of FRAST cipher and a simple introduction to DFA. The notations we used are listed as follows:

1. $\mathbb{G}_q, \mathbb{F}_q$ denote a group a finite field with q elements respectively. $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$.
2. $|A|$ denotes the number of elements in the set A.
3. $[a, b]$ denotes the set of integers $\{i|a \leq i \leq j\}$.
4. $+, -$ denotes the addition and subtraction of the integer.
5. $\cup, \cap$ denote the union and intersection of the set respectively.
6. $||$ denotes the concatenation of two vectors.
7. $RK^{(i)}$ denotes the ith-round key and $RK_j^{(i)}$ denotes the jth nibble of $RK^{(i)}$.
8. $X^{(i)}$ denotes the ith-round state and $X_j^{(i)}$ denotes the jth nibble of $X^{(i)}$. Moreover, $X_{\geq 2}^{(i)}$ denotes $X_k^{(i)}, k \in [2, 32]$.

2.1 Description of FRAST cipher

FRAST is a TFHE-friendly stream cipher proposed at ToSC 2024 [15]. The FRAST cipher with 128-bit security takes a 256-bit key $\boldsymbol{k} \in \mathbb{Z}_{16}^{64}$ and a 128-bit nonce nc $\in \{0, 1\}^{128}$ as input, and outputs a 128-bit keystream block $\boldsymbol{z} \in \mathbb{Z}_{16}^{32}$. The overall structure of FRAST is shown in Figure 1.

The design of FRAST adopts the generalized Feistel structure. Its round function consists of a contracting round function and an expanding round function.

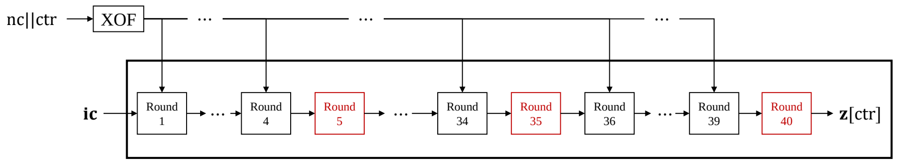


Fig. 1: The overall structure of FRAST in the counter mode, where $\boldsymbol{ic} = (0, 1, \ldots, 15, 0, 1, \ldots, 15) \in \mathbb{Z}_{16}^{32}$ is the input constant and $\boldsymbol{z}[\text{ctr}]$ is the keystream. The ith-round is a fixed round (highlighted in red) if i is a multiple of 5, and a random round otherwise.

In particular, for $X^{(i)} \in \mathbb{Z}_{16}^{32}$, the ith-round function $RF^{(i)}(X^{(i)}) = X^{(i+1)} \in \mathbb{Z}_{16}^{32}$ (see Figure 2) is defined as

$$\begin{aligned} X_j^{(i+1)} &= X_j^{(i)} + S_{\text{erf}}^{(i)}(X_1^{(i)} + RK_j^{(i)}) \text{ for } j \in [2, 32] \\ X_1^{(i+1)} &= X_1^{(i)} + S_{\text{crf}}^{(i)}(X_2^{(i+1)} + X_3^{(i+1)} + \cdots + X_{32}^{(i+1)} + RK_1^{(i)}), \end{aligned} \tag{1}$$

where $S_{\text{erf}}^{(i)}, S_{\text{crf}}^{(i)}$ are the S-boxes over $\mathbb{Z}_{16}$. The cipher employs two variants of

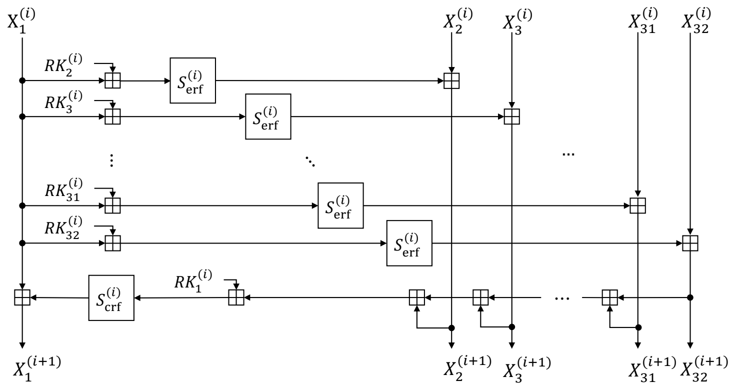


Fig. 2: The ith-round function of FRAST. For a random round, $S_{\text{erf}}^{(i)}, S_{\text{crf}}^{(i)}$ are generated by an extendable output function (XOF) with input nc||ctr.

round functions: randomized and fixed. Both share the same structure illustrated in Figure 2, differing only in their S-boxes. Specifically, FRAST repeats 4 random round functions followed by 1 fixed round function.

The $S_{\text{erf}}^{(i)}$ and $S_{\text{crf}}^{(i)}$ in random rounds are negacyclic[1] and generated as follows: the function value $(S_{\text{erf}}^{(i)}(0), \ldots, S_{\text{erf}}^{(i)}(7))$ and $(S_{\text{crf}}^{(i)}(0), \ldots, S_{\text{crf}}^{(i)}(7))$ are sampled by

[1] For a negacyclic look-up table L over $\mathbb{Z}_q$, we have $\forall i \in \mathbb{Z}_q, L(i + N) + L(i) = 0$.

the output from XOF with input nonce, and then the other function values are determined by the negacyclic property of the S-boxes. For the fixed rounds, FRAST use the same fixed S-box defined in Table 2 as $S^{(i)}_{\mathrm{erf}}$ and $S^{(i)}_{\mathrm{crf}}$.

Table 2: The fixed S-box used in the fixed rounds of FRAST.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S(x)$	0	3	5	8	6	9	12	7	13	10	14	4	1	15	11	2

The round keys $RK^{(i)}$ for $i = 1, 2, \ldots, 40$ are generated by multiplying the master key $\boldsymbol{k}$ with 64×64 invertible matrices over $\mathbb{Z}_{16}$. Specifically, the invertible matrix $\boldsymbol{M}$ is derived via SHAKE256 using a public seed. For $i = 1, 2, \ldots, 20$, round keys $RK^{(2i-1)}$ and $RK^{(2i)}$ are defined by $\boldsymbol{M}^i \boldsymbol{k} = RK^{(2i-1)} || RK^{(2i)}$. The full key schedule is detailed in [15]. The key schedule implies that if an attacker obtains two consecutive round keys $RK^{(2i-1)}$ and $RK^{(2i)}$, (s)he can recover the master key directly by multiplying $\boldsymbol{M}^{-1}$ multiple times.

2.2 Differential Fault Attacks

The DFA was first introduced by Boneh et al. [9] at EUROCRYPT 1997. Subsequently, Hoch and Shamir [19] extended DFA to stream ciphers at CHES 2004. In recent years, significant progress has been made in adapting DFA to HHE-friendly stream ciphers, demonstrating successful key recovery in multiple implementations [35,32,23,29,42,41].

During a DFA, the attacker is able to inject faults into the cipher's internal state, collect both correct and faulty outputs, and exploit these information to recover the secret key. Fault injection can be achieved via physical means such as laser shots, electromagnetic waves, unsupported voltage, etc. Generally, the basic underlying assumptions of the DFA model are listed as follows:

- The attacker can repeatedly restart the cipher using the same key and other public parameters (e.g., nonce and IV).
- The attacker can inject faults at specific timings during the keystream generation phase and monitor both normal and faulty keystreams.
- The attacker has the required tools for injecting faults.
- The number of injected faults must be kept minimal to prevent potential damage to the device.

The fault model plays an important role in how DFA works and what results it can achieve. The random word/nibble error model and the single bit flip model are two commonly used fault models. In the first model, a random word/nibble is added to an internal word/nibble of the cipher for the first model, while the second model flips a single bit in the state. After the fault injection, the attacker needs to identify the injected faults and recover the secret key with normal and faulty keystreams.

The nonce-based encryption scheme was formalized by Rogaway [34] and was shown to provide better resistance against DFA. However, researchers later found ways to bypass this protection. For example: nonce-misuse [38], nonce-bypass [37], internal DFA [36]. Previously, most DFAs on nonce-based HHE-friendly stream ciphers [35,32,23,29,42,41] all used nonce repetition. Recently, Aikata et al. [1] developed a new DFA framework called SASTA-DFA for HHE-friendly ciphers, which avoids nonce repetition. In this work, we assume the attacker can compute the difference between normal and faulty keystreams using nonce repetition or SASTA-DFA.

3 DFA with a Strong Assumption

In this work, we apply the random nibble error model as our fault model, i.e., after the fault injection, a random nibble is added to an internal nibble of the cipher. Due to the unbalanced Feistel network structure of FRAST, faults injected into the nibble x_1 lead to distinct patterns compared to other nibbles (as demonstrated in our later analysis), which critically impacts our DFA workflow. In this section, we assume the attacker can precisely control the location of faulty nibbles. Such a fault model requires sophisticated physical fault injection techniques but is still practical. Some examples using similar fault models with high temporal resolution can be found in [14].

Consider an S-box operating over $\mathbb{Z}_q$. Given an input difference α and corresponding output difference β — forming a differential trail $\alpha \to \beta$ — we define the filtered input set as

$$A(S, \alpha, \beta) = \{x \in \mathbb{Z}_q | S(x + \alpha) - S(x) = \beta\},$$

where the addition operator $(+)$ is defined over $\mathbb{Z}_q$ rather than bitwise XOR. Given the S-box and (α, β), the filtered input set can be computed with $O(q^2)$ time complexity.

By intersecting filtered sets derived from multiple differential pairs (α_i, β_i), i.e., $\cap_i A(S, \alpha_i, \beta_i)$, an attacker can (uniquely) determine the S-box's input value. This foundational relationship forms the basis of our DFA methodology for recovering round keys.

3.1 Two Types of 1-round Differentials

To mount a DFA on FRAST, we need to investigate how the difference propagates through 1-round FRAST at first.

Suppose a fault is injected at $X_1^{(i)}$, according to Equation (1), we can compute the output difference of the round function $\varDelta X^{(i+1)}$ as

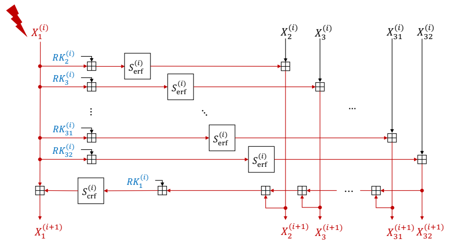
$$\begin{aligned}
\varDelta X_j^{(i+1)} &= \varDelta S_{\mathrm{erf}}^{(i)}(X_1^{(i)} + RK_j^{(i)}), j \in [2, 32] \\
\varDelta X_1^{(i+1)} &= \varDelta X_1^{(i)} + \varDelta S_{\mathrm{crf}}^{(i)}(X_2^{(i+1)} + X_3^{(i+1)} + \cdots + X_{32}^{(i+1)} + RK_1^{(i)}).
\end{aligned} \tag{2}$$

In this scenario, the differential is illustrated in Figure 3a. All nibbles of $X^{(i+1)}$ become active, and the resulting differential equations contain all round key nibbles.

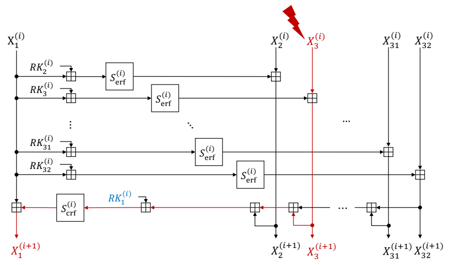
When a fault is injected at $X_k^{(i)}, k \in [2, 32]$, we can compute the output difference $\Delta X^{(i+1)}$ as

$$\begin{aligned} \Delta X_k^{(i+1)} &= \Delta X_k^{(i)}, \Delta X_j^{(i+1)} = 0, j \in [2, 32], j \neq k, \\ \Delta X_1^{(i+1)} &= \Delta S_{\text{crf}}^{(i)}(X_2^{(i+1)} + X_3^{(i+1)} + \cdots + X_{32}^{(i+1)} + RK_1^{(i)}). \end{aligned} \tag{3}$$

In this scenario, the differential is illustrated in Figure 3b. Only two nibbles of $X^{(i+1)}$, $X_1^{(i+1)}$ and $X_k^{(i+1)}$, become active, and the resulting differential equations contain only one round key nibble, $RK_1^{(i)}$.



(a) Fault injected at $X_1^{(i)}$



(b) Fault injected at $X_{\geq 2}^{(i)}$

Fig. 3: Two types of 1-round differentials. Affected state nibbles are highlighted in red, while involved key nibbles are highlighted in blue.

From the above analysis, when recovering round keys using 1-round differentials, the first differential plays a significantly more critical role than the second one due to the linear layer of the round function. Therefore, in our first DFA on FRAST, we will primarily use the first type of 1-round differential, i.e., faults injected into the leftmost nibble ($X_1^{(i)}$) of the round state.

3.2 First DFA: Injecting Faults at the Leftmost Nibble

In our first DFA on FRAST, we assume that all faults are injected at the leftmost nibble of the round state. For this DFA, we begin by injecting faults into the penultimate round of the cipher and utilize these faults to recover or filter the round key of the final round $RK^{(40)}$. The process of recovering the round key with the fault is illustrated in the Figure 4. It is important to note that this path represents the optimal recovery path for DFAs on FRAST, which is derived from FRAST's round function and 1-round differentials. Selecting any other path would expand the guessing space for intermediate variables, thereby increasing the time complexity required for key recovery. In the following part of this section, we provide a detailed explanation of the entire process.

$$\begin{tikzcd}
X^{(41)}, \Delta X^{(41)} \arrow[r, "①"] & \Delta X_1^{(40)} \arrow[r, "②"] & RK_1^{(40)} \arrow[r, "③"] & X_1^{(40)} \arrow[r, "④"] & RK_{2-32}^{(40)} \arrow[r, "⑤"] & X_{2-32}^{(40)}
\end{tikzcd}$$

① $\Delta X_j^{(41)} = \Delta S_{\text{erf}}^{(40)}\left(X_1^{(40)} + RK_j^{(40)}\right), j \in [2,32]$ ② $\Delta X_1^{(41)} = \Delta X_1^{(40)} + \Delta S_{\text{crf}}^{(40)}(X_2^{(41)} + \cdots + X_{32}^{(41)} + RK_1^{(40)})$

③ $X_1^{(41)} = X_1^{(40)} + S_{\text{crf}}^{(40)}(X_2^{(41)} + \cdots + X_{32}^{(41)} + RK_1^{(40)})$ ④ $\Delta X_j^{(41)} = \Delta S_{\text{erf}}^{(40)}\left(X_1^{(40)} + RK_j^{(40)}\right), j \in [2,32]$

⑤ $X_j^{(41)} = X_j^{(40)} + S_{\text{erf}}^{(40)}\left(X_1^{(40)} + RK_j^{(40)}\right), j \in [2,32]$

Fig. 4: The key recovery path using the fault $\Delta X_1^{(40)}$, with the equations required for each step. Known information is highlighted in red.

Given multiple different output differences of an S-box, we can uniquely determine the input difference by intersecting the sets of possible input differences corresponding to these output differences. A small example illustrating this process is provided in Appendix A. From Equation (2), we can derive 31 differential equations with input difference $\Delta X_1^{(i)}$ through the round function. Therefore, we can use $\Delta X_{2-32}^{(41)}$ to uniquely determine the fault value $\Delta X_1^{(40)}$.

After determining ΔX_1^{40}, we can use equation $\Delta X_1^{(41)} = \Delta X_1^{(40)} + \Delta S_{\text{crf}}^{(40)}(X_2^{(41)} + \cdots + X_{32}^{(41)} + RK_1^{(40)})$ to filter the candidate value for $RK_1^{(40)}$. At this stage, the input difference of the S-box $S_{\text{crf}}^{(40)}$ can be calculated as $\sum_{j=2}^{32} \Delta X_j^{(41)}$, and the output difference is $\Delta X_1^{(41)} - \Delta X_1^{(40)}$. By subtracting $\sum_{j=2}^{32} X_j^{(41)}$ from each element in the filtered input set $A(S_{\text{crf}}^{(40)}, \sum_{j=2}^{32} \Delta X_j^{(41)}, \Delta X_1^{(41)} - \Delta X_1^{(40)})$, we obtain the candidate set for $RK_1^{(40)}$.

Given $RK_1^{(40)}$, $X_1^{(40)}$ can be calculated as $X_1^{(41)} - S_{\text{crf}}^{(40)}(X_2^{(41)} + \cdots + X_{32}^{(41)} + RK_1^{(40)})$. Next, the candidate set for $RK_j^{(40)}$ can be derived from the input filtered set $A(S_{\text{erf}}^{(40)}, \Delta X_1^{(40)}, \Delta X_j^{(41)})$ and $X_1^{(40)}$ for all $j \in [2, 32]$. At this point, we have successfully filtered the candidate set for $RK^{(40)}$. Additionally, the intermediate state $X^{(40)}$ can be computed by applying the inverse of the round function. The pesudo-code for recovering the ith-round key is provided in Algorithm 1.

It is important to note that the filtering procedure described above typically results in a candidate set rather than a unique value when only a single fault is injected. A natural approach would be to verify each candidate round key until the correct one is found. However, such verification is unfeasible because the encryption function cannot be simulated with only $RK^{(40)}$. The master key is necessary to simulate the FRAST cipher. Due to the key schedule of FRAST, the master key can be retrieved only when at least two consecutive round keys are known. Therefore, we need to inject faults into the third-to-last round to recover $RK^{(39)}$. With $RK^{(40)}$ already known, we can derive both correct and faulty states of penultimate round by decrypting one round. Repeating a filtering process similar to the one used for solving $RK^{(40)}$, we can then determine the candidate set for $RK^{(39)}$. Finally, we can recover the master key and verify the candidate $RK^{(39)}$ and $RK^{(40)}$.

Algorithm 1 Recover the ith-round key with 1-round differentials

```
Input: Normal round state $X^{(i+1)}$, faulty round states $(F^1, F^2, \dots, F^N)$, $S^{(i)}_{\mathsf{erf}}, S^{(i)}_{\mathsf{crf}}$.
Output: The candidate set for $i$th-round key $\mathcal{RK}$.
 1: $\mathcal{RK} = \emptyset$.
 2: for $k$ from 1 to $N$ do
 3:     $FV^k = \{1, \dots, 15\}$.
 4:     for $j$ from 1 to 32 do
 5:         $\Delta X^{(i+1)}_{j,k} = F^k_j - X^{(i+1)}_j$.
 6:         if $j > 1$ then
 7:             $tmpF$ = all possible input differences that lead to $\Delta X^{(i+1)}_{j,k}$ under $S^{(i)}_{\mathsf{erf}}$.
 8:             $FV^k = FV^k \cap tmpF$.
 9:         end if
10:     end for
11: end for
12: $PF = FV^1 \times FV^2 \times \cdots \times FV^k$.
13: for each possible fault $[\Delta X^{(i)}_{1,1}, \dots, \Delta X^{(i)}_{1,N}] \in PF$ do
14:     for $k$ from 1 to $N$ do
15:         $A^{(i)}_{1,k} = A(S^{(i)}_{\mathrm{crf}}, \sum_{j=2}^{32} \Delta X^{(i+1)}_{j,k}, \Delta X^{(i+1)}_{1,k} - \Delta X^{(i)}_{1,k})$.
16:         $\mathcal{K}^{(i)}_{1,k} = \{a - \sum_{j=2}^{32} X^{(i+1)}_{j,k} | a \in A^{(i)}_{1,k}\}$.
17:     end for
18:     $\mathcal{K}^{(i)}_1 = \cap_{k=1}^{N} \mathcal{K}^{(i)}_{1,k}$.
19:     for $RK^{(i)}_1 \in \mathcal{K}^{(i)}_1$ do
20:         $X^{(i)}_1 = X^{(i+1)}_1 - S^{(40)}_{\mathsf{crf}}(\sum_{j=2}^{32} X^{(i+1)}_j + RK^{(40)}_1)$.
21:         for $j$ from 2 to 32 do
22:             for $k$ from 1 to $N$ do
23:                 $A^{(i)}_{j,k} = A(S^{(i)}_{\mathrm{erf}}, \Delta X^{(i)}_{1,k}, \Delta X^{(i+1)}_{j,k})$.
24:                 $\mathcal{K}^{(i)}_{j,k} = \{a - X^{(i)}_1 | a \in A^{(i)}_{j,k}\}$ for $k = 1, \dots, N$.
25:             end for
26:             $\mathcal{K}^{(i)}_j = \cap_{k=1}^{N} \mathcal{K}^{(i)}_{j,k}$
27:         end for
28:         $tmpK = \{RK^{(i)}_1\} \times \mathcal{K}^{(i)}_2 \times \cdots \times \mathcal{K}^{(i)}_{32}$.
29:         $\mathcal{RK} = \mathcal{RK} \cup tmpK$.
30:     end for
31: end for
32: return $\mathcal{RK}$
```

Using only two faults (one for $RK^{(39)}$ and another for $RK^{(40)}$) will lead to an extremely large candidate set for master key, rendering the verification process impractical. To enhance the efficiency and practicality of the attack, we need to inject more faults and intersect the candidate sets derived from different faults. The process for our first DFA on FRAST is shown in Algorithm 2.

Algorithm 2 DFA with fault position assumption

```
Input: Normal keystream $Z$, faulty keystreams derived from penultimate round
    faults $(Z^1, \ldots, Z^N)$, faulty keystreams derived from third-to-last round faults
    $(Z^{N+1}, \ldots, Z^{N+M})$, Public parameters of FRAST.
Output: The master key of FRAST $\boldsymbol{k}$.
 1: $\mathcal{K}^{(40)} =$ Algorithm 1$(Z, (Z^1, \ldots, Z^N), S_{\mathsf{erf}}^{(40)}, S_{\mathsf{crf}}^{(40)})$.
 2: for $RK^{(40)} \in \mathcal{K}^{(40)}$ do
 3:     Perform one round decryption on $Z, Z^{N+1}, \ldots, Z^{N+M}$ with $RK^{(40)}$ and derive
        $Y, Y^1, \ldots, Y^M$.
 4:     $\mathcal{K}^{(39)} =$ Algorithm 1$(Y, (Y^1, \ldots, Y^M), S_{\mathsf{erf}}^{(39)}, S_{\mathsf{crf}}^{(439)})$.
 5:     for $RK^{(39)} \in \mathcal{K}^{(39)}$ do
 6:         $\boldsymbol{k} = M^{-19}(RK^{(39)}||RK^{(40)})$
 7:         if FRAST(ic,$nc||ctr$, $\boldsymbol{k}$) equals $Z$ then
 8:             return $\boldsymbol{k}$
 9:         end if
10:     end for
11: end for
```

From Algorithm 1 and Algorithm 2, one can know that the runtime of DFA is highly related to the size of the intermediate set, especially $|\mathcal{K}^{(40)}|$ and $|\mathcal{K}^{(39)}|$. Furthermore, we implemented the algorithm with Python and simulated the DFA with a different number of injected faults. The specific implementation code is available in Github repository. In our implementation, we set a one-hour run-time limit for the program. Any instance exceeding this time limit (typically corresponding to excessively large search spaces) was discarded during the experiment. For each case, we tested with 1,000 different nonce. Our experimental results are listed in Table 1.

The results demonstrate high efficiency and effectiveness under the assumption that the attacker can control the injected fault locations. Specifically, with 2 faults injected in rounds 39 and 40 (denoted as (2,2) in rounds (39,40)), the attack achieves a success rate of 85.1% with an average time of 0.27 seconds. When increasing the number of faults to 3 each round, the success rate increases to 99.1% at the cost of a slightly higher average time of 3.49 seconds.

Since the injected fault values are random, the probability of two nibble faults being identical is $\frac{1}{15}$, which accounts for the success rate of Algorithm 2. Taking the case of injecting two faults as an example, when the two fault values are identical, we essentially obtain only one effective faulty keystream. As a result, the filtered key set becomes excessively large, making it impractical to determine the correct key within a reasonable time. Therefore, the probability of obtaining two effective faulty keystreams in rounds 39 and 40 is $(1-\frac{1}{15})(1-\frac{1}{15})=\frac{196}{255}\approx 87.1\%$. Similarly, we can analyze the case of injecting three faults. Beyond the scenario where the injected fault values are identical, there are additional factors that can cause a failure. For example, $\sum_{j=2}^{32}\Delta X_j^{(41)}=0$ prevents us from using $\Delta X_1^{(41)}=\Delta X_1^{(40)}+\Delta S_{\mathrm{crf}}^{(40)}(X_2^{(41)}+\cdots+X_{32}^{(41)}+RK_1^{(40)})$ to filter $RK_1^{(40)}$, which could also result in a large candidate key set. Despite that, under the assumption that the attacker can control the fault locations, we can mount the DFA on FRAST with a minimal number of faults and a high success rate.

4 DFA under Practical Random Fault Model

In previous section, we introduced the DFA under the precise fault assumption and achieved promising results on FRAST. However, the assumption may be challenging or require significant costs to realize in practical scenarios. In this section, we relax the strong assumption and present DFAs under a more practical random nibble error model.

4.1 Second DFA: A Naive Extension of First DFA

In Section 3, we established that when faults are injected into the leftmost nibble, we can use 1-round differentials to significantly reduce the size of the candidate round key set. Without the assumption that the attacker can precisely target a specific nibble, a natural approach for DFA is to inject numbers of faults and select the useful ones. In this section, we extend the prior DFA by injecting additional faults to mount a DFA on FRAST under the practical random nibble error model. Under this model, the attacker can inject faults into a specific round state, but the fault will randomly land on one of the nibbles within that round state.

First, we inject faults into the 40th-round state $X^{(40)}$ and collect the corresponding faulty ciphertexts. After computing the output differences between normal ciphertext, we determine the fault locations by analyzing the pattern of these differences. Specifically, if the output difference $\Delta X^{(41)}$ satisfies $\Delta X_1^{(41)}, \Delta X_k^{(41)}\neq 0, \Delta X_j^{(41)}=0, j\in[2,32], j\neq k$, it indicates that the fault occurs at $X_k^{(40)}$;conversely, if the output difference satisfies $\forall j\in[1,32], \Delta X_j^{(41)}\neq 0$, the fault is located at $X_1^{(40)}$. In other words, we can distinguish whether the fault occurs at $X_1^{(40)}$ or other nibbles by observing the number of non-zero terms in $\Delta X^{(41)}$. Once the fault location is identified, we select the faulty ciphertexts where the fault is injected into $X_1^{(40)}$ and recover $RK^{(40)}$ using Algorithm 1.

Next, we inject faults into the 39th-round state $X^{(39)}$ and use $RK^{(40)}$ to decrypt and compute the faulty state and the corresponding difference $\Delta X^{(40)}$. It is important to note that when an incorrect $RK^{(40)}$ is used for decryption, differences $\Delta X^{(40)}$ will appear random, making it impossible to determine the fault location. However, when the correct $RK^{(40)}$ is used, differences $\Delta X^{(40)}$ exhibit non-random characteristics. In particular, differences with less than 2 non-zero terms make up most of these cases (with a probability of $\frac{31}{32}$). With this observation, we can filter candidate $RK^{(40)}$, and then distinguish the fault location based on the number of non-zero terms in $\Delta X^{(40)}$. Someone may note that the 39th-round is a random round, meaning its S-boxes of round function are randomly generated. According to the specification of FRAST, the randomly generated S-boxes might not be permutations. Therefore, it is possible that a non-zero input difference results in a zero output difference, i.e. $\exists a \neq 0, a \rightarrow 0$. Nevertheless, since the round state consist of 32 nibbles, the probability of misclassifying the fault type due to such differential trail is negligible. After determining the fault location, we can use Algorithm 1 to recover $RK^{(39)}$ again.

The procedure of this DFA is similar to Algorithm 2, with the added step of determining fault location during each round. Accordingly, we simulated it with Python and tried varying numbers of injected faults. Again, we tested 1,000 times for each case. The experimental results are summarized in Table 3.

Table 3: DFAs using 1-round differentials under practical random fault model

#fault in Round 39 and 40	Success Rate	Average Success Time
(32,32)	6.3%	20.11 s
(64,64)	32.8%	0.98 s
(96,96)	61.9%	1.44 s
(128,128)	81.9%	1.32 s

From Table 3, it can be observed that as the number of injected faults increases, the success rate of our attack tends to rise. When 128 faults are injected per round, the success rate reaches 81.9%, with an average runtime of just 1.32 seconds. Based on the results in Section 3, it is known that when more than one faults injected into the leftmost nibble per round, the adversary can efficiently recover the master key with a high probability. Furthermore, when n faults are injected into a specific round, the probability that more than one faults occur at the leftmost nibble can be computed as

$$P(n) = 1 - \sum_{i=0}^{1} \binom{32}{i} (\frac{1}{32})^i (1 - \frac{1}{32})^{(n-i)} = 1 - (1 - \frac{1}{32})^n - \frac{n}{32}(1 - \frac{1}{32})^{n-1}.$$

Since the probability of obtaining two effective faulty keystreams when 2 faults are injected into $X_1^{(i)}$ is $Q = \frac{14}{15}$, we can estimate the lower bound of the DFA success rate as $P^2(n)Q^2$, when n faults are injected per round. Using this formula, we calculate the success probabilities for injecting 32, 64, 96 and 128 faults per round as $6.1\%, 31.2\%, 56.5\%, 72.4\%$, respectively. This estimate aligns well with our experimental results in Table 3. When a larger number of faults are injected, the probability of obtaining two effective faulty keystreams Q increases. This explains why the experimental success rate exceeds the estimated lower bound.

4.2 Third DFA: Reducing the Number of Faults

While we have successfully mounted DFA on FRAST under the practical fault model, achieving a desirable success rate requires a large number of faults, which somewhat limits its applications. In this section, we explore how to minimize the number of fault injections while maintaining reasonable attack efficiency.

In previous analysis, we demonstrated how to recover the round key using faults injected into the leftmost nibble $X_1^{(i)}$. To obtain the faulty keystream for $X_1^{(i)}$, we either assume the attacker can control the fault location (as in Section 3.2), or inject a large number of random faults (as in Section 4.1). Given that the round state contains 32 nibbles, statistically, only 1 in 32 faults is expected to affect $X_1^{(i)}$. This explain why our second DFA requires a significant number of faults to achieve a acceptable success rate. However, if we can use faults injected into $X_{\geq 2}^{(i)}$, we can greatly reduce the number of faults needed. From Equation 3, it is known that 1-round differentials for faults in $X_{\geq 2}^{(i)}$ are ineffective to recover the round key. Therefore, we now investigate the 2-round differential propagation for faults in $X_{\geq 2}^{(i)}$.

Suppose that the fault is injected at $X_k^{(i)}, k \in [2, 32]$. The 2-round differential is illustrated in Figure 5. Based on Equation (3), $X_1^{(i+1)}, X_k^{(i+1)}$ will become active after one round propagation. Then, we can compute the difference of $X^{(i+2)}$ as

$$\begin{aligned}
\Delta X_k^{(i+2)} &= \Delta X_k^{(i+1)} + \Delta S_{\text{erf}}^{(i+1)}(X_1^{(i+1)} + RK_k^{(i+1)}),\\
\Delta X_j^{(i+2)} &= \Delta S_{\text{erf}}^{(i+1)}(X_1^{(i+1)} + RK_j^{(i+1)}), j \in [2, 32], j \neq k,\\
\Delta X_1^{(i+2)} &= \Delta X_1^{(i+1)} + \Delta S_{\text{crf}}^{(i+1)}(X_2^{(i+2)} + \cdots + X_{32}^{(i+2)} + RK_1^{(i+1)}).
\end{aligned} \tag{4}$$

With Equation (3) and Equation (4), we can recover the round key $RK^{(i+1)}$ with faults injected at $X_{\geq 2}^{(i)}$. The key recovery path is illustrated as Figure 6. Therefore, we can inject faults into $X^{(39)}$ and recover $RK^{(40)}$ with the faults. The key recovery process is similar to previous attacks, primarily using $\Delta X_1^{(40)}$ to recover the round key $RK^{(40)}$. Before filtering the round key, we must first determine the location and value of fault. Since the fault location cannot be easily inferred from $\Delta X^{(41)}$, we need to iterate over all possible fault locations $k \in [2, 32]$. For each guessed k, we use equations $\Delta X_1^{(41)} = \Delta X_1^{(40)} + \Delta S_{\text{crf}}^{(40)}(X_2^{(41)} + \cdots + X_{32}^{(41)} + RK_1^{(40)})$ and $\Delta X_j^{(41)} = \Delta S_{\text{erf}}^{(40)}(X_1^{(40)} + RK_j^{(40)}), j \in [2, 32], j \neq k$ to determine $\Delta X_1^{(40)}$. Then, the value of $\Delta X_k^{(40)}$ can be deduced by $\Delta X_k^{(41)} = \Delta X_k^{(40)} + \Delta S_{\text{erf}}^{(40)}(X_1^{(40)} + RK_k^{(40)})$ and $\Delta X_1^{(40)} = \Delta S_{\text{crf}}^{(39)}(X_2^{(40)} + \cdots + X_{32}^{(40)} + RK_1^{(40)})$ because $\Delta X_j^{(40)} = 0, j \in [2, 32], j \neq k$. Using the above relationships, we can obtain a guessed set for the injected fault.

For each guessed fault $\Delta X_k^{(39)}$, we have corresponding $\Delta X_1^{(40)}$ and $\Delta X_k^{(40)}$ (as derived from the above recovery path). Given the information of $\Delta X_1^{(40)}$ and $\Delta X_k^{(40)}$, we can use Equation (4) to determine the round key $RK^{(40)}$. This step

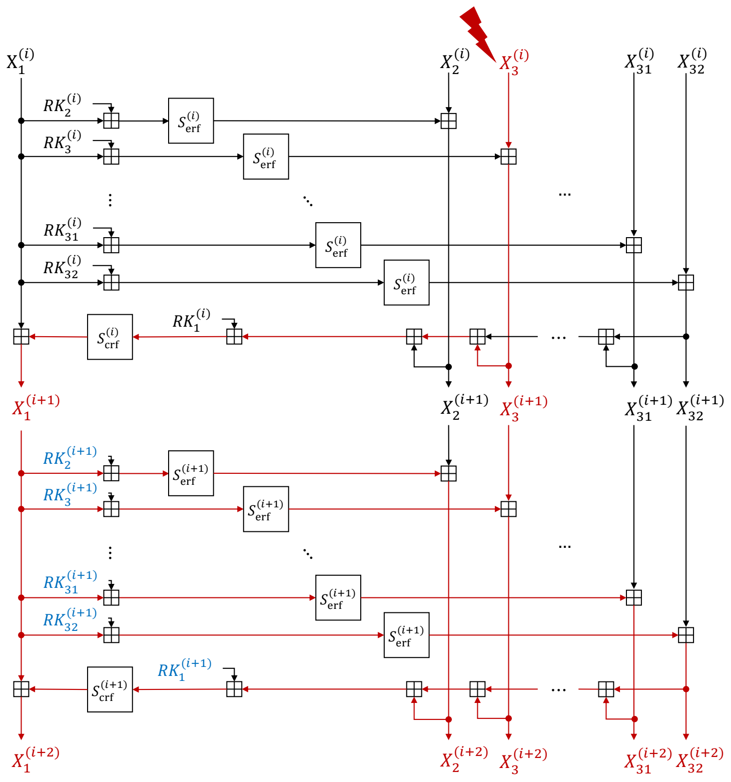


Fig. 5: 2-round differentials when fault is injected at $X_k^{(i)}, k \in [2, 32]$.

$$\begin{tikzcd}
X^{(41)}, \Delta X^{(41)} \arrow[r, "①"] & \Delta X_1^{(40)} \arrow[r, "②"] & \Delta X_k^{(40)} \arrow[r, "③"] & RK_1^{(40)} \arrow[r, "④"] & X_1^{(40)} \arrow[r, "⑤"] & RK_{2-32}^{(40)} \arrow[r, "⑥"] & X_{2-32}^{(40)}
\end{tikzcd}$$

① $\Delta X_j^{(41)} = \Delta S_{\mathrm{erf}}^{(40)}\left(X_1^{(40)} + RK_j^{(40)}\right), j \in [2,32], j \neq k;$ $\Delta X_1^{(41)} = \Delta X_1^{(40)} + \Delta S_{\mathrm{crf}}^{(40)}(X_2^{(41)} + \cdots + X_{32}^{(41)} + RK_1^{(40)})$

② $\Delta X_k^{(41)} = \Delta X_k^{(40)} + \Delta S_{\mathrm{erf}}^{(40)}\left(X_1^{(40)} + RK_k^{(40)}\right);$ $\Delta X_1^{(40)} = \Delta S_{\mathrm{crf}}^{(39)}\left(X_2^{(40)} + \cdots + X_{32}^{(40)} + RK_1^{(39)}\right);$ $\Delta X_j^{(40)} = 0, j \in [2,32], j \neq k$

③ $\Delta X_1^{(41)} = \Delta X_1^{(40)} + \Delta S_{\mathrm{crf}}^{(40)}(X_2^{(41)} + \cdots + X_{32}^{(41)} + RK_1^{(40)})$ ④ $X_1^{(41)} = X_1^{(40)} + S_{\mathrm{crf}}^{(40)}(X_2^{(41)} + \cdots + X_{32}^{(41)} + RK_1^{(40)})$

⑤ $\Delta X_k^{(41)} = \Delta X_k^{(40)} + \Delta S_{\mathrm{erf}}^{(40)}\left(X_1^{(40)} + RK_k^{(40)}\right);$ $\Delta X_j^{(41)} = \Delta S_{\mathrm{erf}}^{(40)}\left(X_1^{(40)} + RK_j^{(40)}\right), j \in [2,32], j \neq k$

⑥ $X_j^{(41)} = X_j^{(40)} + S_{\mathrm{erf}}^{(40)}\left(X_1^{(40)} + RK_j^{(40)}\right), j \in [2,32]$

Fig. 6: The key recovery path for $RK^{(40)}$ using the fault $\Delta X_k^{(39)}, k \in [2, 32]$, with the equations required for each step. Known information is highlighted in red.

is similar to the previous one that uses Equation (2) to recover the round key. The only difference is that, for recovering the nibble $RK_k^{(40)}$, the relationship we use is $\Delta X_k^{(41)} = \Delta X_k^{(40)} + \Delta S_{\text{erf}}^{(40)}(X_1^{(40)} + RK_k^{(40)})$ instead of $\Delta X_k^{(41)} = \Delta S_{\text{erf}}^{(40)}(X_1^{(40)} + RK_k^{(40)})$. It should be noted that the above recovery process only succeeds when the fault is injected at $X_{\geq 2}^{(39)}$. When the fault is injected $X_1^{(39)}$, the process will fail and we cannot filter the round key with the fault. Fortunately, in the case of random fault injection, the probability of a fault being injected at $X_{\geq 2}^{(i)}$ is as high as $\frac{31}{32}$. After determining the round key $RK^{(40)}$, we can recover $RK^{(39)}$ by injecting faults into $X^{(38)}$. Finally, the master key can be derived by inverting the key scheduling algorithm.

We simulated the DFA with a different number of injected faults and tested with 1,000 different nonce. The experimental results are listed in Table 1. Our results demonstrate that our new strategy can effectively reduce the number of faults compared to previous naive extension. The new strategy enables DFA on FRAST with minimal number of faults within a reasonable time, while maintaining a viable success probability. In particular, when 3 faults are injected in rounds 38 and 39, the attack achieves a success rate of 75.2% with an average time of 377.95 seconds. As shown in Table 1, while the success rate of the attacks increases with the number of faults, the time required for the attack also rises rapidly. This is primarily due to the expansion of the guessed fault set that need to be traversed. When the number of faults increases to 4 per round, the runtime and the actual storage capacity of the machine become the main bottlenecks that limit the attack.

5 Discussion about Countermeasures

Although DFAs pose a significant threat to ciphers, designers can employ certain measures to protect them from such attacks. In the past two decades, there have been numerous findings on DFA countermeasures, and interested readers can refer to [3] for further details. In this section, we discuss how to improve the resistance of FRAST against DFA, especially the cipher-level countermeasures.

A direct idea is to use a separate, dedicated device that detects [18] or a shield that blocks any potential source of faults. Such an engineering solution, while applicable to ciphers in general, often incurs high costs in practice and is outside the scope of cryptography design. Protecting specific parts of the circuit in the cryptographic algorithm is a more prudent alternative [4]. In our DFA, faults are often injected into the last two rounds of FRAST. If we can protect the last two rounds from fault injection, the attack can be effectively mitigated. In addition, detection and infection techniques are more sophisticated countermeasures that can help ciphers resist DFAs.

Detection-based countermeasures try to detect data modification caused by fault injection, using principles from coding and information theory, such as error-correcting codes [10,8,40]. Once an anomaly is detected, a predefined procedure is triggered to prevent the attacker from obtaining any information related to the faulty output. Infection techniques are proposed as an extension to detection. They introduce a diffusion effect caused by the faulty intermediate value, which propagates throughout the entire cipher state [6]. This approach avoids the need for explicit detection while making it significantly harder for the attacker to exploit the fault. Through our earlier analysis of FRAST, we know that the DFA can be effectively mitigated by checking whether $X_1^{(i)}$ has been faulted. Therefore, detection and infection techniques can provide low-cost protection for FRAST by focusing on $X_1^{(i)}$. However, it is still possible to break such defenses by injecting a larger number of faults.

Baksi et al. [5] presented the first concept of cipher-level DFA resistance that does not rely on device/protocol assumption or duplication at ASIACRYPT 2021, and gave a full-fledged block cipher DEFAULT. The new design uses special S-boxes with linear structures, and these linear structures imply that certain groups of inputs are differentially equivalent. The definition of linear structure over $\mathbb{F}_2^n$ is given as follows:

Definition 1 (Linear Structure over $\mathbb{F}_2^n$). *A function $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is said to have a linear structure, if there exists an element $a \in \mathbb{F}_2^n$, such that, for some constant $b \in \mathbb{F}_2^n$, $F(x \oplus a) \oplus F(x) = b$ holds for all $x \in \mathbb{F}_2^n$. (a, b) is called a linear structure of F.*

It is clear that any function has a zero linear structure $(0, 0)$, which holds no meaningful significance in this context. Instead, we focus on non-zero linear structures where $a, b \neq 0$. When the S-box L has a non-zero linear structure (a, b), for any differential trail $\alpha \to \beta$, if an element $x \in \mathbb{F}_2^n$ satisfies $L(x \oplus \alpha) \oplus L(x) = \beta$, then $x \oplus a$ will also satisfy $L(x \oplus a \oplus \alpha) \oplus L(x \oplus a) = \beta$. In other words, we cannot uniquely determine the correct input value of the S-box using any number of differential trails. More generally, when the S-box has n non-zero linear structures, the size of the input set filtered with differential trails will be at least $n + 1$. Since there are multiple parallel S-boxes in a round, the introduction of non-zero linear structures causes the size of the filtered key set to grow exponentially, leading to a sharp increase in attack complexity.

Although DEFAULT was later compromised by information-combining DFAs under the precise bit-fault model [30,22], the idea of using structures to resist DFAs remains worthwhile and continues to exhibit resistance under the random fault model. Since FRAST is defined over the group $\mathbb{Z}_{16}$ and our DFAs use differentials over group, we now consider linear structure over group $\mathbb{Z}_N$ and provide the definition by analogy to Definition 1.

Definition 2 (Linear Structure over $\mathbb{Z}_N$). *A function $F : \mathbb{Z}_N \to \mathbb{Z}_N$ is said to have a linear structure, if there exists an element $a \in \mathbb{Z}_N$, such that, for some constant $b \in \mathbb{Z}_N$, $F(x + a) - F(x) = b$ holds for all $x \in \mathbb{Z}_N$. (a, b) is called a linear structure of F.*

Similar to the case on $\mathbb{F}_2^n$, when the S-box S has a non-zero linear structure (a, b), for any differential trail (α, β), if an element $x \in A(S, \alpha, \beta)$, then we can compute $S(x+a+\alpha) - S(x+a)$ as

$$\underbrace{S(x+a+\alpha) - S(x+\alpha)}_{0} + \underbrace{S(x+\alpha) - S(x)}_{\beta} + \underbrace{S(x) - S(x+a)}_{0} = \beta,$$

i.e. $x + a \in A(S, \alpha, \beta)$. This implies that when we replace the S-boxes in a round of FRAST with those containing a non-zero linear structure, using DFA can only reduce the size of the round key space to 2^{32}. From the key schedule, we know that two rounds of round keys are required to recover and verify the master key. Therefore, if we can replace the S-boxes in last two rounds of FRAST with ones with one non-zero linear structure, then the size of candidate key set will be at least 2^{64}, and the practical DFA can be avoided effectively.

It is important to note that the penultimate round of FRAST is a random round. According to the cipher description, the S-boxes in this round should be negacyclic, i.e., $\forall x \in \mathbb{Z}_N, S(x) + S\left(x + \frac{N}{2}\right) = 0$. Simultaneously satisfying the properties of being negacyclic and having a linear structure may severely degrade the S-box, rendering it unsuitable as a secure component. In PBS operation, the group size is required to be a power of two, so we focus on the case where $N = 2^k$. Next, we will prove that when a negacyclic S-box possesses a non-zero linear structure, it will degenerate into a binary function or a constant function. Before proving that, we present a useful lemma along with its proof.

Lemma 1. *Let $N = 2^k, k \geq 2$, for any $a \in \mathbb{Z}_N \setminus \left\{0, \frac{N}{2}\right\}$, there exists a positive integer m satisfying $2a \cdot m \equiv \frac{N}{2} \pmod{N}$.*

Proof. Suppose $a \in \mathbb{Z}_N \setminus \left\{0, \frac{N}{2}\right\}$. By the prime factorization theorem, we can express a as

$$a = 2^s \cdot t \quad (0 \leq s \leq k-2,\ t \text{ is odd}).$$

The original congruence equation $2a \cdot m \equiv \frac{N}{2} \pmod{N}$ becomes:

$$2^{s+1} \cdot t \cdot m \equiv 2^{k-1} \pmod{2^k}.$$

Dividing both sides by 2^{s+1}, we reduce the congruence to:

$$t \cdot m \equiv 2^{k-s-2} \pmod{2^{k-s-1}}.$$

Because t is odd, we have $\gcd(t, 2^{k-s-1}) = 1$. By Bézout's identity, the modular inverse $t^{-1} \pmod{2^{k-s-1}}$ exists. Thus the general solution is

$$m \equiv 2^{k-s-2} \cdot t^{-1} \pmod{2^{k-s-1}},$$

which clearly satisfies $m \in \mathbb{Z}^+$. This completes the proof.

Lemma 1 demonstrates that if we start from any $x \in \mathbb{Z}_{2^k}$ and increment by $2a$ ($a \neq 0$ or 2^{k-1}), the value will always land on $x + 2^{k-1}$. With this lemma, we can show that it is detrimental for a negacyclic S-box to have a linear structure.

Proposition 1. *Let $N = 2^k, k \geq 2$, and let $S : \mathbb{Z}_N \to \mathbb{Z}_N$ be a negacyclic look-up table, if S admits a linear structure $(a, b), a, b \neq 0$, then $b = 2^{k-1}$, and the image of $S(x)$ contains exactly two distinct values.*

Proof. The negacyclic property implies

$$\forall x \in \mathbb{Z}_N, S(x) + S\left(x + \frac{N}{2}\right) = 0 \pmod{N}.$$

Given the linear structure $(a, b), a, b \neq 0$, we have

$$S(x + a) - S(x) \equiv b \pmod{N} \quad \text{for all } x \in \mathbb{Z}_N,$$

And $S(x)$ will never be a constant function.

Let $x' = x + \frac{N}{2}$. Applying the linear structure again:

$$S(x' + a) - S(x') \equiv b \pmod{N}.$$

By the negacyclic property $S(x') \equiv -S(x) \pmod{N}$ and $S(x' + a) \equiv -S(x + a) \pmod{N}$, substitution yields:

$$-S(x + a) + S(x) \equiv b \pmod{N}.$$

Adding the equation to the original linear structure equation gives $2b = 0 \pmod{N}$. Since $b \neq 0$ and $N = 2^k$, we conclude $b = \frac{N}{2} = 2^{k-1}$.

We now analyze the image of S through two cases:

Case 1 ($a = \frac{N}{2}$). The linear structure equation becomes

$$\forall x \in \mathbb{Z}_N, S\left(x + \frac{N}{2}\right) - S(x) = \frac{N}{2} \pmod{N}.$$

Combined with the negacyclic property, substitution gives

$$2S\left(x + \frac{N}{2}\right) = \frac{N}{2} \pmod{N}, 2S(x) = -\frac{N}{2} = \frac{N}{2} \pmod{N}.$$

Solutions satisfy $S(x) \equiv \frac{N}{4} \pmod{\frac{N}{2}}$. Thus the image of S is $\left\{\frac{N}{4}, \frac{3N}{4}\right\}$.

Case 2 ($a \neq \frac{N}{2}$). The linear structure implies periodicity:

$$S(x+2a)-S(x) \equiv (S(x+2a)-S(x+a))+(S(x+a)-S(x)) \equiv 2b \equiv 0 \pmod{N}.$$

By Lemma 1, there exists $m \in \mathbb{Z}^+$ such that $2a \cdot m \equiv \frac{N}{2} \pmod{N}$. Repeatedly applying the linear structure gives:

$$S\left(x + \frac{N}{2}\right) - S(x) \equiv m \cdot b \equiv m \cdot \frac{N}{2} \equiv 0 \pmod{N}.$$

Combining with the negacyclic property $S\left(x + \frac{N}{2}\right) \equiv -S(x) \pmod{N}$, we derive $2S(x) \equiv 0 \pmod{N}$. Thus, the image of S is confined to $\left\{0, \frac{N}{2}\right\}$. This completes the proof.

According to Proposition 1, when a negacyclic S-box has a linear structure, its output will become very simple, failing to provide sufficient confusion. In other words, while introducing a linear structure into a negacyclic S-box can help resist DFA, it introduces other security vulnerabilities, which will render the cipher insecure. Therefore, we recommend removing the negacyclic restriction from the penultimate round of FRAST and incorporating a non-zero linear structure over $\mathbb{Z}_{16}$ into the S-boxes of the last two rounds, thereby mitigate the impact of DFAs.

Some may argue that introducing linear structures over group could make FRAST more vulnerable to linear attacks. However, since we are only modifying the S-boxes in the last two rounds, FRAST still retains sufficient security margin to resist linear attacks. Additionally, the linear structures over group do not compromise FRAST's ability to withstand algebraic attacks. First, due to the presence of the standard fixed layers, the modified FRAST still cannot be expressed as a polynomial over $\mathbb{Z}_{16}^{32}$. Furthermore, the linear properties over $\mathbb{Z}_{16}$ are likely to be non-linear over $\mathbb{F}_2^4$, meaning that the linear structures over group do not weaken the algebraic security of the XOR variant of FRAST. That said, whether there exist new effective attack vectors remains an open question worthy of further in-depth investigation

6 Conclusion

In our study, we introduce DFA against the recent TFHE-friendly cipher, FRAST. Although the unbalanced Feistel structure grants FRAST excellent TFHE application performance, it also introduces security vulnerabilities: modifications to the leftmost nibble directly affect all output nibbles, while changes to the other nibbles only impact the output at their own position and the leftmost nibble. Assuming that all faults could be precisely injected into the leftmost nibble, we give an effective DFA based on 1-round differentials. After discarding the precise fault injection assumption, we employ 2-round differentials to raise the proportion of effective faults and develop a DFA without increasing the number of faults. Our results demonstrate that the secret key of FRAST can be efficiently recovered within a practical time by introducing a few random nibble-based faults into the internal round state. In our discussion on improving FRAST against DFA, we demonstrate that incorporating non-zero linear structures into negacyclic S-boxes is not a viable strategy. Therefore, it is advisable to remove the negacyclic restriction in the final rounds of FRAST and instead employ S-boxes with non-zero linear structures to enhance DFA resistance. Our comprehensive analysis reveals the vulnerabilities of TFHE-friendly ciphers to DFA, highlighting the need for closer scrutiny in the design of ciphers in this category.

References

1. Aikata, A., Dabholkar, A., Saha, D., Roy, S.S.: SASTA: Ambushing hybrid homomorphic encryption schemes with a single fault. Cryptology ePrint Archive, Report 2024/041 (2024), `https://eprint.iacr.org/2024/041`

2. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 430–454. Springer, Berlin, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_17
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.: Protecting block ciphers against differential fault attacks without re-keying. In: 2018 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2018, Washington, DC, USA, April 30 - May 4, 2018. pp. 191–194. IEEE Computer Society (2018). https://doi.org/10.1109/HST.2018.8383913, https://doi.org/10.1109/HST.2018.8383913
5. 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.) ASIACRYPT 2021, Part II. LNCS, vol. 13091, pp. 124–156. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92075-3_5
6. Baksi, A., Saha, D., Sarkar, S.: To infect or not to infect: A critical analysis of infective countermeasures in fault attacks. Cryptology ePrint Archive, Report 2019/355 (2019), https://eprint.iacr.org/2019/355
7. Baudrin, J., Belaïd, S., Bon, N., Boura, C., Canteaut, A., Leurent, G., Paillier, P., Perrin, L., Rivain, M., Rotella, Y., Tap, S.: Transistor: a TFHE-friendly stream cipher. Cryptology ePrint Archive, Paper 2025/282 (2025), https://eprint.iacr.org/2025/282
8. Beierle, C., Leander, G., Moradi, A., Rasoolzadeh, S.: CRAFT: Lightweight tweakable block cipher with efficient protection against DFA attacks. IACR Trans. Symm. Cryptol. **2019**(1), 5–45 (2019). https://doi.org/10.13154/tosc.v2019.i1.5-45
9. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults (extended abstract). In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 37–51. Springer, Berlin, Heidelberg (May 1997). https://doi.org/10.1007/3-540-69053-0_4
10. Breier, J., Hou, X., Liu, Y.: On evaluating fault resilient encoding schemes in software. IEEE Trans. Dependable Secur. Comput. **18**(3), 1065–1079 (2021). https://doi.org/10.1109/TDSC.2019.2897663, https://doi.org/10.1109/TDSC.2019.2897663
11. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. J. Cryptol. **31**(3), 885–916 (2018). https://doi.org/10.1007/S00145-017-9273-9, https://doi.org/10.1007/s00145-017-9273-9
12. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus. J. Cryptol. **33**(1), 34–91 (2020). https://doi.org/10.1007/S00145-019-09319-X, https://doi.org/10.1007/s00145-019-09319-x
13. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: Dolev, S., Margalit, O., Pinkas, B., Schwarzmann, A.A. (eds.) Cyber Security Cryptography and Machine Learning - 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8-9, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12716, pp. 1–19. Springer (2021). https://doi.org/10.1007/978-3-030-78086-9_1, https://doi.org/10.1007/978-3-030-78086-9_1

14. Chiu, T., Xiong, W.: Sok: Fault injection attacks on cryptosystems. In: Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy, HASP 2023, Toronto, Canada, 29 October 2023. pp. 64–72. ACM (2023). https://doi.org/10.1145/3623652.3623671, https://doi.org/10.1145/3623652.3623671
15. Cho, M., Chung, W., Ha, J., Lee, J., Oh, E., Son, M.: FRAST: tfhe-friendly cipher based on random s-boxes. IACR Trans. Symmetric Cryptol. **2024**(3), 1–43 (2024). https://doi.org/10.46586/TOSC.V2024.I3.1-43, https://doi.org/10.46586/tosc.v2024.i3.1-43
16. Cosseron, O., Hoffmann, C., Méaux, P., Standaert, F.X.: Towards case-optimized hybrid homomorphic encryption - featuring the elisabeth stream cipher. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part III. LNCS, vol. 13793, pp. 32–67. Springer, Cham (Dec 2022). https://doi.org/10.1007/978-3-031-22969-5_2
17. Dobraunig, C., Eichlseder, M., Grassi, L., Lallemand, V., Leander, G., List, E., Mendel, F., Rechberger, C.: Rasta: A cipher with low ANDdepth and few ANDs per bit. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 662–692. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1_22
18. He, W., Breier, J., Bhasin, S., Miura, N., Nagata, M.: Ring oscillator under laser: Potential of pll-based countermeasure against laser fault injection. In: 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016. pp. 102–113. IEEE Computer Society (2016). https://doi.org/10.1109/FDTC.2016.13, https://doi.org/10.1109/FDTC.2016.13
19. Hoch, J.J., Shamir, A.: Fault analysis of stream ciphers. In: Joye, M., Quisquater, J.J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 240–253. Springer, Berlin, Heidelberg (Aug 2004). https://doi.org/10.1007/978-3-540-28632-5_18
20. Hoffmann, C., Méaux, P., Standaert, F.X.: The patching landscape of elisabeth-4 and the mixed filter permutator paradigm. In: Chattopadhyay, A., Bhasin, S., Picek, S., Rebeiro, C. (eds.) INDOCRYPT 2023, Part I. LNCS, vol. 14459, pp. 134–156. Springer, Cham (Dec 2023). https://doi.org/10.1007/978-3-031-56232-7_7
21. Hojsík, M., Rudolf, B.: Differential fault analysis of Trivium. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 158–172. Springer, Berlin, Heidelberg (Feb 2008). https://doi.org/10.1007/978-3-540-71039-4_10
22. Jana, A., Kundu, A.K., Paul, G.: More vulnerabilities of linear structure sbox-based ciphers reveal their inability to resist DFA. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part VIII. LNCS, vol. 15491, pp. 168–203. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0944-4_6
23. Jiao, L., Li, Y., Hao, Y., Gong, X.: Differential fault attacks on privacy protocols friendly symmetric-key primitives: Rain and hera. IET Information Security **2024**(1), 7457517 (2024). https://doi.org/https://doi.org/10.1049/2024/7457517
24. Kim, C.H.: Improved differential fault analysis on AES key schedule. IEEE Trans. Inf. Forensics Secur. **7**(1), 41–50 (2012). https://doi.org/10.1109/TIFS.2011.2161289, https://doi.org/10.1109/TIFS.2011.2161289
25. Li, W., Liu, C., Gu, D., Gao, J., Sun, W.: Statistical differential fault analysis of the saturnin lightweight cryptosystem in the mobile wireless sensor networks. IEEE Trans. Inf. Forensics Secur. **18**, 1487–1496 (2023). https://doi.org/10.1109/TIFS.2023.3244083, https://doi.org/10.1109/TIFS.2023.3244083

26. Luo, P., Athanasiou, K., Fei, Y., Wahl, T.: Algebraic fault analysis of SHA-3 under relaxed fault models. IEEE Trans. Inf. Forensics Secur. **13**(7), 1752–1761 (2018). `https://doi.org/10.1109/TIFS.2018.2790938`, `https://doi.org/10.1109/TIFS.2018.2790938`
27. Maitra, S., Siddhanti, A., Sarkar, S.: A differential fault attack on plantlet. IEEE Trans. Computers **66**(10), 1804–1808 (2017). `https://doi.org/10.1109/TC.2017.2700469`, `https://doi.org/10.1109/TC.2017.2700469`
28. Méaux, P., Journault, A., Standaert, F.X., Carlet, C.: Towards stream ciphers for efficient FHE with low-noise ciphertexts. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 311–343. Springer, Berlin, Heidelberg (May 2016). `https://doi.org/10.1007/978-3-662-49890-3_13`
29. Méaux, P., Roy, D.: Theoretical differential fault attacks on FLIP and filip. Cryptogr. Commun. **16**(4), 721–744 (2024). `https://doi.org/10.1007/S12095-024-00698-Y`, `https://doi.org/10.1007/s12095-024-00698-y`
30. Nageler, M., Dobraunig, C., Eichlseder, M.: Information-combining differential fault attacks on DEFAULT. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 168–191. Springer, Cham (May / Jun 2022). `https://doi.org/10.1007/978-3-031-07082-2_7`
31. Piret, G., Quisquater, J.J.: A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In: Walter, C.D., Koç, Çetin Kaya., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 77–88. Springer, Berlin, Heidelberg (Sep 2003). `https://doi.org/10.1007/978-3-540-45238-6_7`
32. Radheshwar, R., Kansal, M., Méaux, P., Roy, D.: Differential fault attack on rasta and $\text{FiLIP}_{\text{DSM}}$. IEEE Trans. Computers **72**(8), 2418–2425 (2023). `https://doi.org/10.1109/TC.2023.3244629`, `https://doi.org/10.1109/TC.2023.3244629`
33. Rivest, R.L., Adleman, L., Dertouzos, M.L., et al.: On data banks and privacy homomorphisms. Foundations of secure computation **4**(11), 169–180 (1978)
34. Rogaway, P.: Nonce-based symmetric encryption. In: Roy, B.K., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 348–359. Springer, Berlin, Heidelberg (Feb 2004). `https://doi.org/10.1007/978-3-540-25937-4_22`
35. Roy, D., Bathe, B.N., Maitra, S.: Differential fault attack on kreyvium & FLIP. IEEE Trans. Computers **70**(12), 2161–2167 (2021). `https://doi.org/10.1109/TC.2020.3038236`, `https://doi.org/10.1109/TC.2020.3038236`
36. 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.) CHES 2016. LNCS, vol. 9813, pp. 581–601. Springer, Berlin, Heidelberg (Aug 2016). `https://doi.org/10.1007/978-3-662-53140-2_28`
37. Saha, D., Chowdhury, D.R.: Scope: On the side channel vulnerability of releasing unverified plaintexts. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. LNCS, vol. 9566, pp. 417–438. Springer, Cham (Aug 2016). `https://doi.org/10.1007/978-3-319-31301-6_24`
38. Saha, D., Kuila, S., Chowdhury, D.R.: EscApe: Diagonal fault analysis of APE. In: Meier, W., Mukhopadhyay, D. (eds.) INDOCRYPT 2014. LNCS, vol. 8885, pp. 197–216. Springer, Cham (Dec 2014). `https://doi.org/10.1007/978-3-319-13039-2_12`
39. Sakiyama, K., Li, Y., Iwamoto, M., Ohta, K.: Information-theoretic approach to optimal differential fault analysis. IEEE Trans. Inf. Forensics Secur. **7**(1), 109–120 (2012). `https://doi.org/10.1109/TIFS.2011.2174984`, `https://doi.org/10.1109/TIFS.2011.2174984`

40. 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.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 581–611. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45721-1_21
41. Wang, W., Méaux, P., Tang, D.: Shortcut2secrets: A table-based differential fault attack framework. IACR Transactions on Cryptographic Hardware and Embedded Systems **2025**(2), 385–419 (2025)
42. Wang, W., Tang, D.: Differential fault attack on he-friendly stream ciphers: Masta, pasta and elisabeth. IEEE Transactions on Computers pp. 1–11 (2025). https://doi.org/10.1109/TC.2025.3558036
43. Zhang, F., Guo, S., Zhao, X., Wang, T., Yang, J., Standaert, F., Gu, D.: A framework for the analysis and evaluation of algebraic fault attacks on lightweight block ciphers. IEEE Trans. Inf. Forensics Secur. **11**(5), 1039–1054 (2016). https://doi.org/10.1109/TIFS.2016.2516905, https://doi.org/10.1109/TIFS.2016.2516905

A Differential Properties of the Fixed S-box of FRAST

To help readers better understand our attacks, we give the Differential Distribution Table (DDT) over $\mathbb{Z}_{16}$ of the fixed S-box (see as Table 2), and show how to determine the input differences with multiple output differences.

According to Table 2, we can calculate the filtered input set

$$A(S, \alpha, \beta) = \{x \in \mathbb{Z}_q | S(x + \alpha) - S(x) = \beta\},$$

for each pair of input and output differences (α, β). By counting the number of elements in the set, we can summarize the DDT as Table 4.

As shown in the table, each non-zero output difference corresponds to multiple possible input differences (on average, approximately 9). By intersecting several input difference sets, we can quickly determine the correct input difference. In the following, we provide an example.

Suppose we obtain four output differences: 2, 10, 14, and 15. According to Table 4, their corresponding input difference sets are $\{1, 3, 5, 6, 10, 11, 14\}$, $\{1, 2, 4, 5, 7, 8, 9, 11, 12, 14\}$, $\{2, 5, 6, 10, 11, 13, 15\}$ and $\{2, 3, 4, 5, 9, 10, 12, 13, 14\}$, respectively. By intersecting these four sets, we can determine the final input difference to be 5.

Table 4: DDT over $\mathbb{Z}_{16}$ of the fixed S-box. "." indicates that no element satisfies the given differential relationship.

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