Shortcut2Secrets: A Table-based Differential Fault Attack Framework

Weizhe Wang[1], Pierrick Méaux[2] and Deng Tang[1(✉)]

[1] Shanghai Jiao Tong University, Shanghai, China,
{SJTUwwz,dengtang}@sjtu.edu.cn
[2] University of Luxembourg, Esch-sur-Alzette, Luxembourg
pierrick.meaux@uni.lu

Abstract. Recently, Differential Fault Attacks (DFAs) have proven highly effective against stream ciphers designed for Hybrid Homomorphic Encryption (HHE). In this work, we present a table-based DFA framework called the *shortcut attack*, which generalizes the attack proposed by Wang and Tang on the cipher Elisabeth. The framework applies to a broad sub-family of ciphers following the Group Filter Permutator (GFP) paradigm and enhances previous DFAs by improving both the fault identification and path generation steps. Notably, the shortcut attack circumvents the issue of function representation, allowing successful attacks even when the cipher's filter function cannot be represented over the ring it is defined on.

Additionally, we provide complexity estimates for the framework and apply the shortcut attack to Elisabeth-4 and its patches. As a result, we optimize the DFA on Elisabeth-4, requiring fewer keystreams and running faster than previous methods. Specifically, we achieve a DFA that requires only 3000 keystreams, which is one-fifth of the previous best result. We also successfully mount a practical DFA on Gabriel-4 and provide a theoretical DFA for Elisabeth-b4.

For the latest patch, Margrethe-18-4, which follows the more general Mixed Filter Permutator (MFP) paradigm, we present a DFA in a stronger model. To the best of our knowledge, these are the first DFA results on the patches of Elisabeth-4. Finally, we derive security margins to prevent shortcut attacks on a broad sub-family of MFP ciphers, which can serve as parameter recommendations for designers.

Keywords: Differential Fault Attack · Hybrid Homomorphic Encryption · Elisabeth-4 · Elisabeth-b4 · Gabriel-4 · Margrethe-18-4

1 Introduction

Differential Fault Attacks (DFAs) are a powerful class of side-channel attacks in which an attacker intentionally injects faults into a cryptographic process and exploits the resulting discrepancies between the correct and faulty outputs to extract sensitive information, such as secret keys. First introduced by Boneh *et al.* in 1997 [BDL97], DFAs have since been applied to a wide range of cryptographic algorithms, from public-key cryptosystems to symmetric-key ciphers. The efficacy of DFAs lies in their ability to bypass traditional cryptographic assumptions, such as the black-box model, by exploiting vulnerabilities in physical implementations.

Symmetric encryption schemes including block ciphers and stream ciphers, are particularly vulnerable to fault attacks due to their reliance on repeated operations over key-dependent transformations. A single fault in a carefully selected stage of these transformations can drastically reduce the complexity of recovering the secret key. For instance, DFAs have been successfully applied to block ciphers like AES [PQ03], where minor perturbations in the internal state lead to significant leakage of key material. Similarly, stream ciphers, which are widely used in real-time applications due to their low latency and high efficiency, are also susceptible to DFAs. When faults are introduced into their keystream generation process, the differences between the correct and faulty keystreams can reveal critical information about the internal state, thus compromising the cipher's security.

Despite their effectiveness, DFA techniques are often tailored to specific cryptographic primitives, requiring intricate knowledge of their structure and state propagation. Stream ciphers, in particular, have

been relatively less explored in DFA research compared to block ciphers. The DFA on a stream cipher was introduced by Hoch and Shamir in 2004 [HS04]. Since then, several notable DFAs have been developed, such as the attacks on Trivium [HR08] and Plantlet [MSS17], where fault attacks have successfully compromised the internal state of these ciphers. The DFA on Plantlet in particular highlighted the vulnerability of modern stream ciphers to fault injections that allow attackers to recover the key with minimal computational effort.

Recently, the landscape of DFA research has expanded to include attacks on symmetric ciphers specifically designed for Hybrid Homomorphic Encryption (HHE). This type of privacy-preserving protocol which merges the efficiency of symmetric encryption with the privacy-preserving capabilities of Fully Homomorphic Encryption (FHE), has become increasingly relevant in the last years. Roy *et al.* [RBM21] presented the first DFA on Kreyvium and FLIP, followed by Radheshwar *et al.* [RKMR23] on RASTA and FiLIP-DSM. Subsequent work has included attacks on RAIN and HERA [JLHG24], FLIP and FiLIP [MR24], and most recently on Masta, Pasta, and Elisabeth [WT24].

HHE, originally introduced by Naehrig *et al.* [NLV11], combines a symmetric encryption scheme with an FHE scheme, allowing privacy-preserving computations on encrypted data without expanding ciphertext size on the client side. Since 2015, several symmetric ciphers have been explicitly designed to support efficient homomorphic evaluation, as this compatibility significantly enhances the overall performance of HHE systems. These HHE-friendly ciphers are engineered to ensure that their structures are compatible with homomorphic evaluation, a key factor in maintaining the efficiency of HHE protocols. However, the aggressive design choices made to optimize these ciphers for homomorphic evaluation have introduced security trade-offs. Specifically, many of these ciphers exhibit a reduced security margin when subjected to algebraic attacks in the traditional black-box model. This vulnerability extends even further when considering DFAs. Given that these ciphers are relatively recent constructions, the full extent of their vulnerability to DFAs is still being explored, but initial research indicates a clear need for heightened protection against such attacks. In this context, the recent work of Aikata *et al.* [ADSR24] illustrates a scenario where fault resistance in HHE protocols could be of critical importance. They highlight that mounting a fault attack on HHE-friendly ciphers may be easier compared to traditional stream cipher applications, due to the independent evaluation of the symmetric scheme on the client and server sides. This underscores the urgency of investigating DFA resistance in the design of future HHE-compatible symmetric ciphers, as fault attacks may pose a greater threat than initially anticipated.

1.1 Contributions and Organization

In this work, we focus on stream ciphers that follow the Group Filter Permutator (GFP) paradigm, such as Elisabeth [CHMS22] and its patches [HMS23] [1], and explore the technique of table-based DFA recently introduced in [WT24]. Unlike previous approaches, the table-based DFA does not depend on the algebraic properties of the system of equations derived from the keystream. This method bypasses the issue of function representation discussed in [CHMS22, HMS23, CCH+24] and challenges the security assumption based on the absence of such representations, as suggested in [GAH+23]. Our main contributions are summarized as follows:

- First, we generalize the attack presented in [WT24] and introduce a table-based DFA framework, referred to as the shortcut attack. The shortcut attack targets a sub-family of ciphers following the GFP paradigm, which we term DS-GFP. This category includes ciphers where the filter function is the direct sum of several smaller functions, each acting on a relatively low number of variables, as is the case with Elisabeth-4, Elisabeth-b4 and Gabriel-4.

  The shortcut attack assumes that one of the N key elements has been randomly faulted, and the adversary uses the normal and faulted keystreams to mount the attack. The primary components of the attack include identifying the fault position and value, generating an optimal path to select the keystream elements in an order that allows the candidate key to be reconstructed element by element, and finally, performing key recovery by combining information from the generated path with the input-output table of a derivative of the filter function.

[1] At ASIACRYPT 2023, a vulnerability in the Elisabeth cipher was found [GBJR23]. The "patch" aims to mitigate this weakness and enhance the cipher's security.

The shortcut attack enhances the DFA described in [WT24] for Elisabeth-4 by improving the fault identification and path generation steps. Additionally, we provide complexity estimates for the shortcut attack, relating them to the parameters of any DS-GFP cipher.

- Second, we apply the shortcut attack to Elisabeth-4 and its patches, Elisabeth-b4 and Gabriel-4. The results are summarized in Table 1, and the corresponding code is available at https://github.com/SJTUwwz/DFA_Elisabeth_family. For Elisabeth-4, our DFAs achieves better performance, requiring fewer keystreams and running faster than the attack in [WT24]. Specifically, we perform a DFA that requires only 3000 keystreams, which is one-fifth of the best result presented in [WT24]. The DFAs on Gabriel-4 remain practical, completing in under one day. Additionally, we provide theoretical estimations for the DFA on Elisabeth-b4. To the best of our knowledge, these are the first DFA results on the patches of Elisabeth-4.

- Then, for the last patch presented in [HMS23], Margrethe-18-4, which follows the more general Mixed Filter Permutator (MFP) paradigm using different groups, we were unable to directly apply the shortcut DFA. Instead, we consider a DFA in a stronger model, where the adversary can inject random faults and re-initiate the attack multiple times. In this model, we propose two attacks: one that recovers the key after injecting 2048 faults (equal to the size of the key), and an improved version that injects only 699 faults and then recovers the remaining key using the shortcut attack.

- Finally, motivated by the practical limitations of the shortcut attack on Elisabeth-b4 and the theoretical challenges on Margrethe-18-4, we explore the feasibility limits of the attack on DS-MFP ciphers. We derive security margins to prevent shortcut attacks based on the parameters of these ciphers and provide the evaluation results for Elisabeth-4 and its patches in Table 2.

Table 1: Our shortcut attacks on Elisabeth and its patches

Cipher	paradigm	#fault	#keystream	Time [a]	Reference
Elisabeth-4	GFP	1	15000	150s	[WT24]
			3000	17215s	Section 4.1
			10000	38s	
Gabriel-4			40000	35757s	Section 4.2
			100000	202s	
Elisabeth-b4			500000	$2^{99.58}$	Section 4.3
			2000000	$2^{75.58}$	
Margrethe-18-4 [b]	MFP	700	230000	29437s	Section 5
		2048	99865	<1s	

[a] We only record the time for filtering the candidate key set, because it is the dominant factor of our DFAs

[b] The DFA on Margrethe-18-4 need a strong attacker with the ability to avoid injecting fault at the same position.

The structure of the article is as follows. In Section 2, we introduce the notations and provide extended background information on Elisabeth and its patches, DFA techniques, methods for state recovery, and the DFA approach from [WT24]. In Section 3, we present the shortcut attack framework, starting with an explanation of the sub-family of GFP ciphers it targets, followed by a detailed description of the various components of the attack and the corresponding time complexity estimates. Under this framework, we successfully implement single-fault DFAs on Elisabeth-4 and its patches in Section 4. The DFA with multiple faults on Margrethe is discussed in Section 5. In Section 6, we investigate the security margins of DS-MFP ciphers against shortcut attacks. Finally, we conclude the paper in Section 7.

Table 2: Secure limits of the number of keystreams for different ciphers

Cipher	(N, t, m)	$\log_{\lvert\mathbb{G}_1\rvert} \lvert\mathbb{G}_2\rvert$	SL	1-fault DFA
Elisabeth-4	$(256, 12, 4)$	1	$2^{8.32}$	✓
Gabriel-4	$(512, 8, 4)$		$2^{10.91}$	✓
Elisabeth-b4	$(512, 14, 6)$		$2^{15.98}$	✓
Margrethe-18-4	$(2048, 14, 18)$	4	$2^{47.01}$	✗

2 Preliminaries

2.1 Notations

We denote a group as $\mathbb{G}$, an integer ring with q elements as $\mathbb{Z}_q$, and a finite field with q elements $\mathbb{F}_q$. For a set X, we use $|X|$ to represent the number of elements or, in other words, the cardinal of X. We use bold italic letters to represent vectors, e.g. $\boldsymbol{a} \in \mathbb{G}^t$ denotes the vector $\boldsymbol{a} = (a_1, \ldots, a_t)$, where $a_i \in \mathbb{G}$. The Hamming weight of a vector is $\mathsf{w_H}(\boldsymbol{a}) = |\{i : a_i \neq 0\}|$. Given an integer k, we write $[k]$ to denote the set $\{1, \ldots, k\}$.

2.2 Elisabeth and Its Patching Landscape

Elisabeth [CHMS22] is an HHE-friendly stream cipher proposed at ASIACRYPT 2022. The design of Elisabeth extends FLIP and FiLIP families of stream ciphers [MJSC16, MCJS19] and follows the Group Filter Permutator (GFP) paradigm. The GFP is illustrated in Figure 1 and defined by a group $\mathbb{G}$ with operation noted $+$, a forward secure Pseudo Random Number Generator (PRNG), a key size N, a subset size n, and a filter function f from $\mathbb{G}^n$ to $\mathbb{G}$. To encrypt m elements of $\mathbb{G}$ under a secret key $\boldsymbol{k} \in \mathbb{G}^N$, the public parameters of the PRNG are chosen and then the following process is executed for each keystream $s^{(i)}$ (for $i \in [m]$):

- The PRNG is updated, its output determines pseudorandom elements: a subset, a permutation, and a length-n whitening vector of $\mathbb{G}$.
- $S^{(i)}$ is the n-out-of-N subset from the N key elements,
- $P^{(i)}$ is the n-to-n (wire-cross) permutation,
- $\boldsymbol{w}^{(i)}$ is the whitening vector, belonging to $\mathbb{G}^n$,
- the keystream element $s^{(i)}$ is computed as $s^{(i)} = f(P^{(i)}(S^{(i)}(\boldsymbol{k})) + \boldsymbol{w}^{(i)})$, where $+$ denotes the element-wise addition of $\mathbb{G}$.

Elisabeth-4 is the GFP paradigm instantiated with

- $\mathbb{G} = \mathbb{Z}_{16}, N = 256, n = 60$,
- the filter function $f(x_1, \ldots, x_{60})$ is the direct sum of 12 times the 5-to-1 function g, which can be expressed as:

$$f(x_1, \ldots, x_{60}) = \sum_{i=0}^{11} g(x_{5i+1}, x_{5i+2}, \ldots, x_{5i+5}),$$

- the 5-to-1 function g is the sum of a nonlinear 4-to-1 function h and the remaining variable, *i.e.*

$$g(x_1, x_2, x_3, x_4, x_5) = h(x_1, x_2, x_3, x_4) + x_5.$$

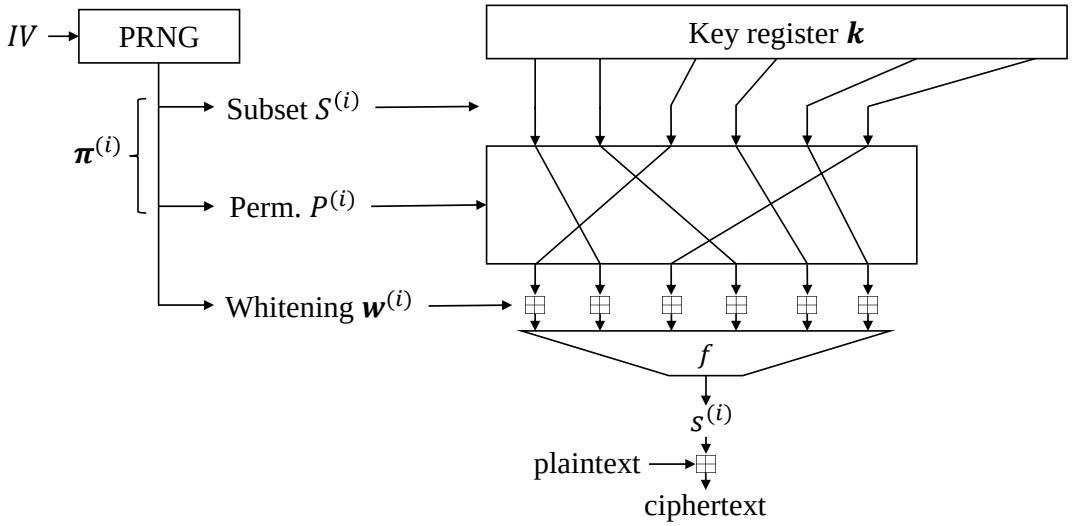

Figure 1: The Group Filter Permutator design.

The construction of function h is given in Appendix B.1. In the GFP paradigm, the filter function is typically a direct sum of several identical inner functions g. We use t to denote the number of g and r to represent the number of input variables for the function g. For example, $t = 12$ and $r = 5$ in Elisabeth-4. Additionally, the inner function g is generally obtained by summing a smaller nonlinear function h and the remaining variables.

In [GBJR23], Gilbert *et al.* proposed an algebraic attack on the full Elisabeth-4 based on the lower than expected number of monomials in the $\mathbb{F}_2$-representation, and broke the 128-bit security claim. As a remedial measure, Hoffmann *et al.* [HMS23] proposed three patched ciphers: Elisabeth-b, Gabriel, and Margrethe, that restore the security of such designs while maintaining their good properties for HHE. They first updated the design by mixing more chunks in order to prevent a too-low number of monomials in one of the Boolean functions obtained when the keystream is written as equations over $\mathbb{F}_2$. As a result, Elisabeth-b was introduced and the specification is given in Appendix A.

Although the 7-to-1 nonlinear function enhances Elisabeth security, it also reduces the cipher's computational efficiency in HHE. By making a trade-off between security and efficiency, the designers introduced the Gabriel cipher. The design approach for Gabriel consists of having two different branches for the filter function, which are called as the left and right parts. Gabriel-4 is the GFP paradigm instantiated with

- $\mathbb{G} = \mathbb{Z}_{16}, N = 512, n = 110$,
- the filter function $f(x_1, \ldots, x_{110})$ is the direct sum of 8 times the 5-to-1 function g_L and 10 times the 7-to-1 function g_R, where g_L is the same as the inner function of Elisabeth-4 and g_R is the same as the inner function of Elisabeth-b4.

The Gabriel-4 is able to tackle both the attacks studied in [CHMS22] and [GBJR23] and has better performances than Elisabeth-b4.

Regarding the weakness presented in [GBJR23], the designers also considered a different setting with two changes, the look-up tables do not need to be negacyclic and the function can operate at the bit level. Specifically, they proposed the idea of MFP, which is the generalization of GFP. The MFP is defined by two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ and its encryption process is similar to that of GFP. The only difference is that the stream ciphers following the MFP paradigm will take a secret key over $\mathbb{G}_1$ and output the keystreams over $\mathbb{G}_2$. Margrethe-18-4 is the MFP paradigm instantiated with:

- $\mathbb{G}_1 = \mathbb{F}_2, \mathbb{G}_2 = \mathbb{Z}_{16}, N = 2048, n = 308$,
- the filter function is the direct sum of 14 times the inner function g, which can be expressed as:

$$f(x_1, \ldots, x_{308}) = \sum_{i=0}^{13} g(x_{22i+1}, \ldots, x_{22i+22}),$$

- the inner function $g : \mathbb{F}_2^{22} \mapsto \mathbb{Z}_{16}$ is defined as:

$$g(x_1, \ldots, x_{22}) = h(x_1, \ldots, x_{18}) + \mathbb{Z}_{16}(\sum_{k=0}^{3} 2^k x_{19+k}),$$

  where the symbols $\sum$ and $+$ denote the addition modulo 16 and $\mathbb{Z}_{16}(x_1, x_2, x_3, x_4)$ denotes the element of $\mathbb{Z}_{16}$ with binary representation (x_1, x_2, x_3, x_4).

The function $h : \mathbb{F}_2^{18} \mapsto \mathbb{Z}_{16}$ is given by a look-up table and the generation of the look-up table is given in [HMS23].

2.3 Differential Fault Attacks

Real-world cryptosystems, implemented in software or hardware, are vulnerable to attacks targeting their implementation rather than their theoretical specifications. In extreme cases, adversaries might directly extract secret keys from the execution environment. While careful key concealment mitigates this, a more prevalent threat is the fault injection attack. This involves introducing faults into the device and analyzing the resulting behavior to recover keys. This technique has proven effective against smart cards, commercial security processors [AK96], cryptographic LSI [FT09] and even, recently, neural networks [BHJ+18]. Among fault injection attacks, DFA is the most effective and common one. The DFA was first proposed by Boneh *et al.* [BDL97] in 1997. The first DFA on stream ciphers was introduced by Hoch and Shamir [HS04] at CHES 2004. Recently, DFAs have been applied to HHE-friendly stream ciphers [RBM21,RKMR23,JLHG24,MR24,WT24] and all achieved good results. The attacker of DFA is more powerful than that of classical attacks. In DFA, the attacker can inject faults into the state of ciphers and collect the normal and faulty outputs to recover the secret key. The faults can be injected by some specific tools, such as laser shots, electromagnetic waves, unsupported voltage, etc. Formally, the underlying assumptions of the DFA model are outlined as follows:

1) The attacker can repeatedly restart the cipher using the same key and other public parameters (e.g., nonce and IV).

2) The attacker can inject faults at specific timings during the keystream generation phase and monitor both the normal and faulty keystreams.

3) The attacker has the required tools (such as laser shots, electromagnetic waves, etc.) for injecting faults.

4) The number of injected faults must be kept minimal to prevent potential damage to the device.

The difficulty of an attack method is highly related to the type of faults injected and the number of fault-injected ciphertexts required. The commonly used fault injection models include bit flip model and random word error model. In the case of bit flip model, a bit-based fault is injected and the value of faulty state bit will flip. For the random word error model, a word-based fault is injected and the value of faulty state word will turn into an unknown random value. After the fault injection, the attacker proceeds with the following steps to recover the secret key:

1) Identify the location and value of the injected fault. If identifying the location or value of the fault is infeasible, then make a guess.

2) Recover the state using information from both the normal and faulty keystreams. This process often involves constructing and solving equations.

3) Derive the secret key from the obtained state.

The nonce-based encryption was formalized by Rogaway [Rog04] and it was shown to provide better resistance against DFA. There are several methods to mount DFA targeting nonce-based encryption, e.g. nonce-misuse [SKC14], nonce-bypass [SC16b], internal DFA [SC16a]. The stream ciphers used in HHE

are also nonce-based and previous DFAs [RBM21, RKMR23, JLHG24, MR24, WT24] on these ciphers all used nonce repetition. Recently, in [ADSR24], Aikata *et al.* proposed a novel framework for DFA called SASTA dedicated to the HHE use-case. In this framework, the attacker obtains the faulted keystream from the client side and the non-faulted one by the homomorphic evaluation of the (symmetric encryption's) decryption from the server side. While this framework has some drawbacks, such as relying on a strong attacker on the client side (involving fault injection, access to the faulted keystream, and the result of homomorphic decryption), and requiring homomorphic encryption of data already owned by the client, it offers the significant advantage of not needing nonce repetition, unlike other attacks. The procedure of generating differential in the SASTA framework is shown in Figure 2.

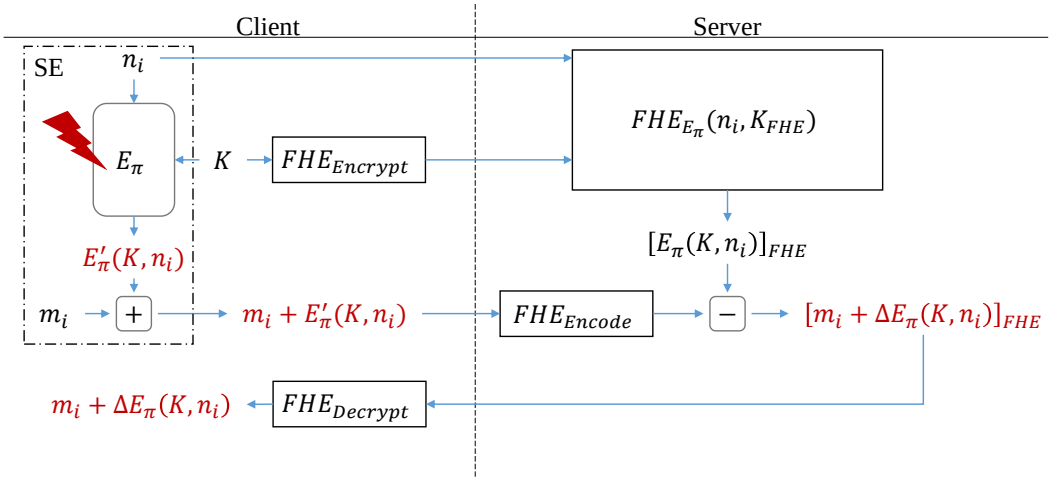

Figure 2: Utilizing SASTA for HHE.

2.3.1 Threat Model

We consider a typical DFA setup where an adversary can inject a fault into the client's device, resulting in a modification of the key. The attack is then mounted solely using two keystreams: one obtained with the original key and the other with the faulted key. Inducing faults on the key is considered more practical than gaining full access or control of the client's key. Since the theoretical basis of the attack relies only on the difference between normal and faulty keystreams, we specify two contexts that enable the adversary to acquire these related keystreams:

- **Using Nonce Repetition**: As in previous attacks in this domain (see [RBM21, RKMR23, MR24, WT24]), we assume that the client first uses the symmetric encryption scheme with a nonce and the correct key. A fault is then injected into the client's device, and the client is coerced into re-encrypting with the same nonce and the faulted key. This coercion can be achieved, for example, by faking non-reception of the encrypted data. The nonce-repetition attack targets only the symmetric encryption scheme and is not specific to HHE.

- **Within the SASTA Framework**: This DFA framework, introduced in [ADSR24], is specific to HHE. Here, the adversary injects a fault into the key of the symmetric encryption scheme on the client's device and gains access to the results of homomorphic decryption, which also occur on the client side. This allows the adversary to obtain equations that are functions of the plaintext plus the difference of the keystreams. The functions derived are those evaluated homomorphically by the server. For instance, if the evaluated function is the identity (as in data storage scenarios), and the plaintext is known, the adversary can directly compute the difference between the keystreams. In this context, nonce repetition is not required.

2.4 Methods for Recovering the State

In DFA, the most important step is to recover the state using information from both the normal and faulty keystreams. This step usually involves formulating multivariate polynomial equations that connect the secrets of a cryptographic primitive (such as its key) with the information available to the attacker (including IV, keystreams, plaintext, ciphertext, etc.). The goal is to solve this system of algebraic equations to recover the secret key. In principle, this can always be accomplished by expressing the relationships derived from the specifications of the cryptographic primitive in equation form. However, in practice, the resulting equations can be challenging to write and store, and may require the introduction of intermediate variables. Even when it is feasible to create these equations, solving a large nonlinear multivariate system with many variables is often impractical due to its complexity.

Although it is hard to solve a system of polynomial equations in general, there still exist some methods. Linearization is an elementary method that considers every monomial appearing in the system as an independent variable and solves the resulting linear system of equations via Gaussian elimination. The time complexity and memory complexity of linearization can be computed as $O(\binom{n+d}{d}^w)$ and $O(\binom{n+d}{d}^2)$, where n is the number of variables, d is the upper bound of the equation degree, and $2 \leq w \leq 3$ is the constant of linear algebra. It is a commonly used technique in algebraic attacks e.g. [CM03, Cou03, GBJR23, LKSM24]. However, the drawback of linearization is that it will introduce a large amount of variables, which necessitates a substantial number of equations to solve, especially when the system of equations is of high degree and has a dense number of terms.

Gröbner basis is another common method for solving multivariate nonlinear equations. In a Gröbner basis attack, there are three main steps: First, compute a Gröbner basis using Buchberger's algorithm [Buc65], F4 [Fau99], or F5 [Fau02]. Next, perform a change of term ordering for the computed Gröbner basis using FGLM [FGLM93]. Finally, solve the univariate equation for the last variable using a polynomial factoring algorithm and substitute it to obtain the complete solution. The Gröbner basis attack is a powerful tool for Arithmetization-Oriented (AO) ciphers [ACG+19, BBLP22, BBL+24]. Although, the Gröbner basis attack has a wide range of applicability, it also has drawbacks such as high memory consumption.

In addition to the two common solving methods of linearization and Gröbner basis, there are also some specialized solving techniques. For example, it is a good choice to transform Boolean equation system into Boolean satisfiability problems (SAT) and the using solvers like CryptoMiniSAT [SNC09] to find solutions [MBB11, RBM21, RKMR23]. Resultant is also a useful structure for solving multivariate equations and has been used to evaluate the security of some AO ciphers accurately [YZY+24].

In certain cases, the multivariate polynomial equations involving the key cannot be formulated easily, as the function may not have a polynomial representation over the input domain, such as in Elisabeth-4. One approach to address this is to bypass the step of constructing polynomial equations and instead store the mapping relationships in a table. The attacker can then iteratively filter the candidate key set using this table. We refer to this approach as the table-based filtering method, which is essentially equivalent to equation solving but does not rely on expressing the mapping relationships through polynomials. This method proves to be more efficient than equation solving when the polynomial expression is complex, but the input space is small. As a result, it has several applications in cryptography. For example, in DFA on block ciphers [PQ03, NDE22, JP22], attackers can filter and determine the value of the state using the Differential Distribution Table (DDT) of the S-box. The table-based filtering method is straightforward and easy to implement, requiring only repeated set intersection operations for filtering. We note that this method circumvents the issue of function representation discussed in [CHMS22, HMS23, CCH+24] and challenges the assumption of security based on the absence of such representation, as advocated in [GAH+23].

2.5 Previous DFAs on Elisabeth-4

In [WT24], Wang and Tang proposed DFAs on three HHE-friendly stream ciphers: Masta, Pasta, and Elisabeth. For Elisabeth-4, the only instance of the Elisabeth family, they presented two different DFAs with both bit-based faults and word-based faults.

In their DFA on Elisabeth-4, the attacker first determines the position of faulty key word with the following lemma.

Lemma 1. *For the DFA on Elisabeth family ciphers, if the difference between normal and faulty keystreams $\Delta s^{(i)}$ is non-zero, then the faulty keyword must be included in the subset $S^{(i)}$ generated by the PRNG.*

Proof. We prove the contrapositive: if the faulty keyword is not included in the subset $S^{(i)}$ generated by the PRNG, then the difference between normal and faulty keystreams $\Delta s^{(i)}$ is zero.

Suppose the normal state is $(x_1^{(i)}, \ldots, x_{60}^{(i)})$ and the faulty state is $(x_1^{(i)'}, \ldots, x_{60}^{(i)'})$, since the faulty keyword is not included in the subset $S^{(i)}$, we have

$$x_j^{(i)} = x_j^{(i)'}, \forall i \in [60].$$

Therefore, $f(x_1^{(i)}, \ldots, x_{60}^{(i)}) = f(x_1^{(i)'}, \ldots, x_{60}^{(i)'})$, i.e. $\Delta s^{(i)} = 0$. □

By intersecting multiple n-out-of-N subset $S^{(i)}$ with $\Delta s^{(i)} \neq 0$, the faulty key position can be identified using very few equations. For key recovery, the attack described in [WT24] abandons the traditional strategy of equation construction and solving, opting instead for a table-based filtering method combined with a path-based key recovery strategy. By analyzing the difference between the normal and faulty keystreams, an attacker can derive the output difference of the function h for the faulty state block. Given the output difference Δh, the possible values of the faulty state block can be filtered. Thanks to the simple and public linear layer of Elisabeth, the attacker can reverse the linear layer to recover the corresponding key values. By iteratively merging and intersecting the candidate key set using multiple output keystream differences, the correct key can eventually be determined. Additionally, the authors of [WT24] proposed a greedy algorithm to generate the order of filtering, or the merging path, and successfully implemented DFAs on Elisabeth-4 in practice. This also marks the first DFA attack on the Elisabeth family of ciphers. We show in the next sections how the table-based filtering method can be used and improved to perform a DFA on Elisabeth and its patches, and more generally on a subfamily of ciphers following the GFP and MFP paradigms.

3 The Shortcut Attack for Elisabeth and Its Patches

In this section, we generalize the attack from [WT24] and propose a table-based DFA framework, referred to as the *shortcut attack*, as it bypasses the traditional cryptographic criteria typically dependent on the representation of the filter function. Furthermore, we enhance the original attack by improving fault identification and path generation. Using the shortcut attack, we can optimize the DFA on Elisabeth-4 and successfully mount DFAs on its patched versions.

3.1 The DS-GFP Ciphers and Its Properties

In this part we specify a subfamily of ciphers following the GFP paradigm and explain why the shortcut DFA is strong for this subfamily.

Elisabeth and two of its patches: Elisabeth-b and Gabriel, follow the GFP paradigm. Moreover, their filter functions are the direct sum (DS) of functions in a small number of variables. We refer to a subclass of ciphers following the GFP as DS-GFP ciphers, the ciphers following the GFP paradigm as defined in Section 2.2 with the following restrictions on f:

- f is the direct sum (over $\mathbb{G}$) of d distinct functions f_i for $i \in [d]$,
- each f_i is the direct sum of t_i times the function g_i,
- g_i is the direct sum of a nonlinear function h_i and a variable x_{r_i}.

Equivalently:

$$f(x_1, \ldots, x_n) = \sum_{i=0}^{d-1} f_i(x_{\phi_i+1}, \ldots, x_{\phi_{i+1}}) = \sum_{i=0}^{d-1} \sum_{j=0}^{t_i-1} g_i(x_{\phi_i+jr_i+1}, \ldots, x_{\phi_i+(j+1)r_i}),$$

where $\phi_i = \sum_{\ell=0}^{i-1} n_\ell, n_i = r_i t_i, n = \sum_{i=0}^{d-1} n_i$, and

$$g_i(x_1, \ldots, x_{r_i}) = h_i(x_1, \ldots, x_{m_i}) + x_{r_i},$$

where $m_i = r_i - 1$.

Following the vocabulary of [CHMS22, HMS23], the g_i functions are called inner functions, Elisabeth-4 and Elisabeth-b4 are DS-GFP ciphers with one inner function, that is $d = 1$ and $m_1 = 4$ and $m_1 = 6$ respectively. Gabriel-4 is a DS-GFP cipher using two different inner functions, with $m_1 = 4$ and $m_2 = 6$. We also remark that most instances of the GFP such as the considered filters for FLIP [MJSC16] and FiLIP [MCJS19, GGM24] are in the DS-GFP family.

The structure of DS-GFP ciphers differs significantly from that of traditional stream ciphers. While the GFP paradigm's offers superior performance in HHE (e.g. transciphering with latency in milliseconds [CDPP22, MPP24]), it also introduces certain security vulnerabilities, making the ciphers more susceptible to algebraic attacks [DLR16, GBJR23] and DFAs [WT24]. At each iteration of the DS-GFP cipher, the public PRNG generates an ordered arrangement $\boldsymbol{\pi}^{(i)} = (\pi_1^{(i)}, \ldots, \pi_n^{(i)})$ and an n-length whitening vector $\boldsymbol{w}^{(i)} = (w_1^{(i)}, \ldots, w_n^{(i)})$. The state is computed as $(k_{\pi_1^{(i)}} + w_1^{(i)}, \ldots, k_{\pi_n^{(i)}} + w_n^{(i)})$, and the output of this iteration is obtained by applying the filter function to the state. Due to the absence of feedback functions, the key register remains unchanged throughout the entire encryption process. Consequently, the relationship between the output keystreams and the secret key does not increase in complexity with each iteration. This is a key difference between DS-GFP ciphers and feedback-based ciphers. When attacking DS-GFP ciphers, attackers do not need to contend with increasingly complex relationships as more keystreams are collected.

The linear layer (or lack thereof) in the DS-GFP cipher exhibits extremely poor diffusion properties. As mentioned earlier, the linear layer is generated by the PRNG and consists of an ordered arrangement and a whitening vector. The ordered arrangement $\boldsymbol{\pi}^{(i)}$ can be viewed as a combination of selecting an n-subset of $\{1, \ldots, N\}$ and a permutation of its elements. As a result, each state element depends on only one key element, which has two main consequences in the context of DFA we consider. First, an attacker can directly compute the values of key elements based on the corresponding state elements. For instance, given the value of a state element $k_{\pi_1^{(i)}} + w_1^{(i)}$ and the whitening value $w_1^{(i)}$, the value of $k_{\pi_1^{(i)}}$ can be computed directly. Second, a fault injected into the key register will affect at most one state element per iteration. Specifically, when the faulted key position is included in the subset $S^{(i)}$, one state element will be impacted. We refer to the resulting keystream as valid keystream. Conversely, when the faulted key position is not part of the subset $S^{(i)}$, no state element will be affected.

The filter function of the DS-GFP cipher is particularly vulnerable to DFAs. Since the filter function is the direct sum of several small inner functions, when a fault occurs in the state, only one inner function g_i is affected. As a result, by calculating the difference between the normal keystream and the faulty keystream, only the output of one inner function is preserved, while the others are eliminated. This allows us to obtain the output difference of a specific inner function from the difference in the filter function, $\Delta s^{(i)}$. This property weakens the security of the DS-GFP cipher, reducing it from the filter function f to an inner function g_i, or even to a smaller nonlinear function h_i.

Without loss of generality, suppose the attacker inject a $\mathbb{G}$-based fault in the first key word k_1, then we have:

$$k_1' = k_1 + \Delta k, k_i' = k_i, i = 2, \ldots N,$$

where $\boldsymbol{k} = (k_1, \ldots, k_N)$ and $\boldsymbol{k'} = (k_1', \ldots, k_N')$ are the normal and faulty keys. For a valid keystream at iteration J, the faulty key position is included in the ordered arrangement $\boldsymbol{\pi}^{(J)}$ and the difference of the state can be computed as

$$\Delta x_i^{(J)} = \Delta(k_{\pi_i^{(J)}} + w_i^{(J)}) = \begin{cases} \Delta k, & \pi_i^{(J)} = 1 \\ 0, & \text{otherwise} \end{cases}. \tag{1}$$

The difference of output keystream $\Delta s^{(J)}$ can be expressed as:

$$\Delta s^{(J)} = \Delta f(x_1^{(J)}, \ldots, x_n^{(J)}) = \sum_{i=0}^{d-1} \sum_{j=0}^{t_i - 1} \Delta g_i(x_{\phi_i + jr_i + 1}^{(J)}, \ldots, x_{\phi_i + (j+1)r_i}^{(J)}). \tag{2}$$

According to Equation (1), we know that only one element of the state will be affected by the fault. Therefore, we can get the output difference of an inner function g_i with the Equation (2). Because the inner function g_i is also a direct sum, we can further decompose the difference. When the fault lies in the first m_i variables of g_i, we have

$$\Delta s^{(J)} = \Delta h_i(x^{(J)}_{\phi_i+jr_i+1}, \ldots, x^{(J)}_{\phi_i+jr_i+m_i}). \tag{3}$$

When the fault lies at the r_i-th variable of g_i, we have

$$\Delta s^{(J)} = \Delta x^{(J)}_{\phi_i+(j+1)r_i}. \tag{4}$$

With Equation (3) and Equation (4), the attacker is able to mount an efficient DFA on the DS-GFP cipher as it leverages the properties of a function with only m_i variables.

3.2 The Shortcut Attack Framework

By injecting a $\mathbb{G}$-based fault into the key register of the DS-GFP cipher, the attacker can derive Equation (3) and Equation (4) from each valid keystream. Though Equation (4) cannot be used to recover the secret key, we can obtain the value of the fault with it. With Equation (3), the value Δh_i can be obtained from $\Delta s^{(J)}$. It should be noted that, Δh_i only relates to m_i state elements, instead of the N-length state, which can be of great help for key recovery (for example $N = 256$ whereas $m_1 = 4$ for Elisabeth-4). The crucial point is how to recover the secret key efficiently with multiple valid keystreams.

A natural approach is to find the polynomial representation of h_i and form a system of multivariate equations related to the secret key using multiple Δh_i, similar to the method used in [MR24] for FLIP and FiLIP ciphers. However, this method becomes infeasible for DS-GFP ciphers that use groups larger than $\mathbb{F}_2$. In the latest DS-GFP ciphers, the inner functions are specifically designed without polynomial representations over $\mathbb{G}$. For example, it has been proven that the inner functions of both Elisabeth-4 and Elisabeth-b4 are not polyfunctions [CHMS22, HMM+23]. While it is theoretically possible to map the input-output relationships to other algebraic structures and build corresponding equations for solving (for example over $\mathbb{F}_2$ for Elisabeth-4 and its patches), this approach is impractical due to the complexity of solving such equations.

Another approach is to compute and store the DDT of the function h_i as the filter table T_i. It is important to note that there is no need to compute and store the table for all $|\mathbb{G}|^{m_i}$ input differences. Due to Equation (1), there are only $m_i \cdot (|\mathbb{G}| - 1)$ possible input differences in the DFA on DS-GFP ciphers under consideration. As a result, the size of the filter table $|T_i|$ is $m_i \cdot (|\mathbb{G}| - 1) \cdot |\mathbb{G}|^{m_i} \approx m_i|\mathbb{G}|^{m_i+1}$. Using the filter table T_i and Δh_i, we can derive a solution set for the input state. Then, by inverting the linear layer of the DS-GFP cipher, we obtain a candidate key set. Multiple candidate key sets can be derived from multiple valid keystreams, and the correct secret key can be retrieved by intersecting all the candidate key sets. This approach is efficient for DS-GFP ciphers, and we formalize the above DFA method as the shortcut attack. The framework for the shortcut attack is presented in Algorithm 1.

Algorithm 1 The shortcut attack framework

1: Compute and store the filter table $T = \{(x_1, \ldots, x_m, \delta, j, \Delta h)\}$ for all $(x_1, \ldots, x_m) \in \mathbb{G}_1^m, \delta \in \mathbb{G}_1^*$ and $j \in \{1, \ldots, m\}$, where δ is the value of input difference and j is the position of the faulty word.
2: Inject a $\mathbb{G}$-based fault into the key and collect a number of normal and faulty keystreams.
3: Identify the position of the fault and determine the value of the fault.
4: Generate the merging path.
5: Recover the secret key by filtering the candidate key set according to the merging path as illustrated in Figure 3.

For the DS-GFP cipher with multiple nonlinear function h_i, we only utilize its Achilles' heel, the one with fewest input variables. In the following sections, for the sake of clarity, we denote the nonlinear function h_i with fewest input variables as h, its filter table as T, its number of input variable as m and the number of times it appears in the filter function f as t, unless otherwise stated. Our shortcut attack generalizes the one on Elisabeth-4 presented in [WT24]. Besides, we present improvements in both fault identification and

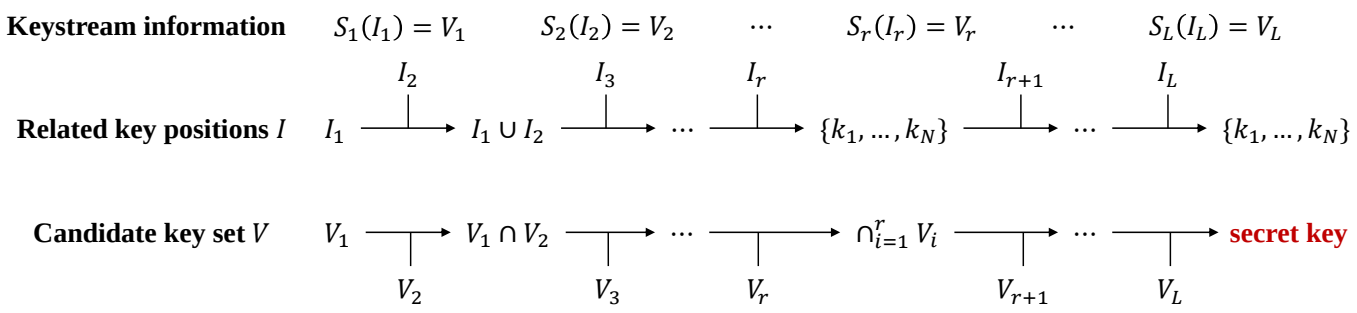

Figure 3: filtering the candidate key set with the merging path $P = S_1 \to S_2 \to \cdots \to S_L$ until the right key is found.

path generation, which greatly enhance the DFA. The description and complexity estimation for each step of Algorithm 1 is given in the following subsections.

3.3 Identifying and Determining the Fault

In DFA, the attacker generally does not know the exact location of the injected fault. In [BMS12], Banik *et al.* proposed the signature-based fault identification technique, which pinpoint the fault location by statistical testing. In situations where linear layers are pseudorandom, the signature will look random and therefore the statistical method is ineffective. In [MR24], Méaux and Roy introduced an additional approach for identifying the fault location with inference sets and applied it to FLIP and FiLIP. With an efficient fault identification, the attacker can avoid guessing the location and it greatly improves the time complexity of the DFA.

In [WT24], Wang and Tang used multiple subsets $S^{(i)}$ with $\Delta s^{(i)} \neq 0$ and Lemma 1 to quickly identify the fault position in the DFA on Elisabeth-4. Lemma 1 can be applied to other DS-GFP ciphers. Besides Lemma 1, we use an additional lemma to identify the position in our shortcut attack.

Lemma 2. *For the DFA on DS-GFP ciphers, if the difference between normal and faulty keystreams $\Delta s^{(i)}$ is zero, then the faulty keyword does not appear at the linear part of the inner function.*

Proof. Similar to the proof of Lemma 1, we prove the contrapositive version of this lemma, i.e. if the faulty keyword appears in the linear part of the inner function, then the difference between normal and faulty keystream $\Delta s^{(i)}$ is non-zero.

The difference of output keystream $\Delta s^{(i)}$ can be expressed as

$$\Delta s^{(i)} = \Delta f(x_1^{(i)}, \ldots, x_n^{(i)}) = \sum_{j=0}^{d-1} \sum_{l=0}^{t_j - 1} \Delta g_j(x^{(i)}_{\phi_j + l r_j + 1}, \ldots, x^{(i)}_{\phi_j + (l+1) r_j}).$$

Since the inner function $g_j(x_1, \ldots, x_{r_j}) = h_j(x_1, \ldots, x_{r_j - 1}) + x_{r_j}$, when the fault lies at the linear part of inner function g_j, i.e. the r_j-th variable of g_j, we have

$$\Delta s^{(i)} = \Delta x^{(i)}_{\phi_j + (l+1) r_j}.$$

Hence, the output difference $s^{(i)}$ equals to the value of the fault, which is non-zero. □

Lemma 1 and Lemma 2 use direct generalizations to $\mathbb{G}$-variable the concept of full and null inference set defined in [MR24] for binary variables. The new identification method is described in Algorithm 2.

With the additional part using Lemma 2, the identification is completed with less keystreams than in [WT24]. For each iteration, we need $O(nN)$ operations to filter out the wrong candidate fault positions. Assuming only one wrong position is removed each time, the time complexity of Algorithm 2 is $O(nN^2)$.

Algorithm 2 Identify the faulty key positions with Lemma 1 and Lemma 2.

Input: The difference of keystreams $\Delta \boldsymbol{s}$, initial vector IV, $PRNG$.
Output: The unique fault position $PosCan$.

$PosCan \leftarrow \{1, \ldots, N\}$
$i = 0$
while $|PosCan| > 1$ **do**
  Generate the ordered arrangement $\boldsymbol{\pi}^{(i)} \leftarrow PRNG(IV, i)$
  if $\Delta s^{(i)} \neq 0$ **then** ▷ Lemma 1 case
    $PosCan \leftarrow PosCan \cap \boldsymbol{\pi}^{(i)}$
  end if
  if $\Delta s^{(i)} == 0$ **then** ▷ Lemma 2 case
    for $j \in \{1, \ldots, n\}$ **do**
      if x_j is the linear part of the inner function and $\pi_j^{(i)} \in PosCan$ **then**
        Remove $\pi_j^{(i)}$ from $PosCan$
      end if
    end for
  end if
  $i \leftarrow i + 1$
end while
return $PosCan$

In the random word error model, the value of the fault is random and unknown to the attacker. Therefore, the attacker must determine the fault value before proceeding with secret key recovery. In [WT24], the approach consisted in testing all possible fault values, which means the key recovery process needed to be repeated an average of 8 times for Elisabeth-4 (since $\mathbb{G} = \mathbb{Z}_{16}$). In the shortcut attack, however, we can directly determine the fault value by analyzing the difference between the normal and faulty keystreams using Equation (4). This improvement significantly reduces the time complexity of the DFA. Once the fault position is known, the fault value can be determined with $O(1)$ operations. Therefore, the time complexity of identifying and determining the fault is $O(nN^2)$.

3.4 Generating the Merging Path

After determining the fault, the attacker needs to recover the secret key with multiple Δh. Given the filter table T and the output difference Δh, a solution set for the input state can be derived. The attacker can then obtain a candidate key set by inverting the linear layer of the DS-GFP cipher. Due to the properties of the function h and the linear layer, each candidate key set derived from one element of the valid keystream only corresponds to m key elements.

To represent the candidate key set, we use an indexed set $S(I) = V$ where I is the related key positions and V is the set of possible values. This representation is considerably more compact compared to directly recording all possible solutions. For example, we represent the set $\{(1,0,0,0),(1,0,0,1),(1,0,1,0),(1,0,1,1)\} \subset \mathbb{F}_2^4$ as $S(\{1,2\}) = \{(1,0)\}$. An indexed set $S(I) = V$ can also be viewed as a constraint, where I is the variables related to the constraint and V is the specific constraint. This allows us to derive multiple candidate sub-key sets from the valid keystreams: $S_1(I_1) = V_1$, $S_2(I_2) = V_2, \ldots, S_L(I_L) = V_L$, $|I_i| = m, 1 \leq i \leq L$. The key recovery process is actually solving a constraint problem with L constraints, and each constraint relates to only m key variables. Given two constraints $S_1(I_1) = V_1, S_2(I_2) = V_2$, we can merge them and obtain a new constraint $S(I) = V$, where $I = I_1 \cup I_2, V = \{w | u_i = v_i, i \in I_1 \cap I_2, \boldsymbol{u} \in V_1, \boldsymbol{v} \in V_2\}$, w is the vector of length $len(I_1) + len(I_2) - |I_1 \cap I_2|$ and made from the elements of $\boldsymbol{u}$ and the ones of $\boldsymbol{v}$ minus the intersection. This process can also be viewed as the intersection of two indexed sets.

It is crucial to determine the optimal order in which to use these keystreams elements to filter the candidate keys. We keep the number of key elements that are not fully determined small during the process, in order to always keep the size of the candidate keys set small. We refer to this sequence as the merging path. While different merging paths may ultimately yield the same filtering result, the number of intersection

operations required—and consequently, the time complexity—can vary significantly. In Appendix C, we provide a small example to illustrate the impact of different merging paths on time complexity.

We denote the maximum size of the candidate set during the filtering process as $|S_{max}|$. To minimize the cost of filtering, $|S_{max}|$ should be kept as small as possible. Therefore, when determining the merging path, $|S_{max}|$ is estimated, and the optimal merging path is the one that minimizes this value. The filtering process involves multiple intersections, and regarding the sizes of the sets after the intersections we give a proposition and estimate $|S_{max}|$ based on an assumption.

Proposition 1. *Let $S_1(I_1) = V_1$, $S_2(I_2) = V_2$ be two indexed sets, $|I_1| = m, |I_1| = n$, The elements of V_1, V_2 are chosen uniformly at random from $\mathbb{G}^m, \mathbb{G}^n$ respectively and independently, and $|V_1| = M, |V_2| = N$. If the number of common indexes $|I_1 \cap I_2|$ is ℓ, then the size of the set after taking the intersection is $\frac{MN}{|\mathbb{G}|^\ell}$.*

Proof. On the ℓ indexes in common $I_1 \cap I_2$ for any $x \in \mathbb{G}^\ell$ an element of V_1 takes the value x with probability $p_{1,x} = \frac{1}{|\mathbb{G}|^\ell}$ since its element are chosen uniformly at random from $\mathbb{G}^m$. Similarly, for any $x \in \mathbb{G}^\ell$ an element of V_2 takes the value x with probability $p_{2,x} = \frac{1}{|\mathbb{G}|^\ell}$. Since the distributions of V_1 and V_2 are independent, the average size after intersection is given by:

$$\begin{aligned}\{\mathbf{u} \cup \mathbf{v} \,|\, u_i = v_i, \forall i \in I_1 \cap I_2, \mathbf{u} \in V_1, \mathbf{v} \in V_2\}| &= \sum_{x \in \mathbb{G}^\ell} p_{1,x}|V_1| \cdot p_{2,x}|V_2| \\ &= |\mathbb{G}|^\ell \frac{M}{|\mathbb{G}|^\ell}\frac{N}{|\mathbb{G}|^\ell} = \frac{MN}{|\mathbb{G}|^\ell}.\end{aligned}$$

□

Definition 1 (Preimage ratio)**.** Let $m \in \mathbb{N}$ and $\mathbb{G}$ be a finite group. Let f be a function from $\mathbb{G}^m$ to $\mathbb{G}$ and $\boldsymbol{a} \in \mathbb{G}^m$, we denote by $M_{f,\boldsymbol{a}} = \max_{u \in \mathbb{G}} |\{\boldsymbol{x} \in \mathbb{G}^m \text{ such that } \Delta_{\boldsymbol{a}}(f) = u\}|$ and by $m_{f,\boldsymbol{a}} = \min_{u \in \mathbb{G}} |\{\boldsymbol{x} \in \mathbb{G}^m \text{ such that } \Delta_{\boldsymbol{a}}(f) = u\}|$.

We denote by $c_{f,\boldsymbol{a}}$ the value $M_{f,\boldsymbol{a}}/m_{f,\boldsymbol{a}}$ and $c_f = \max_{\mathsf{w_H}(\boldsymbol{a})=1} c_{f,\boldsymbol{a}}$, where $\mathsf{w_H}(\boldsymbol{a})$ denotes the Hamming weight of $\boldsymbol{a}$ (*i.e.* the number of non null coefficients of $\boldsymbol{a}$).

We note that if the derivative of f at $\boldsymbol{a}$ is balanced, then $c_{f,\boldsymbol{a}} = 1$. For instances of Elisabeth-4 and its patches, we observe that $c_{f,\boldsymbol{a}} < 2$[2], meaning that no preimage has significantly more occurrences than another. Therefore, for simplicity in the analysis, we assume $c_{f,\boldsymbol{a}} = 1$. When $c_h = 1$, we can assume that the candidate key sets derived from each keystream differential equation are of equal size, and each candidate key has the same probability. This allows us to estimate the sizes of the sets used in the merging path based on Proposition 1. We proceed under the following working assumption:

Assumption 1. *Let h be a function from $\mathbb{G}^m$ to $\mathbb{G}$, if c_h is small, e.g. $c_h < 5$, the result of Proposition 1 can be used to estimate the average size of the candidate key sets.*

Assuming Assumption 1, we can estimate $|S_{max}|$ for a specific merging path. According to Assumption 1, and using Proposition 1 when the size of the input sets remains unchanged, the more common positions two input sets have, the smaller the resulting intersection set is. Therefore, to keep $|S_{max}|$ as small as possible, we need to generate a path such that the related key positions I increase as slow as possible. The most straightforward approach to finding the optimal path is to enumerate all possible paths and select the one with the smallest $|S_{max}|$ as the merging path. However, this approach leads to a factorial growth in time complexity as the number of keystreams increases, making it impractical for an attack.

The Greedy Algorithm (GA) is an alternative method for efficiently generating an appropriate merging path. An intuitive approach to keeping $|S_{max}|$ small is to select, at each step, the keystream that shares the most common key positions with the current candidate key set, which naturally leads to a greedy algorithm. The pseudo-code for the greedy algorithm is provided in Algorithm 3. A straightforward implementation of the greedy algorithm starts by selecting the first valid keystream as the initial point and then iteratively choosing the closest keystream. While the greedy algorithm is both efficient and easy to implement, it is sub-optimal, and the choice of the initial point can significantly impact the quality of the result.

[2]The table for the different functions h can be found in https://github.com/SJTUwwz/DFA_Elisabeth_family, they have the following preimage ratio: $c_h = 1.35$, 1.46 and 1.04 for Elisabeth-4, Elisabeth-4b and Margrethe-18-4, respectively.

Algorithm 3 Generate the merging path by greedy algorithm

Input: The set of m-length key positions of valid keystreams KP, the group size $|\mathbb{G}|$, the number of variables m, the initial pair of indexes (p_1, p_2).

Output: The merging path P, estimated $|S_{max}|$.

$TI \leftarrow \{1, \ldots, len(KP)\}$
$tS \leftarrow KP[p_1] \cup KP[p_2]$ ▷ The initialization phase
$P \leftarrow [p_1, p_2]$
$M \leftarrow$ the number of common key positions between $KP[p_1]$ and $KP[p_2]$.
$|S_{max}| \leftarrow |\mathbb{G}|^{2(m-1)-M}$ ▷ Using Proposition 1
$NewS \leftarrow |S_{max}|$
Remove p_1, p_2 from TI
Remove $KP[p_1], KP[p_2]$ from KP
while $|tS| < N$ and $KP \neq \emptyset$ **do**
    $j \leftarrow 0$
    $com \leftarrow 0$
    for $i \in 1$ to $len(I)$ **do**
        if $|tS \cap KP[i]| \geq com$ **then**
            $j \leftarrow i$
            $com \leftarrow |tS \cap KP[i]|$
        end if
    end for
    $NewS \leftarrow NewS \cdot |\mathbb{G}|^{m-1-com}$ ▷ Using Proposition 1
    if $NewS > |S_{max}|$ **then**
        $|S_{max}| \leftarrow NewS$
    end if
    $tS \leftarrow tS \cup KP[j]$
    $P.add(j)$
    Remove $KP[j]$ from KP
    Remove j from TI
end while
if $I \neq \emptyset$ **then**
    $P \leftarrow P + T$ ▷ $|tS| = N$, and we add the remaining points to the end directly
end if
return P, $|S_{max}|$

Instead of selecting the first valid keystream as the initial point, the attack in [WT24] enumerates all pairs of valid keystreams and selects the pair with the most common key positions as the initial point. Using this improved starting point, they generate a more effective merging path and successfully mount a DFA on Elisabeth-4 with 15000 keystreams in 150 seconds. The paper also demonstrates the theoretical benefits of using a better initial point.

While searching for a better initial point can lead to a more optimal merging path, it also increases the cost of generating the path. However, since the main overhead of the table-based DFA lies in the filtering process, spending more time to generate a better merging path that reduces the time spent filtering the candidate key set is worthwhile. This approach can significantly lower the overall time complexity of the attack. Building on this idea, we further improve the greedy algorithm used in table-based DFA. In the improved version, we introduce an initial point pool that stores all keystream pairs with the maximum number of common positions. For each initial point in the pool, we generate a merging path using the greedy algorithm and then select the best one as the final result, rather than generating the path based on only the first initial point. We refer to this enhanced method as the Multiple Initial Points Greedy Algorithm (MIP-GA), and we refer to the algorithm in [WT24] as original GA. The pseudo-code for MIP-GA is provided in Algorithm 4. Figure 4 illustrates the differences between MIP-GA and original GA. Specifically, Algorithm 3 considers only one path, while Algorithm 4 considers multiple paths (including the path

considered by Algorithm 3) and compares them before selecting the best one. This indicates that the merging path found by Algorithm 4 is at least as good as, if not better than, that of Algorithm 3. The results in Table 3 also demonstrate that Algorithm 4 indeed performs better.

Algorithm 4 Generate the merging path by MIP-GA

Input: The set of m-length key positions of valid keystreams KP, the group size $|\mathbb{G}|$, the number of variables m.

Output: The optimal merging path.

$ip \leftarrow []$
$\ell \leftarrow len(KP)$
$num \leftarrow 0$
for $i = 0, \ldots, \ell - 1$ **do**
  for $j = i + 1, \ldots \ell - 1$ **do**
    if $|KP[i] \cap KP[j]| > num$ **then**
      $ip \leftarrow [(i, j)]$
      $num = |KP[i] \cap KP[j]|$
    end if
    if $|KP[i] \cap KP[j]| = num$ **then**
      $ip.add((i, j))$
    end if
  end for
end for
$RP \leftarrow []$
$|S_{max}| \leftarrow +\infty$
for $pair \in ip$ **do**
  $P, |S| \leftarrow$ Algorithm 3$(KP, |\mathbb{G}|, m, pair)$
  if $|S| < |S_{max}|$ **then**
    $RP \leftarrow P$
    $|S_{max}| \leftarrow |S|$
  end if
end for
return RP

Compared to the original GA, MIP-GA calls the greedy algorithm multiple times to generate paths for all initial points. When the number of keystreams is large, resulting in many initial points with the same number of common positions, we can impose a limit on the number of initial points in the pool to balance efficiency and performance. Although MIP-GA takes more time than the original GA to generate and select merging paths, it consistently finds better paths with the same set of keystreams. By using MIP-GA, we can achieve DFAs with fewer keystreams and reduced running time.

In Algorithm 3, the number of iterations in the while loop is $O(\frac{mtL}{N})$, where L is the number of keystreams and $\frac{mtL}{N}$ represents the average number of valid keystreams (since in each iteration, one element is removed from KP, and the loop stops when KP is empty). In each iteration of the while loop, the for loop iterates a maximum of L times, which we bound again by $\frac{mtL}{N}$. In each iteration of the for loop, computing the number of common key positions between the current index and the candidate m-length index set requires $O(mN)$ operations. Thus, the time complexity of a single greedy algorithm is $O((\frac{mtL}{N})^2 \cdot mN)$, or $O(\frac{m^3t^2L^2}{N})$.

Algorithm 4 involves selecting the initial points, applying the greedy algorithm to each initial point, and then choosing the best path. The time complexity of MIP-GA is therefore $O((\frac{mtL}{N})^2 \cdot m^2 + |ip|\frac{m^3t^2L^2}{N})$, where $|ip|$ denotes the size of the initial point pool. The attacker can set an proper upper bound M for $|ip|$ to balance the quality of the paths with the runtime. Since $m < N$ and M is a constant decided by the attacker, the asymptotic complexity of MIP-GA is $O(\frac{Mm^3t^2L^2}{N})$.

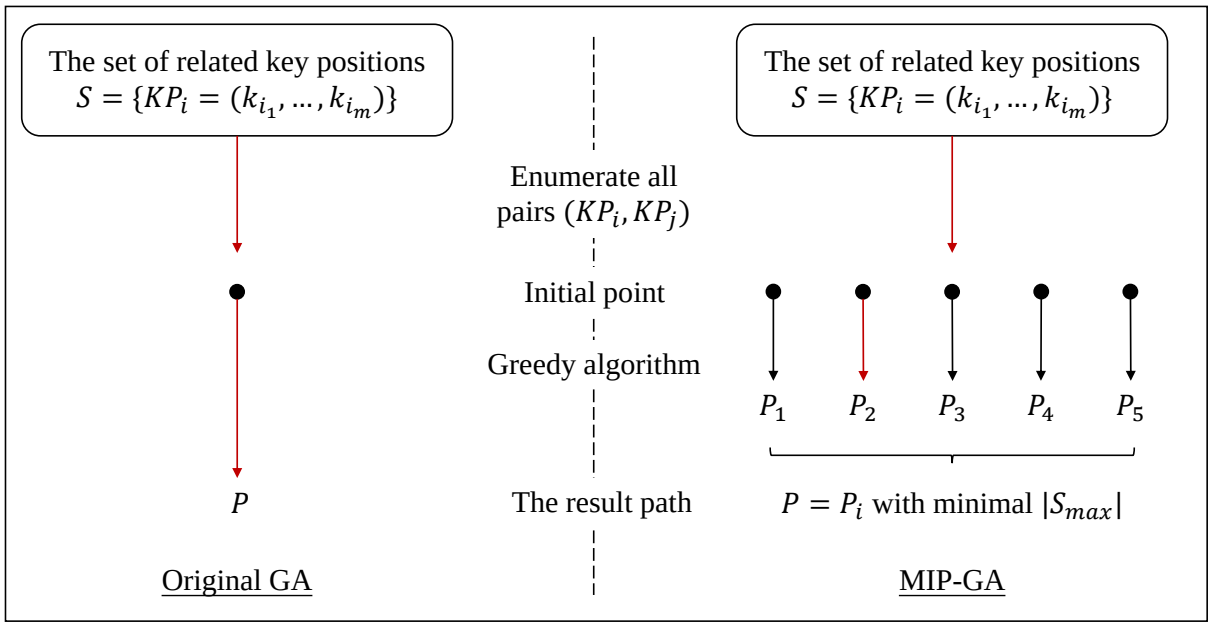

Figure 4: Illustration of MIP-GA and original GA.

3.5 Recovering the Secret Key

During the key recovery step, the correct key must be included in all candidate key sets derived from valid keystreams. Therefore, the secret key can be determined through a step-by-step intersection process. Given a merging path, the attacker merges and intersects the candidate key sets, as shown in Figure 3. The pseudo-code for this process is provided in Algorithm 5. Throughout the process, the number of totally determined key elements gradually increases to N, and merging continues until the uniquely correct key is found, *i.e.* , when $|I| = N$ and $|V| = 1$. If $|V| = 0$, it indicates that the assumed information is incorrect.

Let S denote the candidate set during the filtering process. For each intersection, it requires $|S| \cdot |S_i|$ comparisons, with each comparison taking $O(mN)$ operations, where S_i is the candidate key set derived from $\Delta s^{(i)}$. This process is repeated multiple times until the correct secret key is found. The size of S is upper bounded by $|S_{max}|$, which we estimate using Proposition 1. According to Assumption 1, the average size of S_i is $|\bar{S}| = \frac{|\mathbb{G}|^m}{|\mathbb{G}|} = |\mathbb{G}|^{m-1}$. Therefore, the time complexity for the key recovery step is $O(mN \cdot |S_{max}| \cdot |\mathbb{G}|^{m-1})$.

The total time complexity of the shortcut attack can be expressed as:

$$\text{Time complexity} = O(m|\mathbb{G}|^{m+1} + nN^2 + \frac{Mm^3t^2L^2}{N} + mN|S_{max}||\mathbb{G}|^{m-1}), \tag{5}$$

which corresponds to Steps 1, 3, 4, and 5 in Algorithm 1. In most cases, the time complexity of the key recovery step dominates the attack. This part can be reduced by finding a better merging path with more keystreams. However, as the number of keystreams L increases, the overhead of generating paths may surpass that of key recovery. To perform an efficient shortcut attack on DS-GFP ciphers, the attacker must balance the costs of Step 4 and Step 5, aiming to minimize the overall time complexity as expressed in Equation (5).

Remark 1. In this work, we primarily consider single-fault scenarios. If an attacker is able to inject multiple faults, the complexity of DFA may be further reduced. For example, when an attacker has the capability to inject multiple different faults at the same position (this requires injecting numerous faults randomly or stronger assumptions on the injection model), key recovery can be completed more rapidly. Injecting multiple faults at the same position reduces the size of the candidate set. For different faults, the linear layer remains unchanged, which means that the m-out-of-N subsets generated by PRNG are the same. This allows attackers to refine their search by intersecting the candidate key sets derived from the filter table T, resulting in a smaller $|\bar{S}|$ and, consequently, a smaller candidate set during the merging phase.

Algorithm 5 Key recovery: merge and intersect the candidate key sets

Input: The set of m-length key positions of valid keystreams KP, The difference of valid keystreams DS, the value of fault δ, the filter table T, the merging path P, PRNG, IV

Output: The correct secret key

Calculate the exact fault positions in input of function h IND and the value of the corresponding whitening value W, from IV and PRNG.
$t \leftarrow P[0]$
$S \leftarrow \{x - W[t] \,|\, (x, \delta, IND[t], DS[t]) \in T\}$ ▷ S is the candidate key set derived from T with the fault (δ, $IND[t]$) and the output difference $DS[t]$.
$tS \leftarrow KP[t]$
for $i = 1, \ldots, len(P) - 1$ **do**
    $tmpS \leftarrow \emptyset$
    $t \leftarrow P[i]$
    $H \leftarrow \{x - W[t] \,|\, (x, \delta, IND[t], DS[t]) \in T\}$
    for $u \in S$ **do** ▷ Merge and intersect two sets
        for $v \in H$ **do**
            if u, v have the same value on $tS \cap KP[t]$ **then**
                $w \leftarrow u||v$ ▷ $u||v$ denotes the vector of length $len(u) + len(v) - |tS \cap KP[t]|$ made from u and from v minus the intersection.
                $tmpS.add(w)$
            end if
        end for
    end for
    $tS \leftarrow tS \cup KP[t]$
    $S \leftarrow tmpS$
end for
for $sol \in S$ **do** ▷ Generally, $|S| \leq 2$
    Use sol as the key and simulate the cipher to generate keystreams with IV
    if The generated keystreams equals the normal keystreams **then**
        return sol
    end if
end for

4 DFAs on Elisabeth-4, Gabriel-4, and Elisabeth-b4 with a Single Fault

To validate the effectiveness of the shortcut DFA, we apply it to several versions and patches of the Elisabeth cipher family, including Elisabeth-4, Gabriel-4, and Elisabeth-b4. As a result, we successfully improve the DFA on Elisabeth-4, perform practical DFAs on Gabriel-4, and provide theoretical DFAs for Elisabeth-b4. In the following parts, we introduce the specific attacks in detail, and give the practical results obtained from the attacks described in Section 3. We simulate the DFA using Python 3.9, and all experiments are conducted on our workstation (2× Intel(R) Xeon(R) 5220R CPUs with 24 cores, running Ubuntu 20.04).

4.1 Application to Elisabeth-4

Elisabeth-4 cipher is the first instance of Elisabeth family of ciphers, introduced in [CHMS22], we refer to Section 2.2 for its full description. For the shortcut attack, we consider Elisabeth-4 as any DS-GFP cipher and focus on its nonlinear function h defined from $\mathbb{G}^4$ to $\mathbb{G}$ where $\mathbb{G} = \mathbb{Z}_{16}$.

For the shortcut attack, the first step is to compute the filter table T for the nonlinear function $h(x_1, \ldots, x_4)$. The whole filter table T has $2^{16} \cdot 15 \cdot 4 \approx 2^{22}$ rows. In the implementation, we decompose the whole large table T into $15 \cdot 4 = 60$ subtables of size 2^{16}, denoted as $T_{\delta,j}$. For each subtable $T_{\delta,j}$, we calculate and stored output difference

$$\Delta h = h(x_1, ..., x_4) - h(x'_1, \ldots, x'_4),$$

where $x'_j - x_j = \delta$ and $x'_i = x_i, i \neq j$, for all $(x_1, \ldots, x_4) \in \mathbb{Z}_{16}^4$.

Next, we inject the $\mathbb{Z}_{16}$-based fault into the key register and collect L normal and faulty keystreams. With 100000 random IVs and faults, the average number of keystreams required to determine the fault position is 18.82, indicating a low cost for identifying the faulty key position. When the fault occurs in a linearly represented variable, we determine the fault value using the output difference from Equation (4), bypassing the need for a guess-and-check strategy, as described in [WT24]. This approach significantly reduces the time complexity of the DFA.

In the process of generating merging paths, we conducted comparative experiments using both the original GA and MIP-GA. The results indicate that MIP-GA performs significantly better than the original GA. The specific experimental results are listed in Table 3. When the number of keystreams is 3000, the merging path generated by the original GA are unable to recover the correct key within a reasonable time, whereas the path generated by MIP-GA can still achieve successful key recovery. With MIP-GA and our new table-based framework, we can mount a DFA with 10000 keystreams and 38 seconds[3], which outperforms than the DFA proposed in [WT24] both in the number of keystreams and runtime. Additionally, a practical DFA with only 3000 keystreams, which is one-fifth of that in [WT24], is also provided.

4.2 Application to Gabriel-4

As detailed in Section 2.2, the filter function of Gabriel is

$$f(x_1, \ldots, x_{110}) = \sum_{i=0}^{8} g_L(x_{5i+1}, \ldots, x_{5i+5}) + \sum_{i=0}^{9} g_R(x_{7i+41}, \ldots, x_{7i+47}).$$

It corresponds to a DS-GFP cipher with $d = 2$, and the shortcut attack uses only the 5-to-1 function g_L as function h, which is the one already studied for Elisabeth-4 in Section 4.1.

$$\Delta f(x_1, \ldots, x_{110}) = \sum_{i=0}^{8} \Delta g_L(x_{5i+1}, \ldots, x_{5i+5}).$$

The difference between the two shortcut attacks lies in the fact that the key space of Gabriel is larger, and the proportion of valid keystreams is lower. Specifically, the key space increase from 2^{1024} to 2^{2048}, and the average proportion of valid keystreams decreased from $\frac{4\cdot 12}{256} = \frac{3}{16}$ to $\frac{4\cdot 8}{512} = \frac{1}{16}$. Therefore, when implementing the DFA on Gabriel-4, it requires a greater amount of keystreams.

Similar to the DFA on Elisabeth-4, we use both the original GA and MIP-GA to generate merging paths and perform key recovery. The results is displayed in Table 3. As predicted, we observe the number of keystreams required to attack Gabriel-4 is significantly higher than for Elisabeth-4. Additionally, when the estimated $|S_{max}|$ is the same as for Elisabeth-4, the time needed to attack Gabriel-4 is also longer. These results indicate that Gabriel-4 has stronger resistance to the shortcut DFA compared to Elisabeth-4. Nonetheless, we successfully implemented the DFA on Gabriel-4.

4.3 Application to Elisabeth-b4

The parameters of Elisabeth-b are summarized in Section 2.2. Compared to the original Elisabeth, this patch uses an inner function h with 6 variables from $\mathbb{Z}_{16}$ instead of 4. The increase in the number of input variables m directly affects the size of the filter table T. In the previous two attacks, the table size was $4 \cdot 2^{4\cdot 5} = 2^{22}$, whereas for the attack on Elisabeth-b4, the size of T grows to $6 \cdot 2^{4\cdot 7} = 2^{30.38}$. Additionally, the size of the candidate key set derived from T for a given output difference increases from 2^{12} to 2^{20}. The most significant impact of increasing m is the need for more common positions during merging to ensure the candidate key set remains manageable. According to Proposition 1, when the sizes of two input sets are $|S_{tmp}|$ and 2^{20}, and the number of common positions is ℓ, the size of the intersection set will be $|S_{tmp}| \cdot 2^{20-4\ell}$. Thus, to maintain the size of the output set close to $|S_{tmp}|$, at least 5 common positions are needed, compared to 3 for Elisabeth-4. As a result, more keystreams are required to execute a feasible

[3]Although only the time for key filtering is provided here, the actual runtime for the entire attack (including fault determination and path generation) does not exceed 45 seconds.

Table 3: Shortcut attacks on Elisabeth-4, Gabriel-4, and Elisabeth-b4 with different GA

Cipher	#keystream	Method	$\lvert S_{max}\rvert$[a]	Time[b]
Elisabeth-4	10000	Original GA	2^{16}	651s
		MIP-GA	2^{12}	38s
	7500	Original GA	2^{16}	706s
		MIP-GA	2^{12}	63s
	4000	Original GA	2^{20}	15403s
		MIP-GA	2^{16}	1303s
	3000	Original GA	2^{36}	-
		MIP-GA	2^{20}	17215s
Gabriel-4	100000	Original GA	2^{16}	2027s
		MIP-GA	2^{12}	202s
	80000	Original GA	2^{20}	18672s
		MIP-GA	2^{12}	336s
	60000	Original GA	2^{28}	-
		MIP-GA	2^{12}	406s
	40000	Original GA	2^{28}	-
		MIP-GA	2^{20}	35757s
Elisabeth-b4[c]	500000	Original GA	2^{72}	$2^{103.58}$
		MIP-GA	2^{68}	$2^{99.58}$
	1000000	Original GA	2^{60}	$2^{91.58}$
		MIP-GA	2^{56}	$2^{87.58}$
	1500000	Original GA	2^{56}	$2^{87.58}$
		MIP-GA	2^{48}	$2^{79.58}$
	2000000	Original GA	2^{48}	$2^{79.58}$
		MIP-GA	2^{44}	$2^{75.58}$

[a] The maximum size is estimated by Proposition 1

[b] The time is for filtering the candidate key set, and "-" means that we cannot obtain a result with a reasonable time in practice

[c] The DFA on Elisabeth-b4 is theoretical and the time complexity is computed by $O(mN|S_{max}||\mathbb{G}|^{m-1})$

shortcut attack on Elisabeth-b4. If the number of keystreams is insufficient, the size of the candidate key set will grow rapidly during the filtering process, leading to an unacceptably large $|S_{max}|$.

We attempt to apply the shortcut attack to Elisabeth-b4 with different numbers of keystreams and estimate the time complexity based on the merging path and Proposition 1. The results are shown in Table 3. When using 500000 keystreams, we successfully obtain a theoretical DFA with a time complexity of $O(2^{99.58})$. As the number of keystreams increases, we can derive a more efficient merging path, reducing the time required for filtering keys. For instance, with 2000000 keystreams, we generate a merging path where the estimated $|S_{max}|$ is 2^{44}, resulting in an attack with a time complexity of $O(2^{75.58})$. It is important to note that, as the number of keystreams increases, the filtering time decreases, but the time required to generate the merging path increases. For example, when the number of keystreams reaches 2^{21}, a single execution of the greedy algorithm requires tens of thousands of seconds. Therefore, when the number of keystreams surpasses a certain threshold, the cost of generating the merging paths may exceed the cost of filtering and become the primary overhead of the attack.

5 DFAs on Margrethe-18-4 with Multiple Faults

The MFP paradigm differs from the GFP by the use of two different groups. Mixing operation from two different structures to get a secure cryptographic primitive is a principle that has been widely used [BIP+18, DR20, DMMS21, DGH+21, HMM+23]. Margrethe-18-4 is the MFP paradigm instantiated with parameters $\mathbb{G}_1 = \mathbb{F}_2, \mathbb{G}_2 = \mathbb{Z}_{16}, N = 2048, n = 308, t = 14, r = 22, m = 18$ and the filter function

$$f(x_1, \ldots, x_{308}) = \sum_{i=0}^{13} h(x_{22i+1}, \ldots, x_{22i+18}) + \mathbb{Z}_{16}(\sum_{k=0}^{3} 2^k x_{22i+19+k}).$$

The large values of N and m prevent us from directly applying the shortcut attack described in Section 3 to Margrethe-18-4. However, with a stronger attacker, we can mount a DFA on Margrethe-18-4 using multiple faults. In this section, we assume that the attacker is capable of injecting faults multiple times at different positions[4]. This assumption is crucial for the presented DFA.

Since the key of Margrethe-18-4 is over $\mathbb{F}_2$, the injected fault is bit-based, with the fault value always being 1 (*i.e.* a bit-flip). When the fault occurs at the input of h, we can compute and store a filter table T containing $18 \cdot 2^{18} = 2^{22.17}$ elements. However, when attempting to use the filter table T for a shortcut attack, the large number of inputs creates a significant challenge in generating an effective merging path. The insufficient number of common key positions during each merge leads to a rapid expansion of the candidate key set. As a result, a direct single-fault shortcut attack cannot be applied to Margrethe.

Unlike previous target ciphers following the GFP paradigm, Margrethe-18-4 adopts the MFP paradigm, where the input and output groups of filter function are different. Consequently, the $\mathbb{Z}_{16}$ transformation is also nonlinear over $\mathbb{F}_2$, which means that we can extract key information through the differences in the $\mathbb{Z}_{16}$ transformation. When the fault lies at the input of the $\mathbb{Z}_{16}$ transformation, we have:

$$\Delta\mathbb{Z}_{16} = \mathbb{Z}_{16}(x'_1, x'_2, x'_3, x'_4) - \mathbb{Z}_{16}(x_1, x_2, x_3, x_4) = \sum_{i=1}^{4}(x'_i - x_i)2^{i-1} \bmod 16, \tag{6}$$

where (x'_1, x'_2, x'_3, x'_4) is the faulty input. Based on Equation (6) and the fact that the fault affects only one bit of the state—where a bit-flip on the variable x_i changes it to $1 - x_i$—we can derive the following six relationships:

$$\Delta\mathbb{Z}_{16} = 15 \Rightarrow x_1 = 1, \Delta\mathbb{Z}_{16} = 1 \Rightarrow x_1 = 0,$$
$$\Delta\mathbb{Z}_{16} = 14 \Rightarrow x_2 = 1, \Delta\mathbb{Z}_{16} = 2 \Rightarrow x_2 = 0,$$
$$\Delta\mathbb{Z}_{16} = 12 \Rightarrow x_3 = 1, \Delta\mathbb{Z}_{16} = 4 \Rightarrow x_3 = 0.$$

[4] In practice we assume the adversary can flip one bit of the key state, collect the corresponding keystreams, and then reapply the same attack on an unfaulty state. That is, the number of faults in the state is always one.

Furthermore, when the fault occurs at x_4, regardless of the value of (x_1, x_2, x_3, x_4), the output difference of the $\mathbb{Z}_{16}$ transformation is always 8, which means this difference cannot be used to recover the faulted key element. In the other three cases, a single fault is sufficient to determine the value of the faulted variable. If the attacker can precisely inject each fault, by injecting $N = 2048$ faults, they can recover the values of all key bits. Since the probability of a fault occurring on x_1, x_2, or x_3 in the $\mathbb{Z}_{16}$ transformation is $\frac{3 \cdot 14}{2048}$, the total number of keystreams required for the attack is, on average, $\frac{2048 \cdot 2048}{3 \cdot 14} \approx 99865$.

While the aforementioned attack can quickly recover the key, it requires a highly precise fault inject method and a large number of faults. To reduce the number of faults, we combine this approach with the shortcut attack. First, we use the output difference of the $\mathbb{Z}_{16}$ transformation to determine partial key information by injecting multiple faults. Then, we inject an additional fault and use the output difference of h to recover the remaining part of the key. Through experiments, we found that with 700 faults—determining the values of 699 key bits— we can use the shortcut attack to recover the remaining 1349 key bits. As a result, we successfully implemented a DFA with 700 faults, 230000 normal and faulty keystreams, and a runtime of approximately 30000 seconds.

6 Towards Security Margins to Avoid Shortcut Attacks

In this section, we examine the security margins of DS-MFP ciphers, a generalization of DS-GFP ciphers that follow the MFP paradigm, to mitigate the risk of shortcut attacks. As analyzed in previous sections, shortcut attacks become infeasible if the number of keystreams is insufficient to generate a path that keeps the size of the candidate key set within a manageable range. For any path, the first intersection involves combining two solution sets derived from valid keystreams. If the candidate key set grows after the first intersection, it will continue to expand rapidly, making key recovery impractical. Therefore, we estimate the number of keystreams required to find two valid keystreams such that the candidate key set remains unchanged after the first intersection. We expect this estimate to provide valuable insights for DS-MFP cipher designers in selecting appropriate parameters to prevent shortcut attacks.

Margrethe follows the MFP paradigm and its filter function is the direct sum of functions in small number of variables. We refer to a subclass of ciphers following MFP as DS-MFP ciphers, the ciphers following the MFP paradigm as defined in Section 2.2 with the following restrictions on f:

- f is the direct sum (over $\mathbb{G}_2$) of d distinct functions f_i for $i \in [d]$,
- each f_i is the direct sum of t_i times the function g_i,
- g_i is the direct sum of a nonlinear function h_i and a direct $\mathbb{G}_2$-transformation.

Equivalently:

$$f(x_1, \ldots, x_n) = \sum_{i=0}^{d-1} f_i(x_{\phi_i+1}, \ldots, x_{\phi_{i+1}}) = \sum_{i=0}^{d-1} \sum_{j=0}^{t_i-1} g_i(x_{\phi_i+jr_i+1}, \ldots, x_{\phi_i+(j+1)r_i}),$$

where $\phi_i = \sum_{\ell=0}^{i-1} n\ell, n_i = r_i t_i, n = \sum_{i=0}^{d-1} n_i$ and

$$g_i(x_1, \ldots, x_{r_i}) = h_i(x_1, \ldots, x_{m_i}) + \mathbb{G}_2(x_{m_i+1}, \ldots, x_{r_i}),$$

where $h_i : \mathbb{G}_1^{m_i} \mapsto \mathbb{G}_2$. $\mathbb{G}_2(x_1, \ldots, x_{r_i-m_i}) = \mathbb{G}_2(\sum_{k=0}^{r_i-m_i-1} x_k |\mathbb{G}_1|^k)$ denotes the element of $\mathbb{G}_2$ with $|\mathbb{G}_1|$-ary representation $(x_1, \ldots, x_{r_i-m_i})$ and its output can be viewed as adding a linear variable over $\mathbb{G}_2$. DS-GFP ciphers are a special case of DS-MFP ciphers where $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$. Due to the similar structure between DS-GFP ciphers and DS-MFP ciphers, shortcut attacks can be applied to DS-MFP ciphers in the same manner.

From the analysis of Section 3, the complexity of a shortcut attack depends on the maximum size of the candidate key set, $|S_{max}|$, which is influenced by the merging paths. In the following, we examine the impact of different parameters on this resistance. In a shortcut attack, the attacker uses the inner function with the fewest input variables to compute the filter table T. Let g denote the inner function with the

fewest input variables, and let its nonlinear component be represented by h. Additionally, let m represent the number of input variables for h, and let t denote the number of times h appears in the filter function f of the DS-MFP cipher. Given this, generalizing Assumption 1 we can estimate the average size of the candidate key sets derived from the filter table T as $|\bar{S}| = \frac{|\mathbb{G}_1|^m}{|\mathbb{G}_2|}$.

To start the merging path step, the attacker needs to identify a pair of valid keystreams to merge and intersect. The related key positions for a valid keystream form an m-out-of-N subset, denoted as $\{x_1, x_2, \ldots, x_m\} \in [N]^m$. Given two valid keystreams, if their related key positions have ℓ common elements in their m-out-of-N subsets, the size of the output set after intersection can be calculated as:

$$|S| = \frac{|\bar{S}|^2}{|\mathbb{G}_1|^\ell} = \frac{|\mathbb{G}_1|^{2m-\ell}}{|\mathbb{G}_2|^2}, \tag{7}$$

following Proposition 1.

The more common key positions they share, the smaller the resulting output set will be. This problem is analogous to the birthday paradox: we want to determine how many people (m-out-of-N subsets) need to be randomly chosen to have a significant probability (e.g., 0.5) of finding two that share a birthday (i.e., have ℓ common key positions).

Property 1 (Birthday paradox)**.** Given q integers taken uniformly at random in the range $[n]$, if $q = O(\sqrt{n})$, then at least two integers are the same with non-negligible probability.

Based on the birthday paradox, we can derive the following proposition.

Proposition 2. *Given q m-out-of-N subsets drawn uniformly at random from a collection of N objects, if $q = O\left(\frac{\sqrt{\binom{N}{\ell}}}{\binom{m}{\ell}}\right)$, then at least two subsets share a minimum of ℓ objects with non-negligible probability.*

Proof. If two m-out-of-N subsets have a collision of ℓ elements, then there must be an ℓ-out-of-N subset common to both m-out-of-N subsets.

The total number of possible ℓ-out-of-N subsets is $\binom{N}{\ell}$. According to Property 1, having $O\left(\sqrt{\binom{N}{\ell}}\right)$ ℓ-out-of-N subsets taken uniformly at random is sufficient to have at least two identical with non-negligible probability.

Given an m-out-of-N subset, there are $\binom{m}{\ell}$ distinct ℓ-out-of-N subsets. Therefore, with $O\left(\sqrt{\binom{N}{\ell}}\binom{m}{\ell}^{-1}\right)$ random m-out-of-N subsets, we can expect to find at least two ℓ-out-of-N subsets that are identical, i.e., at least two subsets with a collision of ℓ elements, with a non-negligible probability. □

Using Proposition 2, we can estimate the number of keystreams required for an attacker to find two valid keystreams with ℓ common related key positions in their m-out-of-N subsets. Since the faulty key position must be included in each valid m-out-of-N subset, these subsets consist of the faulty key position along with $m-1$ distinct key positions selected from the remaining $N-1$ possible elements. Therefore, the total number of possible m-out-of-N subsets containing the fault position is $\binom{N-1}{m-1}$. To achieve ℓ common key positions, $\ell-1$ common key positions must be found among the $m-1$ non-faulty positions in the m-out-of-N subsets.

Given that the PRNG output is pseudorandom and the fault is injected randomly, according to Proposition 2 we expect to find a pair of subsets with a collision of ℓ elements in a collection of $O\left(\frac{\sqrt{\binom{N-1}{\ell-1}}}{\binom{m-1}{\ell-1}}\right)$ m-out-of-N subsets containing the faulted position. Since the proportion of valid keystreams is $\frac{mt}{N}$, the estimated number of keystreams required, q can be expressed as:

$$q = O\left(\frac{N\sqrt{\binom{N-1}{\ell-1}}}{mt\binom{m-1}{\ell-1}}\right). \tag{8}$$

Based on the previous analysis and Equation (7), if the attacker aims to keep the size of the candidate key set $|S|$ after the first intersection less than or equal to the size of the initial set $|\bar{S}|$, it requires $\ell \geq m - \log_{|\mathbb{G}_1|} |\mathbb{G}_2|$. When $\ell = m - \log_{|\mathbb{G}_1|} |\mathbb{G}_2|$, we have:

$$|S| = \frac{|\mathbb{G}_1|^{2m-\ell}}{|\mathbb{G}_2|^2} = \frac{|\mathbb{G}_1|^m}{|\mathbb{G}_2|} = |\bar{S}|.$$

Substituting $\ell = m - \log_{|\mathbb{G}_1|} |\mathbb{G}_2|$ into Equation (8), we derive the estimated security limit SL as:

$$SL = O\left(\frac{N \sqrt{\binom{N-1}{m-\log_{|\mathbb{G}_1|} |\mathbb{G}_2|-1}}}{mt\binom{m-1}{m-\log_{|\mathbb{G}_1|} |\mathbb{G}_2|-1}} \right). \tag{9}$$

If the available number of keystreams is less than SL, the maximum size $|S_{max}|$ during the shortcut attack will be at least $\frac{|\mathbb{G}_1|^{m+1}}{|\mathbb{G}_2|^3}$. Additionally, the size of the candidate set $|S|$ will keep increasing after the first intersection, as more keystreams will be required to find an m-out-of-N subset with ℓ common positions with the candidate key positions in subsequent intersections. This makes the shortcut attack we presented infeasible. Therefore, we propose using SL as a security limit for DS-MFP ciphers in scenarios where fault attacks are a concern. We encourage designers to choose DS-MFP parameters that ensure SL exceeds the target security level, thereby enhancing protection against shortcut attacks.

The concrete values of SL for several DS-MFP ciphers are shown in Table 2. The results demonstrate that Elisabeth-4 and Gabriel-4 have relatively low SL values, making them vulnerable to practical shortcut attacks. In contrast, Elisabeth-b4 shows a slightly higher SL, which prevents practical shortcut attacks. However, theoretical shortcut attacks that exceed its 128-bit security claim under the black-box model remain feasible. For Margrethe-18-4, the SL is significantly higher than the other ciphers, making a single-fault shortcut attack infeasible. Additionally, all the DFAs involving a single fault, as shown in Table 1, require more keystreams than the corresponding SL, confirming the effectiveness of our proposed security limit. Finally, from Table 2 and Equation (9), we observe that increasing m provides a greater security boost against shortcut DFA for DS-MFP ciphers than increasing N or decreasing t.

Varying countermeasures result in different costs for the performance of HHE. The choice of t and N is always a trade-off between security and efficiency, with efficiency being highly dependent on advancements in TFHE or other FHE schemes. Increasing N is considered a low-cost countermeasure countermeasure, since it will impact only the initialization phase of the HHE protocol, and this cost becomes negligible if the client sends a large amount of data (the baseline for using FHE). On the other hand, decreasing t while maintaining security requires one to use a function acting on more inputs, which has a greater impact since programmable boostrapping (PBS) for larger input size takes more times (with current techniques). Alternatively, using x layers of the same size of PBS instead of y results in HHE being approximately $\frac{x}{y}$ times slower.

7 Conclusion

In this work, we presented a table-based DFA framework, called the shortcut attack, which extends the attack in [WT24] from Elisabeth-4 to any DS-GFP cipher. Our new framework improves both fault identification and path generation. For fault identification, we propose using the linear part of the inner function to determine the fault's value. This method eliminates the need for a guess-and-check approach, directly reducing the attack complexity to $\frac{2}{|\mathbb{G}|}$ of the original, unimproved attack.

For path generation, we introduced the MIP-GA to replace the original greedy algorithm. This enhancement allows us to find better merging paths using the same number of keystreams, significantly reducing the complexity of key recovery. Additionally, we provided a theoretical complexity analysis for each step in the framework, enabling attackers to make more informed trade-offs when mounting the attack.

We also implemented shortcut attacks on Elisabeth-4 and its patched versions, achieving state-of-the-art DFA results for these ciphers. For Elisabeth-4, we improved upon previous attacks, reducing both runtime and the required number of keystreams. For Gabriel-4, we successfully mounted practical DFAs.

For Elisabeth-b4, we provided various theoretical DFAs with time complexities significantly lower than the security claim under the black-box model. All the aforementioned DFAs were single-fault attacks. While such a single-fault DFA is not feasible for Margrethe-18-4, we presented a multi-fault DFA under a more powerful assumption.

Lastly, we analyzed the security margins of more general DS-MFP ciphers to mitigate shortcut attacks and proposed a secure limit on the number of keystreams. These findings can serve as parameter recommendations for DS-MFP cipher designers, helping them create ciphers that are resistant to shortcut attacks.

Finally, we mention three open questions:

- As analyzed in Section 3, the efficiency of the shortcut attack relies on the direct sum of functions with a small number of variables. Can the shortcut attack be adapted to apply to MFP ciphers outside the DS-MFP family?
- While the table-based approach circumvents the problem of non-representation, it incurs a high memory cost when the input space of the target function is large. Given a large function composed of several smaller functions or S-boxes (like the 6-to-1 function h of Elisabeth-b4 is constructed with several 1-to-1 functions), is it possible to obtain information about the larger function by storing multiple tables for the smaller functions or S-boxes?
- Unlike previous DFAs on stream ciphers used in HHE frameworks, the SASTA framework [ADSR24] leverages the specificities of the full protocol to execute the attack. This raises an intriguing question: Are there other ways to exploit higher-level protocols to enhance existing attack vectors or develop new ones?

Acknowledgments

We are very grateful for the insightful comments and suggestions from the anonymous reviewers that improved the technical as well as editorial quality of this paper. The work of Pierrick Méaux was funded by the European Research Council (ERC) under the Advanced Grant program (reference number: 787390). The work of Deng Tang was supported in part by the National Natural Science Foundation of China (Nos. 62272303, 12101404).

References

[ACG+19] Martin R. Albrecht, Carlos Cid, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, and Markus Schofnegger. Algebraic cryptanalysis of STARK-friendly designs: Application to MARVELLous and MiMC. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 371–397. Springer, Cham, December 2019.

[ADSR24] Aikata Aikata, Ahaan Dabholkar, Dhiman Saha, and Sujoy Sinha Roy. SASTA: Ambushing hybrid homomorphic encryption schemes with a single fault. Cryptology ePrint Archive, Report 2024/041, 2024.

[AK96] Ross Anderson and Markus Kuhn. Tamper resistance-a cautionary note. In *Proceedings of the second Usenix workshop on electronic commerce*, volume 2, pages 1–11, 1996.

[BBL+24] Augustin Bariant, Aurélien Boeuf, Axel Lemoine, Irati Manterola Ayala, Morten Øygarden, Léo Perrin, and Håvard Raddum. The algebraic FreeLunch: Efficient Gröbner basis attacks against arithmetization-oriented primitives. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part IV*, volume 14923 of *LNCS*, pages 139–173. Springer, Cham, August 2024.

[BBLP22] Augustin Bariant, Clémence Bouvier, Gaëtan Leurent, and Léo Perrin. Algebraic attacks against some arithmetization-oriented primitives. *IACR Trans. Symm. Cryptol.*, 2022(3):73–101, 2022.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 37–51. Springer, Berlin, Heidelberg, May 1997.

[BHJ+18] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. Practical fault attack on deep neural networks. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 2204–2206. ACM, 2018.

[BIP+18] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. Exploring crypto dark matter: New simple PRF candidates and their applications. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 699–729. Springer, Cham, November 2018.

[BMS12] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A differential fault attack on the Grain family of stream ciphers. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 122–139. Springer, Berlin, Heidelberg, September 2012.

[Buc65] Bruno Buchberger. Ein algorithmus zum auffinden der basiselemente des restklassenringes nach einem nulldimensionalen polynomideal. *Ph. D. Thesis, Math. Inst., University of Innsbruck*, 1965.

[CCH+24] Mingyu Cho, Woohyuk Chung, Jincheol Ha, Jooyoung Lee, Eun-Gyeol Oh, and Mincheol Son. FRAST: tfhe-friendly cipher based on random s-boxes. *IACR Trans. Symmetric Cryptol.*, 2024(3):1–43, 2024.

[CDPP22] Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder V. L. Pereira. SortingHat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 563–577. ACM Press, November 2022.

[CHMS22] Orel Cosseron, Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. Towards case-optimized hybrid homomorphic encryption - featuring the elisabeth stream cipher. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 32–67. Springer, Cham, December 2022.

[CM03] Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 345–359. Springer, Berlin, Heidelberg, May 2003.

[Cou03] Nicolas Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 176–194. Springer, Berlin, Heidelberg, August 2003.

[DGH+21] Itai Dinur, Steven Goldfeder, Tzipora Halevi, Yuval Ishai, Mahimna Kelkar, Vivek Sharma, and Greg Zaverucha. MPC-friendly symmetric cryptography from alternating moduli: Candidates, protocols, and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 517–547, Virtual Event, August 2021. Springer, Cham.

[DLR16] Sébastien Duval, Virginie Lallemand, and Yann Rotella. Cryptanalysis of the FLIP family of stream ciphers. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 457–475. Springer, Berlin, Heidelberg, August 2016.

[DMMS21] Sébastien Duval, Pierrick Méaux, Charles Momin, and François-Xavier Standaert. Exploring crypto-physical dark matter and learning with physical rounding. *IACR TCHES*, 2021(1):373–401, 2021.

[DR20] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition.* Information Security and Cryptography. Springer, 2020.

[Fau99] Jean-Charles Faugere. A new efficient algorithm for computing gröbner bases (f4). *Journal of pure and applied algebra*, 139(1-3):61–88, 1999.

[Fau02] Jean Charles Faugere. A new efficient algorithm for computing gröbner bases without reduction to zero (f 5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, pages 75–83, 2002.

[FGLM93] Jean-Charles Faugere, Patrizia Gianni, Daniel Lazard, and Teo Mora. Efficient computation of zero-dimensional gröbner bases by change of ordering. *Journal of Symbolic Computation*, 16(4):329–344, 1993.

[FT09] Toshinori Fukunaga and Junko Takahashi. Practical fault attack on a cryptographic LSI with ISO/IEC 18033-3 block ciphers. In Luca Breveglieri, Israel Koren, David Naccache, Elisabeth Oswald, and Jean-Pierre Seifert, editors, *Sixth International Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2009, Lausanne, Switzerland, 6 September 2009*, pages 84–92. IEEE Computer Society, 2009.

[GAH+23] Lorenzo Grassi, Irati Manterola Ayala, Martha Norberg Hovd, Morten Øygarden, Håvard Raddum, and Qingju Wang. Cryptanalysis of symmetric primitives over rings and a key recovery attack on rubato. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part III*, volume 14083 of *LNCS*, pages 305–339. Springer, Cham, August 2023.

[GBJR23] Henri Gilbert, Rachelle Heim Boissier, Jérémy Jean, and Jean-René Reinhard. Cryptanalysis of elisabeth-4. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part III*, volume 14440 of *LNCS*, pages 256–284. Springer, Singapore, December 2023.

[GGM24] François Gérard, Agnese Gini, and Pierrick Méaux. Toolip: How to find new instances of filip cipher with smaller key size and new filters. In Serge Vaudenay and Christophe Petit, editors, *Progress in Cryptology - AFRICACRYPT 2024 - 15th International Conference on Cryptology in Africa, Douala, Cameroon, July 10-12, 2024, Proceedings*, volume 14861 of *Lecture Notes in Computer Science*, pages 21–45. Springer, 2024.

[HMM+23] Clément Hoffmann, Pierrick Méaux, Charles Momin, Yann Rotella, François-Xavier Standaert, and Balazs Udvarhelyi. Learning with physical rounding for linear and quadratic leakage functions. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part III*, volume 14083 of *LNCS*, pages 410–439. Springer, Cham, August 2023.

[HMS23] Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. The patching landscape of elisabeth-4 and the mixed filter permutator paradigm. In Anupam Chattopadhyay, Shivam Bhasin, Stjepan Picek, and Chester Rebeiro, editors, *INDOCRYPT 2023, Part I*, volume 14459 of *LNCS*, pages 134–156. Springer, Cham, December 2023.

[HR08] Michal Hojsík and Bohuslav Rudolf. Differential fault analysis of Trivium. In Kaisa Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 158–172. Springer, Berlin, Heidelberg, February 2008.

[HS04] Jonathan J. Hoch and Adi Shamir. Fault analysis of stream ciphers. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 240–253. Springer, Berlin, Heidelberg, August 2004.

[JLHG24] Lin Jiao, Yongqiang Li, Yonglin Hao, and Xinxin Gong. Differential fault attacks on privacy protocols friendly symmetric-key primitives: Rain and hera. *IET Information Security*, 2024(1):7457517, 2024.

[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.

[LKSM24] Fukang Liu, Abul Kalam, Santanu Sarkar, and Willi Meier. Algebraic attack on FHE-friendly cipher HERA using multiple collisions. *IACR Trans. Symm. Cryptol.*, 2024(1):214–233, 2024.

[MBB11] Mohamed Saied Emam Mohamed, Stanislav Bulygin, and Johannes Buchmann. Using SAT solving to improve differential fault analysis of trivium. In Tai-Hoon Kim, Hojjat Adeli, Rosslin John Robles, and Maricel O. Balitanas, editors, *Information Security and Assurance - International Conference, ISA 2011, Brno, Czech Republic, August 15-17, 2011. Proceedings*, volume 200 of *Communications in Computer and Information Science*, pages 62–71. Springer, 2011.

[MCJS19] Pierrick Méaux, Claude Carlet, Anthony Journault, and François-Xavier Standaert. Improved filter permutators for efficient FHE: Better instances and implementations. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *INDOCRYPT 2019*, volume 11898 of *LNCS*, pages 68–91. Springer, Cham, December 2019.

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards stream ciphers for efficient FHE with low-noise ciphertexts. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 311–343. Springer, Berlin, Heidelberg, May 2016.

[MPP24] Pierrick Méaux, Jeongeun Park, and Hilder V. L. Pereira. Towards practical transciphering for FHE with setup independent of the plaintext space. *CiC*, 1(1):20, 2024.

[MR24] Pierrick Méaux and Dibyendu Roy. Theoretical differential fault attacks on FLIP and filip. *Cryptogr. Commun.*, 16(4):721–744, 2024.

[MSS17] Subhamoy Maitra, Akhilesh Siddhanti, and Santanu Sarkar. A differential fault attack on plantlet. *IEEE Trans. Computers*, 66(10):1804–1808, 2017.

[NDE22] Marcel Nageler, Christoph Dobraunig, and Maria Eichlseder. Information-combining differential fault attacks on DEFAULT. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 168–191. Springer, Cham, May / June 2022.

[NLV11] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In Christian Cachin and Thomas Ristenpart, editors, *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, pages 113–124. ACM, 2011.

[PQ03] Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 77–88. Springer, Berlin, Heidelberg, September 2003.

[RBM21] Dibyendu Roy, Bhagwan N. Bathe, and Subhamoy Maitra. Differential fault attack on kreyvium & FLIP. *IEEE Trans. Computers*, 70(12):2161–2167, 2021.

[RKMR23] R. Radheshwar, Meenakshi Kansal, Pierrick Méaux, and Dibyendu Roy. Differential fault attack on rasta and $\text{FiLIP}_{\text{DSM}}$. *IEEE Trans. Computers*, 72(8):2418–2425, 2023.

[Rog04] Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 348–359. Springer, Berlin, Heidelberg, February 2004.

[SC16a] 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, *CHES 2016*, volume 9813 of *LNCS*, pages 581–601. Springer, Berlin, Heidelberg, August 2016.

[SC16b] Dhiman Saha and Dipanwita Roy Chowdhury. Scope: On the side channel vulnerability of releasing unverified plaintexts. In Orr Dunkelman and Liam Keliher, editors, *SAC 2015*, volume 9566 of *LNCS*, pages 417–438. Springer, Cham, August 2016.

[SKC14] Dhiman Saha, Sukhendu Kuila, and Dipanwita Roy Chowdhury. EscApe: Diagonal fault analysis of APE. In Willi Meier and Debdeep Mukhopadhyay, editors, *INDOCRYPT 2014*, volume 8885 of *LNCS*, pages 197–216. Springer, Cham, December 2014.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.

[WT24] Weizhe Wang and Deng Tang. Differential fault attack on HE-friendly stream ciphers: Masta, pasta and elisabeth. Cryptology ePrint Archive, Report 2024/1005, 2024.

[YZY+24] Hong-Sen Yang, Qun-Xiong Zheng, Jing Yang, Quan feng Liu, and Deng Tang. A new security evaluation method based on resultant for arithmetic-oriented algorithms. Cryptology ePrint Archive, Report 2024/886, 2024.

A The Specification of Elisabeth-b4

Elisabeth-b4 is the GFP paradigm instantiated with:

- $\mathbb{G} = \mathbb{Z}_{16}, N = 512, n = 98$,
- the filter function $f(x_1, \ldots, x_{98})$ is the direct sum of 14 times the 7-to-1 function g, which can be expressed as:
$$f(x_1, \ldots, x_{98}) = \sum_{i=0}^{13} g(x_{7i+1}, x_{7i+2}, \ldots, x_{7i+6}, x_{7i+7}),$$
- the 7-to-1 function g is the sum of a nonlinear 6-to-1 function h and the remaining variable, *i.e.*
$$g(x_1, \ldots, x_7) = h(x_1, \ldots, x_6, 0) + x_7.$$

The detailed construction of function h can be found in Appendix B.2.

B The Concrete Constructions of Nonlinear Functions

B.1 Function h in Elisabeth-4

The construction of function h in Elisabeth-4 is described in Figure 5.

All the look-up tables $S_1, \ldots, S_8$ over $\mathbb{Z}_{16}$ were selected at random, their descriptions can be found in [CHMS22][Appendix B].

B.2 Function h in Elisabeth-b4

The detailed construction of the nonlinear function h for Elisabeth-b4 is given in Algorithm 6. The generation of the Sboxes can be found in [HMS23].

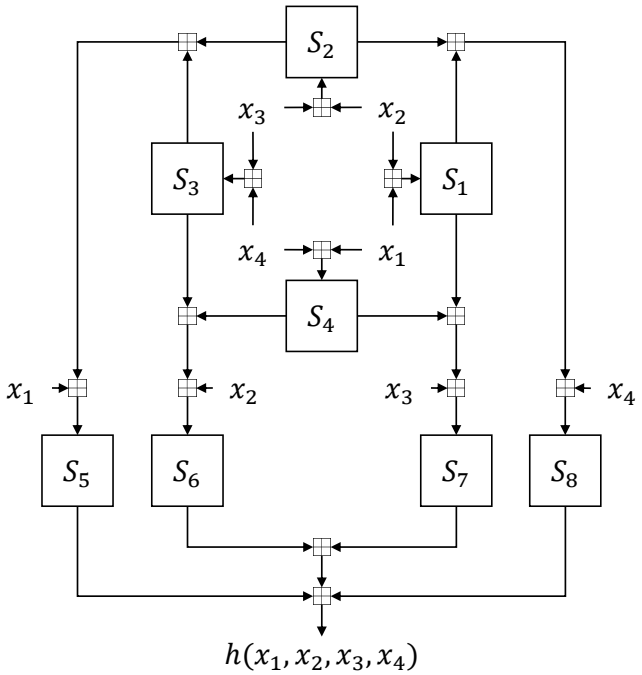

Figure 5: The construction of function h for Elisabeth-4.

C An Example for Different Merging Paths

Suppose the secret key $(k_1, k_2, k_3, k_4) \in \mathbb{Z}_4^4$ and we have the following 4 relations:

$$\begin{aligned} R1 &: k_1 + k_2 = 1, \\ R2 &: k_2 + k_3 = 3, \\ R3 &: k_3 + k_4 = 1, \\ R4 &: k_1 + k_3 = 2. \end{aligned}$$

Then the corresponding solution spaces are:

$$\begin{aligned} S_1(k_1, k_2) &= \{(0,1),(1,0),(2,3),(3,2)\}, \\ S_2(k_2, k_3) &= \{(1,2),(2,1),(0,3),(3,0)\}, \\ S_3(k_3, k_4) &= \{(0,1),(1,0),(2,3),(3,2)\}, \\ S_4(k_1, k_3) &= \{(0,2),(2,0),(1,1),(3,3)\}, \end{aligned}$$

If we follow the merging path $P1 = R1 \to R2 \to R4 \to R3$, then the candidate key set will be:

$$\begin{aligned} & C(k_1, k_2) = \{(0,1),(1,0),(2,3),(3,2)\} \\ \to & C(k_1, k_2, k_3) = \{(0,1,2),(1,0,3),(2,3,0),(3,2,1)\} \\ \to & C(k_1, k_2, k_3) = \{(0,1,2),(2,3,0)\} \\ \to & C(k_1, k_2, k_3, k_4) = \{(0,1,2,3),(2,3,0,1)\}. \end{aligned}$$

Algorithm 6 Elisabeth-b4 nonlinear function h

Input: $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) \in \mathbb{Z}_{16}^7$
Output: $z \in \mathbb{Z}_{16}$

for $i = 1, 2, 3$ **do**
  $x_{2i} = x_{2i} + x_{2i-1}$
end for
for $i = 1, 2, 3, 4, 5, 6$ **do**
  $y_i = S_i(x_i)$
end for
for $i = 0, 1, 2$ **do**
  $z_{2i+1} = y_{2i \bmod 6} + y_{2i+1}$
  $z_{2i+2} = y_{2i+5 \bmod 6} + y_{2i+2}$
end for
for $i = 1, 2, 3, 4, 5, 6$ **do**
  $z_i = z_i + x_{i+2 \bmod 6}$
  $z_i = S_{i+6}(z_i)$
end for
for $i = 0, 1$ **do**
  $t_{3i+1} = z_{3i+1} + z_{3i+2} + z_{3i+3}$
  $t_{3i+2} = z_{3i+2} + z_{3i+4 \bmod 6}$
  $t_{3i+3} = z_{3i+3} + z_{3i+4 \bmod 6} + y_{3i+1}$
end for
$t_1 = t_1 + x_6$
$t_2 = t_2 + x_5$
$t_3 = t_3 + x_4$
$t_4 = t_4 + x_2$
$t_5 = t_5 + x_1$
$t_6 = t_6 + x_3$
for $i = 1, 2, 3, 4, 5, 6$ **do**
  $u_i = S_{i+12}(t_i)$
  $z = z + u_i$
end for
return z

The maximum size of candidate key set C is 4 and the total number of intersection is 40. We can also follow another merging path $P2 = R1 \to R3 \to R4 \to R2$. then the candidate key set will be:

$$\begin{aligned}
&C'(k_1, k_2) = \{(0,1),(1,0),(2,3),(3,2)\}\\
\to &C'(k_1, k_2, k_3, k_4) = \{(0,1,0,1),(0,1,1,0),(0,1,2,3),(0,1,3,2),(1,0,0,1),\\
&(1,0,1,0),(1,0,2,3),(1,0,3,2),(2,3,0,1),(2,3,1,0),(2,3,2,3),\\
&(2,3,3,2),(3,2,0,1),(3,2,1,0),(3,2,2,3),(3,2,3,2)\}\\
\to &C'(k_1, k_2, k_3, k_4) = \{(0,1,2,3),(1,0,1,0),(2,3,0,1),(3,2,3,2)\}\\
\to &C'(k_1, k_2, k_3, k_4) = \{(0,1,2,3),(2,3,0,1)\}.
\end{aligned}$$

Though the final set of C' is the same as C, the maximum size of candidate key set C' is 16 and the total number of intersection is 96. Therefore, filtering the candidate key set with P_2 takes more than twice the time compared to filtering with P_1.