# Differential Fault Attack on HE-Friendly Stream Ciphers: Masta, Pasta and Elisabeth

Weizhe Wang, Deng Tang

**Abstract**

In this paper, we propose the Differential Fault Attack (DFA) on three Homomorphic Encryption (HE) friendly stream ciphers Masta, Pasta, and Elisabeth. Both Masta and Pasta are Rasta-like ciphers with publicly derived and pseudorandom affine layers. The design of Elisabeth is an extension of FLIP and FiLIP, following the group filter permutator paradigm. All these three ciphers operate on elements over $\mathbb{Z}_p$ or $\mathbb{Z}_{2^n}$, rather than $\mathbb{Z}_2$. We can recover the secret keys of all the targeted ciphers through DFA. In particular, for Elisabeth, we present a new method to determine the filtering path, which is vital to make the attack practical. Our attacks on various instances of Masta are practical and require only one block of keystream and a single word-based fault. By injecting three word-based faults, we can theoretically mount DFA on two instances of Pasta, Pasta-3 and Pasta-4. For Elisabeth-4, the only instance of the Elisabeth family, we present two DFAs in which we inject four bit-based faults or a single word-based fault. With 15000 normal and faulty keystream words, the DFA on Elisabeth-4 can be completed in just a few minutes.

**Index Terms**

Differential fault attack, Masta, Pasta, Elisabeth

## I. Introduction

HOMOMORPHIC Encryption (HE) was initially introduced by Rivest et al. in 1978 [1] and quickly emerged as a notable technology in the realms of privacy preservation, cloud computing, and other areas. Recently, there has been an observation that symmetric-key primitives with low multiplicative depth can offer a notable improvement in communication efficiency for HE protocols. As a result, the development of tailored symmetric-key ciphers like those with low multiplicative depth, has emerged as a recent trend in research. In 2015, Albrecht et al. [2] introduced LowMC, marking the first design of a block cipher targeted for HE. Meanwhile, for the realm of HE-friendly stream ciphers, Kreyvium, proposed by Canteaut et al. [3], stands as the pioneer design. Derived from the Trivium cipher, Kreyvium relies on nonlinear feedback shift registers. The FLIP family ciphers were first proposed in 2015 and the updated design was presented at EUROCRYPT 2016 [4]. The design of FLIP is innovative, as its state is updated by a pseudorandom generator (PRG) instead of a feedback function. Subsequently, Méaux et al. introduced FiLIP [5], a new family of stream ciphers that aligns with the filter permutator paradigm of FLIP but employs a more intricate PRG. At CRYPTO 2018, Dobraunig et al. introduced another novel family of stream ciphers named Rasta [6]. The design of Rasta adopts the SPN structure commonly employed in block ciphers, which also differs significantly from the other stream ciphers. These new ciphers have attracted the attention of the community, leading to an emergence of security analysis studies [7]–[11]. In this work, we will study three recent HE-friendly stream ciphers, Masta [12], Pasta [13], and Elisabeth [14].

Masta, the first $\mathbb{Z}_p$ variant of Rasta proposed by Ha et al. in 2020 [12], utilizes modular arithmetic to support HE schemes over a non-binary plaintext space and employs finite field multiplication to establish the affine layers. Consequently, the implementation is significantly more efficient than that of Rasta. Though Masta achieved good results at client-side runtime, its homomorphic runtime is slow in many settings. Pasta is another $\mathbb{Z}_p$ variant of Rasta proposed by Dobraunig et al. at TCHES 2023 [13]. The designers of Pasta proposed a relatively cheap way to generate random matrices and used truncation to prevent the inverse of the last layer. Moreover, Pasta also applies the Feistel $\chi$ function and cubic function as its nonlinear layers, which are more complicated than that of Masta. Recently, Grassi et al. [15] proposed a method to reduce the randomness in Rasta-like Designs and presented a modified version of Pasta called Pasta$_{v2}$. To the best of our knowledge, while the algebraic attack is applied to Rasta [7], there is currently no similar attack on Masta and Pasta.

Elisabeth is a family of HE-friendly stream ciphers introduced by Cosseron et al. at ASIACRYPT 2022 and Elisabeth-4 is the only fully specified instance. Instead of using feedback functions, Elisabeth updates its state with an extendable output function (XOF) like FLIP. In particular, Elisabeth operates in an additive group $\mathbb{Z}_{2^n}$ rather than a binary extension field $\mathbb{F}_{2^n}$, which plays an important role in achieving the goals of designers. The design of Elisabeth extends FiLIP and applies a more intricate filter function. Instead of using the direct sum of monomials or the Xor-Threshold function, Elisabeth's nonlinear component consists of eight negacyclic look-up tables. At ASIACRYPT 2023, Gilbert et al. [16] proposed a linearization attack in classical setting for Elisabeth-4 with $2^{88}$ elementary operations. This is the first and the only third-party attack on Elisabeth.

Weizhe Wang and Deng Tang are with the Shanghai Jiao Tong University, Shanghai 200240, China. (e-mail:{SJTUwwz, dengtang}@sjtu.edu.cn)

The differential fault attack (DFA) was first proposed by Boneh et al. [17] in 1997. The first DFA on stream ciphers was introduced by Hoch and Shamir [18] at CHES 2004. 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, clock glitches, electromagnetic waves, unsupported voltage, etc. After the fault injection, the attacker can notice a distinguishable impact of the introduced difference in the generated keystream. Generally, the attacker does not know the exact location of injected faults. One potential method to pinpoint the fault locations is statistical testing, with the most famous technique being signature-based fault identification [19]. In situations where the statistical data exhibits random, the statistical test may prove ineffective, as is often observed in the context of most HE-friendly ciphers. Recently, Méaux and Roy [20] introduced an additional approach for identifying fault locations in DFA aimed at FLIP and FiLIP. When the location of an injected fault cannot be identified using any technique, the attacker may guess the location to recover the state. This approach would be feasible if the number of injected faults is minimal and the state size of the cipher is small. In our DFA, we will guess the location of faults for Masta and Pasta, and use a simple method to locate the faulty word for Elisabeth.

In the context of DFA on HE-friendly ciphers, there exist several works. In 2020, Roy et al. [21] applied DFA to two stream ciphers: Kreyvium and FLIP, which was the first try of DFA on HE-friendly ciphers. In 2023, Radheshwar et al. [22] mounted DFA on Rasta and FiLIP$_{\mathrm{DSM}}$. Generally, for DFA over $\mathbb{Z}_2$, the attacker will construct Boolean equations and employ the SAT solver to solve them. However, this approach is unsuitable for DFA over $\mathbb{Z}_p$ or $\mathbb{Z}_{2^n}$. In 2024, Jiao et al. [23] introduced DFA on RAIN [24] and HERA [25], where RAIN is a secure multi-party computation (MPC) friendly block cipher and HERA is an HE-friendly stream cipher. The work of Jiao et al. is notable as it is the first DFA over $\mathbb{Z}_p$. Their approach involved injecting a word-based fault and constructing a system of quadratic equations for HERA. How to analyze the security of ciphers over $\mathbb{Z}_p$ or $\mathbb{Z}_{2^n}$ against DFA requires further investigation.

## A. *Our Contributions and Organization of the Article*

TABLE I
SUMMARY OF OUR RESULTS

| Cipher | Scope | Parameters | Type of fault | # Faults | # Keystreams | Time complexity |
|---|---|---|---|---|---|---|
| Masta | $\mathbb{Z}_{2^{16}+1}$ | $n = 32, r = 4, s = 80$ | word-based | 1 | 32 | $2^{35}$ |
| | | $n = 16, r = 5, s = 80$ | | | 16 | $2^{31.2}$ |
| | | $n = 32, r = 6, s = 128$ | | | 32 | $2^{35}$ |
| | | $n = 16, r = 7, s = 128$ | | | 16 | $2^{31.2}$ |
| Pasta | | $n = 256, r = 3, s = 128$ | | 3 | 128 | $2^{97.2}$ |
| | | $n = 64, r = 4, s = 128$ | | | 32 | $2^{85.6}$ |
| Elisabeth | $\mathbb{Z}_{2^4}$ | $N = 256, n = 60, s = 128$ | bit-based | 4 | 15000 | 60 sec |
| | | | word-based | 1 | | 150 sec |

In this paper, we analyze three recent HE-friendly stream ciphers, Masta, Pasta, and Elisabeth. The main contributions of this paper are as follows:

- We first propose the DFA on Masta in Section II. By injecting a word-based fault into the internal state of Masta, we can construct a system of linear equations regarding the internal state and solve it with Gaussian elimination. Subsequently, the secret key can be recovered by a partial encryption. In particular, we can reveal the secret keys of all the common instances of Masta in a practical time ($< 2^{40}$ elementary operations).
- In Section III, we propose the DFA on Pasta. A system of quadratic equations can be constructed by injecting a word-based fault into the internal state of Pasta. According to the characteristic of the equations, we apply linearization and Gaussian elimination for solving them. We observed that apart from the difference between normal and faulty keystreams, we can also construct equations using the differences between different faulty keystreams. Ultimately, we mount the DFA by injecting 3 word-based faults. With our DFA, we can recover the secret key of both two common instances of Pasta theoretically.
- We propose two DFAs on Elisabeth in Section IV. In the case of Elisabeth-4, the secret key of the cipher can be recovered by injecting 4 bit-based faults or a word-based faults. Rather than constructing and solving equations, we use a table to iteratively filter the candidate keys for Elisabeth-4. To effectively recover the key, we present a simple method to locate the fault. To reduce the time and memory complexities, we carefully analyze the structure of Elisabeth-4 and employ a greedy algorithm to generate filtering paths. This is crucial for our DFA as a random path could make the attack impractical. Experimentally, we show that the secret key of Elisabeth-4 can be recovered in a few minutes.

The summary of our contributions is provided in Table I. All the experiments are completed with our personal computer (Intel Core i5-10400 CPU with 6 cores, 2.90 GHz clock, 16 GB memory, Windows 11).

The rest of the article is organized as follows. Section I-B, Section I-C and Section I-D describe the design of Masta, Pasta, and Elisabeth respectively. The common process of DFA is described in Section I-E. The article is finally concluded in Section V.

## B. Design Specification of Masta

Masta [12] is an HE-friendly family of stream ciphers. For each block, Masta would first generate a pseudo-random permutation $\pi_{nc}$ using nonce $nc$ and an XOF, and then take a secret key $\boldsymbol{k} \in \mathbb{Z}_p^n$ as input to produce a block of keystream $\boldsymbol{k}_{nc} \in \mathbb{Z}_p^n$. The $r$-round Masta construction is shown in Fig. 1.

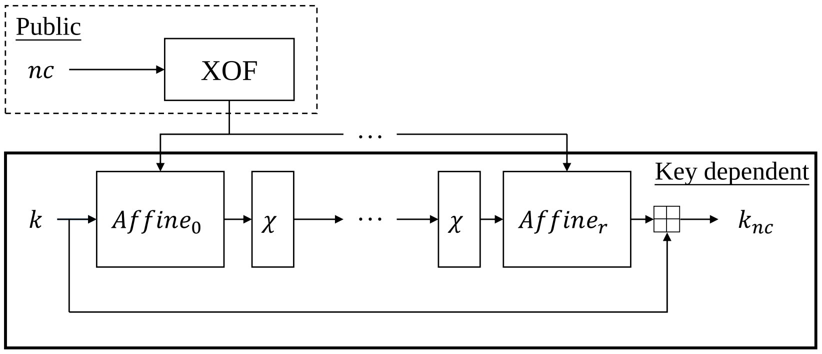

Fig. 1. Design specification of Masta

For an $r$-round Masta, the permutation $\pi_{nc}$ consists of $r+1$ affine layers and $r$ non-linear layers. All the affine layers $Affine_i, i = 0, \ldots, r$ are generated by XOF which can be instantiated with AES in the counter mode or a sponge-type hash function. Each affine layer is decomposed as $Affine_i = ARC_i \circ FMul_i, i = 0, \ldots, r$. Namely,

$$FMul_i(\boldsymbol{x}) = \boldsymbol{a}^{(i)}\boldsymbol{x},$$
$$ARC_i(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{b}^{(i)},$$

where $\boldsymbol{x}, \boldsymbol{b}^{(i)} \in \mathbb{Z}_p^n, \boldsymbol{a}^{(i)} \in \mathbb{Z}_{p^n}$. The field multiplication element $\boldsymbol{a}^{(i)}$ can also be represented by an $n \times n$ invertible matrix $M^{(i)}$. The non-linear layer of Masta is a $\mathbb{Z}_p$-variant of the $\chi$-transformation. Let $\boldsymbol{x} = (x_0, \ldots, x_{n-1}), \boldsymbol{y} = (y_0, \ldots, y_{n-1})$, where $x_i, y_i \in \mathbb{Z}_p, i = 0, \ldots, n-1$. Then $\boldsymbol{y} = \chi(\boldsymbol{x})$ consists of

$$y_i = x_i + x_{i+2} + x_{i+1}x_{i+2},$$

where all the indices taken modulo $n$. After the permutation $\pi_{nc}$, the secret key $\boldsymbol{k}$ is added. The designers choose $p = 2^{16}+1$ and the parameters $n$ and $r$ are given in the Table II. For simplicity, we will use $\boldsymbol{s}_r$ to denote the $r$-th round state. This notation will also be used for Pasta.

TABLE II
PARAMETERS $n$ FOR $r$-ROUND MASTA

| Security (bit) / $r$ | 80 | 128 | 192 | 256 |
|---|---|---|---|---|
| 4 | 32 | 128 | 512 | 2048 |
| 5 | 16 | 64 | 128 | 256 |
| 6 | 16 | 32 | 64 | 128 |
| 7 | 8 | 16 | 32 | 64 |

## C. Design Specification of Pasta

Pasta is an HE-friendly cipher [13] that is thoroughly optimized for integer hybrid HE use cases. The workflow of Pasta is the same as Masta, while Pasta employs a completely different permutation. The $r$-round Pasta construction is shown in Fig. 2.

The permutation Pasta-$\pi(\boldsymbol{x})$ operates on a vector $\boldsymbol{x} = \boldsymbol{x}_L || \boldsymbol{x}_R \in \mathbb{Z}_p^{2t}$ where $||$ represents concatenation, and is defined as:

$$\text{Pasta-}\pi(\boldsymbol{x}) = A_r \circ S_{cube} \circ A_{r-1} \circ S_{feistel} \circ \cdots \circ S_{feistel} \circ A_0(\boldsymbol{x}).$$

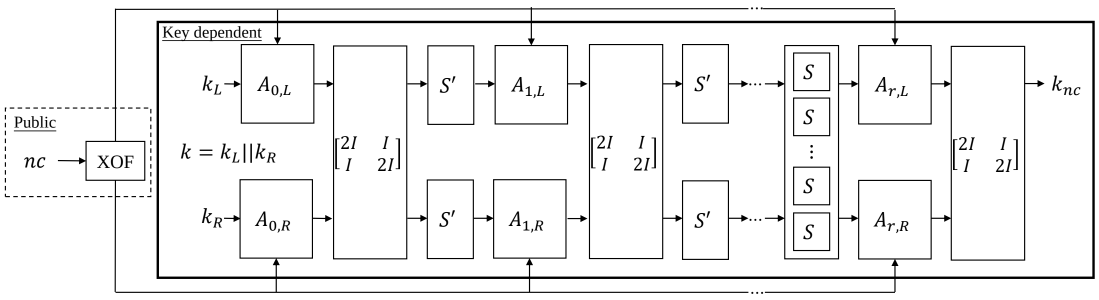

Fig. 2. Design specification of Pasta

For an $r$-round Pasta, the permutation consists of $r+1$ affine layers, $r-1$ $S_{feistel}$ and one $S_{cube}$. For $i = 0, \dots, r$, the affine layer $A_i$ is define as

$$A_i(\boldsymbol{x}) = \begin{bmatrix} 2I & I \\ I & 2I \end{bmatrix} \begin{bmatrix} A_{i,L}(\boldsymbol{x}_L) \\ A_{i,R}(\boldsymbol{x}_R) \end{bmatrix} = \begin{bmatrix} 2I & I \\ I & 2I \end{bmatrix} \begin{bmatrix} M_{i,L}\boldsymbol{x}_L + \boldsymbol{c}_{i,L} \\ M_{i,R}\boldsymbol{x}_R + \boldsymbol{c}_{i,R} \end{bmatrix},$$

where $I \in \mathbb{Z}_p^{t \times t}$ is the identity matrix. Here, $M_{i,L}, M_{i,R} \in \mathbb{Z}_p^{t \times t}$ and $\boldsymbol{c}_{i,L}, \boldsymbol{c}_{i,R} \in \mathbb{Z}_p^t$ are generated from an XOF seeded with nonce $nc$. $S_{feistel}$ is an S-box layer defined as $S_{feistel}(\boldsymbol{x}) = S'(\boldsymbol{x}_L)||S'(\boldsymbol{x}_R)$. $S'$ is a Feistel structure over $\mathbb{Z}_p^t$ defined as

$$S'(\boldsymbol{y})_l = \begin{cases} y_l, & \text{if } l = 0 \\ y_l + y_{l-1}^2, & \text{otherwise} \end{cases}, \forall l \in \{0, 1, \dots, t-1\},$$

where $\boldsymbol{y} = y_0 || \cdots || y_{t-1} \in \mathbb{Z}_p^t$. $S_{cube}$ is another S-box layer defined as $S_{cube}(\boldsymbol{x}) = S(x_0)|| \cdots ||S(x_{n-1}) = x_0^3 || \cdots || x_{n-1}^3$, where $n = 2t$ is the length of the block. In [13], the designers provide two instances with 128 bit security: Pasta-3 with $(r, t) = (3, 128)$ and Pasta-4 with $(r, t) = (4, 32)$. For the characteristic $p$ of prime field, it requires $p > 2^{16}$ and $\gcd(p-1, 3) = 1$.

## D. Design Specification of Elisabeth

Elisabeth [14] is an HE-friendly stream cipher proposed at ASIACRYPT 2022. The design of Elisabeth extends FLIP family stream ciphers [4], [5] and follows the group filter permutator paradigm. Elisabeth-4 is an instance with 128 bits security level of the Elisabeth family. It is parameterized by a 1024-bit key and an IV of unspecified length. The overall structure of Elisabeth-4 is displayed in Fig. 3.

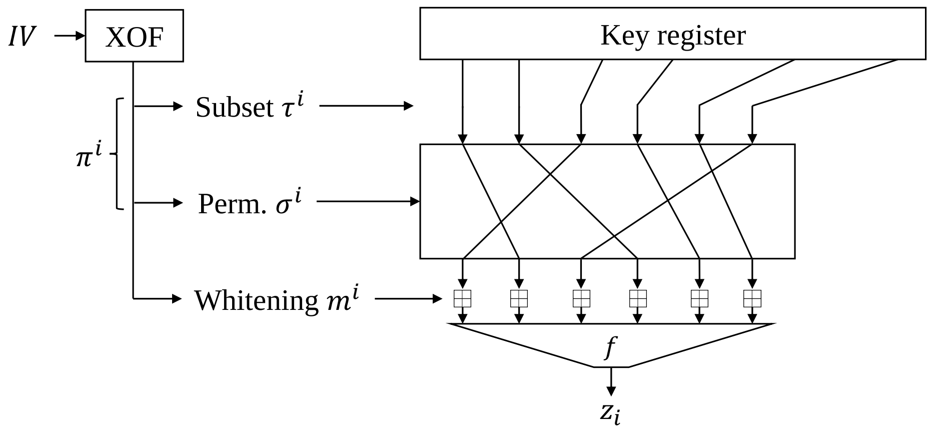

Fig. 3. Overall structure of Elisabeth-4.

Elisabeth-4 operates on elements over $\mathbb{Z}_{2^4}$. The key register can be viewed as an array of length $N = 256$, $\boldsymbol{k} = (k_1, \dots, k_{256}) \in \mathbb{Z}_{2^4}^{256}$. At each moment $i$, an ordered arrangement $\boldsymbol{\pi}^i = (\pi_1^i, \dots, \pi_{60}^i)$ of length $r \cdot t = 60$ would be selected by XOF. The arrangement can be seen as the composition of a selection of 60-subset $\boldsymbol{\tau}^i$ of $\{1, \dots, N\}$ and a permutation $\boldsymbol{\sigma}^i$ of its elements. Besides, the XOF also generates a whitening vector $\boldsymbol{m}^i = (m_1^i, \dots, m_{60}^i) \in \mathbb{Z}_{2^4}^{60}$. The keystream element $z_i \in \mathbb{Z}_{2^4}$ at moment $i$ is obtained by

$$z_i = f(k_{\pi_1^i} + m_1^i, \dots, k_{\pi_{60}^i} + m_{60}^i),$$

where $f$ is the filtering function.

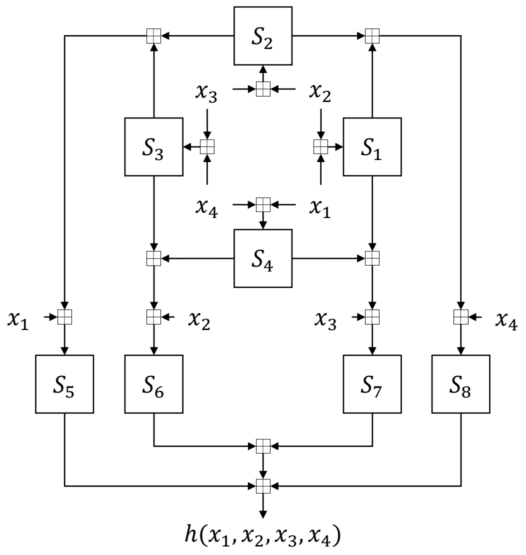

Fig. 4. The function $h$ of Elisabeth-4

The filtering function $f$ internally uses $t = 12$ parallel calls to a function $g$ applied on $r = 5$ elements. The $t$ outputs would be summed together to produce the output of $f$. Specifically, we have

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

The 5-to-1 function is constructed as the sum of a nonlinear 4-to-1 function $h$ and the remaining variable. That is,

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

The construction of function $h$ is shown in Fig. 4. All the eight look-up tables $S_1, \ldots, S_8$ over $\mathbb{Z}_{2^4}$ are selected at random by the designers. An instance of these Sboxes can be found in the Appendix A of [16].

## *E. Underlying Assumptions and Main Steps for DFA*

In DFA, faults are intentionally injected into the state of cipher to observe the distinctions between the normal and faulty keystreams. The underlying assumptions of this fault attack 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.

In the case of a bit-based fault, the value of the faulty state bit would simply flip. For a word-based fault, the value of the faulty state word would turn into a random value. Following the injection of faults, the attacker proceeds with the following steps to recover the secret key:

(1) Identify the location of the injected faults if possible. If the identification of the location of the fault is infeasible, then guess the location.
(2) Recover the state using information from both the normal and faulty keystreams. This process may involve constructing and solving equations or employing truth tables to iteratively filter.
(3) Derive the secret key from the obtained state.

## II. DFA ON MASTA

In this section, we will present our DFA on Masta. To mount the DFA on Masta, we need to inject a word-based fault into the internal state $\boldsymbol{s}_{r-1}$. As we cannot identify the location of the fault and the value of the faulty state word is unknown, we need to exhaustively try all $n$ words and all possible values in $\mathbb{Z}_p$. The process of our DFA on Masta is shown in Algorithm 1. Our DFA requires only one block of keystream. The line 6 is the most vital part of DFA. Therefore, we will give a detailed description of constructing the system of linear equations.

**Algorithm 1** DFA on Masta

```
1:  Collect a block of normal keystream $\boldsymbol{z}$ for an unknown key $\boldsymbol{k}$ on a nonce $nc$
2:  Inject a word-based fault at random position in the register of the internal state $\boldsymbol{s}_{r-1}$
3:  Collect a block of faulty keystream $\boldsymbol{z}'$ for the same key and nonce
4:  for each possible faulty position $i$ do
5:      for each possible value of the difference $\Delta s_{r-1,i} \in \mathbb{Z}_p$ do
6:          Construct $n$ linear equations over $\mathbb{Z}_p$ on the normal input $\boldsymbol{x}$ of the last $\chi$ function
7:          Solve the equations via Gaussian elimination
8:          $\boldsymbol{k}' = \boldsymbol{z} - \mathit{Affine}_r \circ \chi(\boldsymbol{x})$
9:          if $\mathsf{Masta}(\boldsymbol{k}') = \boldsymbol{z}$ then
10:             return $\boldsymbol{k}'$
11:         end if
12:     end for
13: end for
```

Given the fault location $i$ and the value of difference $\Delta s_{r-1,i}$, we have

$$s_{r-1,i} - s'_{r-1,i} = \Delta s_{r-1,i},$$
$$s_{r-1,j} - s'_{r-1,j} = 0,\ j = 0, \dots, n-1, j \neq i,$$

where $\boldsymbol{s}_r$ and $\boldsymbol{s}'_r$ denotes the normal and faulty internal states respectively. Moreover, according to the expression of Masta, we have the relation

$$\boldsymbol{z} = \boldsymbol{k} + \mathit{Affine}_r \circ \chi \circ \mathit{Affine}_{r-1}(\boldsymbol{s}_{r-1}).$$

The relation also holds for $\boldsymbol{z}'$ and $\boldsymbol{s}'_{r-1}$. Let $\boldsymbol{x}$ and $\boldsymbol{x}'$ represent the normal and faulty inputs, while $\boldsymbol{y}$ and $\boldsymbol{y}'$ denote the corresponding normal and faulty outputs of the nonlinear function $\chi$, respectively. We have the following equations:

$$\boldsymbol{x} = \mathit{Affine}_{r-1}(\boldsymbol{s}_{r-1}) = ARC_{r-1} \circ FMul_{r-1}(\boldsymbol{s}_{r-1}),$$
$$\boldsymbol{z} = \boldsymbol{k} + \mathit{Affine}_r(\boldsymbol{y}) = \boldsymbol{k} + ARC_r \circ FMul_r(\boldsymbol{y}).$$

Moreover, the relations about the differences are:

$$\Delta\boldsymbol{x} = FMul_{r-1}(\Delta\boldsymbol{s}_{r-1}),\ \Delta\boldsymbol{y} = FMul_r^{-1}(\Delta\boldsymbol{z}).$$

Because $\Delta\boldsymbol{s}_{r-1}, \Delta\boldsymbol{z}$ are known and $FMul_i$ are linear operations, the value of $\Delta\boldsymbol{x}$ and $\Delta\boldsymbol{y}$ are determined. For each component $i = 0, \dots, n-1$ of the $\chi$ function, we have the following equations:

$$y_i = x_i + x_{i+2} + x_{i+1}x_{i+2},$$
$$y'_i = x'_i + x'_{i+2} + x'_{i+1}x'_{i+2},$$
$$x'_i = x_i + \Delta x_i.$$

Furthermore, a linear equation regarding $\boldsymbol{x}$ over $\mathbb{Z}_p$ is derived:

$$\Delta y_i = \Delta x_i + \Delta x_{i+2} + x_{i+1}\Delta x_{i+2} + x_{i+2}\Delta x_{i+1}, \tag{1}$$

where $\Delta y_i, \Delta x_i, \Delta x_{i+1}$ and $\Delta x_{i+2}$ are all known. In total, we can obtain $n$ linear equations like Equation (1) with $n$ variables $x_0, \dots, x_{n-1}$. Next, we can solve the equations by Gaussian elimination. The complexity of Gaussian elimination on an $n \times n$ matrix is $\mathcal{O}(n^\omega)$. The straightforward way to perform Gaussian elimination will result in $\omega = 3$. In [26], $\omega$ is reduced to $\log_2 7 \approx 2.8$ with Strassen's divide-and-conquer method. In recent decades, many efficient algorithms have been proposed, and the upper bound of $\omega$ has been decreasing [27]–[29]. Recently, the upper bound on $\omega$ has been updated to 2.371552 [30]. However, these algorithms may not be easy to implement and have a large hidden constant. In this paper, we will use $\omega = 2.8$. The cost of checking candidate keys is negligible. Therefore, the time complexity of Algorithm 1 is $pn^{\omega+1} = pn^{3.8}$. For Masta, we have $p = 2^{16} + 1$, and then the time complexities of our DFA under different lengths of block $n$ are shown in Table III.

TABLE III
THE TIME COMPLEXITIES OF DFA ON MASTA UNDER DIFFERENT $n$

| $n$ | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 2048 |
|---|---|---|---|---|---|---|---|---|
| Cost | $2^{27.4}$ | $2^{31.2}$ | $2^{35.0}$ | $2^{38.8}$ | $2^{42.6}$ | $2^{46.4}$ | $2^{50.2}$ | $2^{57.8}$ |

By comparing Table II and Table III, it can be observed that our DFA is effective for all instances of Masta.

## III. DFA ON PASTA

This section presents our DFA on Pasta. To mount DFA on Pasta, we need to inject more than one word-based faults in the internal state $\boldsymbol{s}_{r-1}$ in total. For all the faults, we need to exhaustively try all possible positions and values. The process of our DFA on Pasta is displayed in Algorithm 2. Like the DFA on Masta, our DFA on Pasta also requires only one block of keystream. The equation construction is similar in both DFA, while the number of faults and equations solving are completely different. In the following context, a detailed description of constructing and solving equations will be given.

**Algorithm 2** Our DFA on Pasta

```
1:  Collect a block of normal keystream $\boldsymbol{z}$ for an unknown key $\boldsymbol{k}$ on a nonce $nc$
2:  Inject 3 word-based faults at random position in the register of the internal state $\boldsymbol{s}_{r-1}$
3:  Collect a block of faulty keystream $(\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{z}^{(3)})$ for the same key and nonce
4:  for each possible faulty position $(i_1, i_2, i_3)$ do
5:      for $(\Delta s^{(1)}_{r-1,i_1}, \Delta s^{(2)}_{r-1,i_2}, \Delta s^{(3)}_{r-1,i_3}) \in \mathbb{Z}_p^3$ do
6:          Construct $3t$ quadratic equations over $\mathbb{Z}_p$ on the normal input $\boldsymbol{x}$ of $S_{cube}$ using $(\boldsymbol{z}, \boldsymbol{z}^{(j)})$ and $\Delta s^{(j)}_{r-1,i_j}, j = 1, 2, 3$
7:          Construct another $t$ quadratic equations over $\mathbb{Z}_p$ on the normal input $\boldsymbol{x}$ of $S_{cube}$ using $(\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)})$ and $(\Delta s^{(1)}_{r-1,i_1}, \Delta s^{(2)}_{r-1,i_2})$
8:          Solve the $4t$ quadratic equations via linearization and Gaussian elimination
9:          Recover the candidate key $\boldsymbol{k}'$ using the inverse of $r-1$ round Pasta-$\pi$
10:         if Pasta-$\pi(\boldsymbol{k}') = \boldsymbol{z}$ then
11:             return $\boldsymbol{k}'$
12:         end if
13:     end for
14: end for
```

In particular, we will focus on how to derive $t$ quadratic equations for the first fault. Then the process will be repeated for the remaining faults. Suppose the fault location is $i_1$ and the value of difference is $\Delta s^{(1)}_{r-1,i_1}$, we have

$$\Delta s^{(1)}_{r-1,j} = \begin{cases} s^{(1)}_{r-1,j} - s_{r-1,j}, & j = i_1 \\ 0, & j = 0, 1, \ldots, n-1, j \neq i \end{cases}.$$

Let $\boldsymbol{x}, \boldsymbol{x}^{(1)}$ represent the normal and faulty inputs, while $\boldsymbol{y}, \boldsymbol{y}^{(1)}$ denotes the normal and faulty outputs of the $S_{cube}$, respectively. Then the difference $\Delta \boldsymbol{x}^{(1)}$ can be computed as

$$\Delta \boldsymbol{x}^{(1)} = \boldsymbol{x}^{(1)} - \boldsymbol{x} = A_{r-1}(\Delta \boldsymbol{s}^{(1)}_{r-1}).$$

Because $A_{r-1}$ is a linear operation, the value of $\Delta \boldsymbol{x}$ can be obtained based on $\Delta \boldsymbol{s}^{(1)}_{r-1}$. For each component $i = 0, \ldots, n-1$ of the $S_{cube}$, we have the following relation:

$$\Delta y_i = 3\Delta x^{(1)}_i x_i^2 + 3(\Delta x^{(1)}_i)^2 x_i + (\Delta x^{(1)}_i)^3. \tag{2}$$

Because the output of Pasta is truncated, we cannot obtain the value of $\Delta \boldsymbol{y}$ by inverting the final linear layer. However, Equation (2) still indicates that the output difference $\Delta y_i$ of the $S_{cube}$ is quadratic concerning $x_i$ and the only quadratic term is $x_i^2$. By propagating the difference forward, we have

$$\Delta \boldsymbol{z}^{(1)} = A_r(\Delta \boldsymbol{y})_L = 2M_{r,L}\Delta \boldsymbol{y}_L + M_{r,R}\Delta \boldsymbol{y}_R. \tag{3}$$

Therefore, according to Equation (2) and Equation (3), $t$ quadratic equations on $n$ variables $x_0, \ldots, x_{n-1}$ can be obtained. Specifically, there are exactly $n$ quadratic terms $x_0^2, \ldots, x_{n-1}^2$.

To solve the system of equations efficiently, we employ the linearization technique and Gaussian elimination. Considering every monomial appearing in the system as an independent variable, the system can be viewed as a linear system. As we mentioned above, the only quadratic term in Equation (2) is $x_i^2$. If we use the linearization technique, the number of independent variables is $2n$, i.e., $4t$ in total. To get a unique solution, we need at least $4t$ equations.

As mentioned above, we can obtain $t$ quadratic equations for each pair of normal and faulty keystream blocks. Besides, by using a pair of different faulty keystream blocks, we can also obtain $t$ quadratic equations. For example, for the first two faulty keystream blocks $(\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)})$, the input difference of $S_{cube}$ can be computed as

$$\Delta \boldsymbol{x}' = \boldsymbol{x}^{(2)} - \boldsymbol{x}^{(1)} = \Delta \boldsymbol{x}^{(2)} - \Delta \boldsymbol{x}^{(1)} = A_{r-1}(\boldsymbol{s}^{(2)}_{r-1} - \boldsymbol{s}^{(1)}_{r-1}).$$

Based on $\Delta \boldsymbol{x}'$ and $\boldsymbol{z}^{(2)} - \boldsymbol{z}^{(1)}$, $t$ new quadratic equations can be derived. Specifically, when the number of injected faults is $m$, we can acquire $\frac{m^2+m}{2}$ quadratic equations for Pasta in total. To collect more than $4t$ equations, we only need to inject 3 faults, instead of 4 faults.

For each guess of faults, we need to solve the equations and validate the solution. The complexity of Gaussian elimination is $(2n)^{2.8}$, and the time of recovering and verifying candidate keys is negligible. Therefore, the time complexity of Algorithm 2 is $(np)^3(2n)^{2.8} = 2^{8.6}p^3t^{5.8}$. In particular, the costs of our first DFA on Pasta-3 and Pasta-4 are $2^{37.6}p^3$ and $2^{49.2}p^3$ respectively. In other words, we can mount theoretical DFA on Pasta-3 and Pasta-4 when $p < 2^{26.2}$ and $p < 2^{30.1}$ with Algorithm 2, respectively. When $p \approx 2^{16}$, the costs of our DFA on Pasta-3 and Pasta-4 are $2^{97.2}$ and $2^{85.6}$.

## IV. DFA ON ELISABETH

In this section, we will describe how to mount the DFA on Elisabeth. We propose a total of two different DFAs and implement them on the only instance, Elisabeth-4. For the DFA on Elisabeth-4, we present a simple method to locate the faulty word. With our method, we no longer need to exhaustively test all possible word positions for the injected fault, greatly reducing the time complexity of the DFA. Due to the completely different structures of Elisabeth and Rasta-like ciphers, the DFA on Elisabeth also differs significantly from those on Masta and Pasta.

### *A. DFA on Elisabeth-4 using bit-based faults*

For our first DFA on Elisabeth-4, we need to inject 4 bit-based faults in the secret key register $\boldsymbol{k}$ on average. The process of our DFA on Elisabeth-4 is shown in Algorithm 3. In our key recovery process, we need to ensure that the faulty bit is the most significant bit (MSB) of the key word, hence we will first judge the position of the faulty bit and repeatedly inject faults until the faulty bit meets the condition. The probability that the faulty bit is the MSB of a word for Elisabeth-4 is $\frac{1}{4}$, so we need to inject 4 faults on average to complete our first DFA. We will describe our DFA in detail below.

**Algorithm 3** Our first DFA on Elisabeth-4
```
1:  Compute and store $T = \{(x_1, x_2, x_3, x_4, j, \Delta h)\}$ for all $(x_1, x_2, x_3, x_4) \in \mathbb{Z}_{24}^4$ and $j \in \{1, 2, 3, 4\}$, where $j$ is the word
    position of flipped bit
2:  Collect $L$ normal keystream words $\boldsymbol{z}$ for an unknown key $\boldsymbol{k}$ on an initial vector $IV$
3:  while correct key is not found do
4:      Inject a bit-based fault at random position in the the register of the secret key $\boldsymbol{k}$
5:      Collect $L$ faulty keystream words $\boldsymbol{z}'$ for the same $IV$ and compute $\Delta\boldsymbol{z} = \boldsymbol{z}' - \boldsymbol{z}$
6:      $tmp \leftarrow Identify(\Delta\boldsymbol{z}, IV, XOF)$
7:      if the faulty bit is the MSB of a key word then
8:          $Path \leftarrow GenPath(tmp, L, IV, XOF)$
9:          $S \leftarrow Filter(\Delta\boldsymbol{z}, Path, T, tmp, IV, XOF)$
10:         for $sol \in S$ do
11:             if Elisabeth($sol$, $IV$) = $\boldsymbol{z}$ then
12:                 return $sol$
13:             end if
14:         end for
15:     end if
16: end while
```

As described in Section I-E, the first step of DFA is to inject and identify the faults. For DFA on traditional stream ciphers, the signature-based identification technique [19] would be a good choice. The attacker will precompute the patterns, called signatures, for different faults in the offline phase and use it the identify the injected faults in the online phase. However, the structure of Elisabeth is completely different from the traditional stream ciphers like Kreyvium. There is no feedback function for Elisabeth and it updates the state according to the subset, permutation, and whitening vector produced by XOF. Therefore, if a single fault is injected into the key register of Elisabeth, it would affect only one word of state at a moment. Due to the unpredictable movement of faults in the state, we cannot apply the signature-based identification technique to Elisabeth. In [20], Méaux and Roy proposed an identification technique for FLIP-like ciphers using influence. In our work, we will employ a similar but more simple method to identify the position of the state word where the faulty bit is located. According to the workflow of Elisabeth-4, 60 out of 256 key words will be selected at each moment. If the injected fault is not included, the difference between normal and faulty keystream words at this moment will be zero. In other words, if the difference $\Delta z_i$ at moment $i$ is non-zero, then the fault must belong to the subset $\tau^i$. Based on this rule, we can filter the possible position of the injected fault by using non-zero difference iteratively. The details of our identification method are given in Algorithm 4.

We implemented the Algorithm 4 on our personal computer and tested it with 100,000 random IVs and faults. The experimental results show that, on average, only 22 keystream words are required and the cost of time is less than 0.01 seconds for each identification. Therefore, the cost of identification is negligible. With Algorithm 4, we can detect the word position of the injected fault.

**Algorithm 4** Identify the word where the faulty bit locates

```
1: procedure $Identify$ (The difference of keystream words $\Delta \boldsymbol{z}$, initial vector $IV$, $XOF$)
2:     $PosCan \leftarrow \{1, \ldots, 256\}$
3:     $i = 0$
4:     while $|PosCan| > 1$ do
5:         Generate the subset $\boldsymbol{\tau}^i \leftarrow XOF(IV, i)$
6:         if $\Delta z_i \neq 0$ then
7:             $PosCan \leftarrow PosCan \cap \boldsymbol{\tau}^i$
8:         end if
9:         $i \leftarrow i + 1$
10:    end while
11:    return the only element of $PosCan$
12: end procedure
```

The second step of DFA is to recover the state or key using the normal and faulty keystream words. A straightforward approach is to gather a sufficient number of equations and then proceed with solving them. We attempt to use the least significant bit (LSB) of $g$ as [16] and form the polynomial equations over $\mathbb{Z}_2$.[1] The resulting differential Boolean equation is complex and hard to solve. In other words, we cannot solve the system of equations in practical time. The intricate design of $g$ and the group $\mathbb{Z}_{2^4}$ make it impossible to construct a system of simple equations for Elisabeth-4 like FLIP-family ciphers.

Instead of constructing and solving equations, it is feasible to directly store all the solutions in a table, given the small input space of $h$. Between the input of function $g$ and key word, there is a whitening step, i.e., a whitening vector will be added to the state. Modular addition is a non-linear operation over $\mathbb{F}_{2^n}$, which means it will affect the XOR difference. Therefore, the table will become too large to store. In order to eliminate the impact of modular addition and simplify the stored table, we need to deeply study how the XOR difference propagates in modular addition and try to turn it to the difference over $\mathbb{Z}_{2^4}$ in the attack.

By observing the propagation of XOR difference in modular addition, we can get the following results:

TABLE IV
PROPAGATION OF XOR DIFFERENCE IN MODULAR ADDITION

| Input difference | Possible output difference |
|---|---|
| 0001 | 0001, 0011, 0111, 1111 |
| 0010 | 0010, 0110, 1110 |
| 0100 | 0100, 1100 |
| 1000 | 1000 |

From Table IV, we see that when the input XOR difference is 1000, the output XOR difference will remain unchanged. This is because flipping of the MSB of $a$ is equivalent to $a + 8$. Therefore, if the injected fault is located at the MSB, we can analyze the XOR difference as the difference over $\mathbb{Z}_{2^4}$ and the whitening step will not affect the difference. Without loss of generality, we suppose the injected bit-based fault locates at the first key word $k_1$:

$$k'_1 - k_1 = \Delta k = 8,\ k'_i = k_i, i = 2, \ldots, 256,$$

where $\boldsymbol{k}' = (k'_1, \ldots, k'_{256})$ is the faulty key. For a certain moment $j$, if $k_1$ does not belong to the subset $\boldsymbol{\tau}^j$, the output difference will be zero and this cannot help us recover the key. Therefore, we focus on the moments that $k_1$ is used to generate the keystream word. For a pair of normal and faulty keystream words $(z_j, z'_j)$, the difference can be represented as

$$\begin{aligned}\Delta z_j &= \sum_{i=0}^{t-1} \Delta g(x_{ir+1}, \ldots, x_{ir+5}) \\ &= \sum_{i=0}^{t-1} \Delta h(x_{ir+1}, \ldots, x_{ir+4}) + \Delta x_{ir+5}.\end{aligned}$$

According to the structure of Elisabeth-4, at most one state word will be affected at each moment. As a result, only one function $g$ will have an output difference. Suppose the faulty word is in $(x_1, \ldots, x_5)$, we have

$$\Delta z_j = \Delta g(x_1, \ldots, x_5) = \Delta h(x_1, \ldots, x_4) + \Delta x_5.$$

If the faulty word is $x_5$, $\Delta z_j$ is equal to $\Delta k$. This can help us to judge whether the faulty bit is the MSB or not. When the faulty bit is the MSB, the output difference will equal 8. Thus, after identifying the position of faulty word, we can judge the

[1]The LSB of the addition in $\mathbb{Z}_{2^4}$ behaves linearly in $\mathbb{Z}_2$.

type of the faulty bit by observing the output difference when the faulty word appears at $x_5$. This is what line 7 in Algorithm 3 does.

For key recovery, we utilize the moments when the faulty word lies at the input of nonlinear function $h$. In this case, the output difference of $h$ is equal to $\Delta z_j$. Because the whitening vector is known when initial vector $IV$ and moment $j$ are given, the value of $\boldsymbol{k}$ can be simply computed if $\boldsymbol{x}$ is known. Now, the problem has been converted to recovering the secret key with many $\Delta h$.

For each possible fault, we compute the output difference of $h$ and then store the result in table $T$ offline. In particular, the size of $T$ is $4 \times (2^4)^4 = 2^{18}$. This is because the input space of $h$ is $(2^4)^4$ and there are 4 possible faulty word positions. During the online phase, we only need to look up the table $T$ according to the position of fault $j$ and the actual value of difference $\Delta h$. Looking up a table is essentially still a form of equation solving, which is a trade-off between time and memory complexity. It is more advantageous than directly solving an equation when the equation is complex but has a few variables. On average, each equation about difference can reduce the value space of $(x_1, x_2, x_3, x_4)$ from $2^{16}$ to $2^{12}$. In other words, an equation can compress the space of the key to at most $2^{-4}$ of its original size. Given many equations, the correct key must satisfy all equations simultaneously. Assuming the candidate key sets derived from equations are $S_1, S_2, \ldots, S_n$, we need to continuously take the intersection until filtering out the unique correct key. This corresponds to lines 8-14 in Algorithm 3. It is obvious that the entire process includes $n-1$ intersection operations in total, and the cost of an intersection operation is closely related to the sizes of input sets. Since the positions of the keys in each candidate key set $S_i$ are not the same, the order of intersection will have a significant impact on the sizes of sets. For example, suppose $S_1$ and $S_2$ relate to keys $(k_{i_1}, k_{i_2}, k_{i_3}, k_{i_4})$ and $(k_{j_1}, k_{j_2}, k_{j_3}, k_{j_4})$ respectively, and $|S_1| = |S_2| = 2^{12}$. When $S_1$ and $S_2$ involve completely different key positions[2], the size of the set after taking the intersection will become $2^{24}$. However, if the related four key positions are the same, the size of the set after taking the intersection will shrink to around $2^8$. Specifically, we have the following proposition.

**Proposition 1.** *Let $S_1 \subseteq \mathbb{Z}_{2^4}^m$, $S_2 \subseteq \mathbb{Z}_{2^4}^n$ and $|S_1| = M, |S_2| = N$. If they have $t$ common related key positions, then the size of the set after taking the intersection is $\frac{MN}{2^{4t}}$.*

*Proof.* Suppose the corresponding key positions for $S_1$ and $S_2$ are $(k_{i_1}, \ldots, k_{i_m}), (k_{j_1}, \ldots, k_{j_m})$ respectively. Without loss of generality, suppose the first $t$ positions are the same, i.e., $k_{i_1} = k_{j_1}, \ldots, k_{i_t} = k_{j_t}$. Denote $Z = \{(\boldsymbol{s}_1, \boldsymbol{s}_2) | \boldsymbol{s}_1 \in S_1, \boldsymbol{s}_2 \in S_2\}$, then the size of Z is $MN$.

The probability of a collision over $\mathbb{Z}_{2^4}$ is $\frac{1}{2^4}$, and that of $t$ collisions is $\frac{1}{2^{4t}}$. Hence, the size of the set after taking the intersection is

$$|Z| \cdot Prob(t \text{ collisions}) = \frac{MN}{2^{4t}}.$$

This completes the proof. □

According to Proposition 1, the more common key positions there are, the smaller the size of the intersection will be. Therefore, to lower time complexity, we prefer to a set with the most common key positions with the current set each time we intersect. This leads to Algorithm 5, which is a greedy algorithm.

According to Proposition 1, we can infer that when $|S_2| = 2^{12}$ and the number of common key positions is 3, the size of the set after intersection will be $|S_1| \cdot 2^{12} \cdot 2^{-4\times 3} = |S_1|$. In other words, the size of the sets is very likely to remain unchanged after the intersection. Therefore, if we can ensure that each selected set has 3 or 4 common key positions with the existing set, then the size of our set can always be controlled within a reasonable range. That is the basis of our greedy algorithm and what is accomplished in lines 24-30 of Algorithm 5. Lines 13-23 of Algorithm 5 is the process of selecting a good starting point, which may help us reduce the runtime of DFA. If we can find a pair of sets with identical key positions, then the size of our initial set will be reduced to around $2^8$. Otherwise, it will be around $2^{12}$. By rough estimation, the solving time for the former is only $2^{-8}$ of the latter. After obtaining the path of intersection, we can proceed to recover the secret key with Algorithm 6. We implemented the first DFA on our personal computer. Given 15000 keystream words, the entire process took about 60 seconds.

## *B. DFA on Elisabeth-4 using word-based faults*

In the previous section, we introduced a DFA on Elisabeth-4 with single bit flip model. However, flip a single bit in an actual micro-controller may require strong attacker assumptions such as decapsulation or laser fault injection are required. A random word error fault model is more relaxed and easier to implement in a realistic environment. In this section, we will present our second DFA on Elisabeth-4 using word-based faults.

For our second DFA on Elisabeth-4, we need to inject a single word-based fault in the secret key register $\boldsymbol{k}$. The process of the second DFA is similar to that of the first one, which is described as Algorithm 7.

[2]This situation will never happen during our DFA. The number of common key positions will be at least 1, since the faulty key word must be included in each set.

**Algorithm 5** Find the (sub-)optimal merged path

```
1: procedure $GenPath$ (The position of faulty word $tmp$, the length of keystream $L$, initial vector $IV$, $XOF$)
2:     $TF \leftarrow []$
3:     $PA \leftarrow []$
4:     $RR \leftarrow []$
5:     $tS \leftarrow \emptyset$
6:     for $i = 0, \ldots, L-1$ do
7:         $\boldsymbol{\tau}^i, \boldsymbol{\pi}^i \leftarrow XOF(IV, i)$
8:         if $tmp \in \boldsymbol{\tau}^i$ and $tmp$ is a input of $h$ then
9:             $TF.add(i)$
10:            $PA.add($the input set of $h$ that includes $tmp)$
11:        end if
12:    end for
13:    if there exist $i$ and $j$ such that $PA[i] = PA[j]$ then
14:        $tS \leftarrow PA[i]$
15:        Add $TF[i]$ and $TF[j]$ to $RR$
16:        Remove $TF[i]$ and $TF[j]$ from $TF$
17:        Remove $PA[i]$ and $PA[j]$ from $PA$
18:    else
19:        $tS \leftarrow PA[0]$
20:        $RR.add(TF[0])$
21:        $TF.remove(TF[0])$
22:        $PA.remove(PA[0])$
23:    end if
24:    while $|tS| < 256$ and $TF \neq \emptyset$ do
25:        $new \leftarrow \max_i |PA[i] \cap tS|$
26:        $tS \leftarrow tS \cup PA[new]$
27:        $RR.add(TF[new])$
28:        $TF.remove(TF[new])$
29:        $PA.remove(PA[new])$
30:    end while
31:    $RR \leftarrow RR + TF$
32:    return $RR$
33: end procedure
```

**Algorithm 6** Filter the solution space with the output

```
1: procedure $Filter$ (The difference of keystream words $\Delta\boldsymbol{z}$, merged path $Path$, guessed bit position $pos$, filtering table $T$,
   initial vector $IV$, $XOF$)
2:     $t \leftarrow Path[0]$
3:     $\boldsymbol{\tau}^t, \boldsymbol{\pi}^t, \boldsymbol{m}^t \leftarrow XOF(IV, t)$
4:     Determine the exact position $ind$ of faulty bit and adding mask $\boldsymbol{w}$ with $tmp, \boldsymbol{\tau}^t, \boldsymbol{\pi}^t$ and $\boldsymbol{m}^t$
5:     $S \leftarrow \{\boldsymbol{x} - \boldsymbol{w} | (\boldsymbol{x}, ind, \Delta z_t) \in T\}$        ▷ $S \leftarrow \{\boldsymbol{x} - \boldsymbol{w} | (\boldsymbol{x}, \delta, ind, \Delta z_t) \in T\}$ for second DFA
6:     for $i = 1, \ldots, |Path| - 1$ do
7:         $t \leftarrow Path[i]$
8:         $\boldsymbol{\tau}^t, \boldsymbol{\pi}^t, \boldsymbol{m}^t \leftarrow XOF(IV, t)$
9:         Determine the exact position $ind$ of faulty bit and adding mask $w$ with $tmp, \boldsymbol{\tau}^t, \boldsymbol{\pi}^t$ and $\boldsymbol{m}^t$
10:        $H \leftarrow \{\boldsymbol{x} - \boldsymbol{w} | (\boldsymbol{x}, ind, \Delta z_t) \in T\}$        ▷ $H \leftarrow \{\boldsymbol{x} - \boldsymbol{w} | (\boldsymbol{x}, \delta, ind, \Delta z_t) \in T\}$ for second DFA
11:        $S \leftarrow Intersect(S, H)$        ▷ Intersect at the common indexes of $S$ and $H$
12:    end for
13:    return $S$
14: end procedure
```

**Algorithm 7** Our second DFA on Elisabeth-4

```
1: Compute and store $T' = \{(x_1, x_2, x_3, x_4, \delta, j, \Delta h)\}$ for all $(x_1, x_2, x_3, x_4) \in \mathbb{Z}_{2^4}^4$, $j \in \{1,2,3,4\}$ and $\delta \in \{1, 2, \dots, 15\}$,
   where $j$ is the position of faulty word and $\delta$ is the value of input difference
2: Collect $L$ normal keystream words $\boldsymbol{z}$ for an unknown key $\boldsymbol{k}$ on an initial vector $IV$
3: Inject a word-based fault at random position in the the register of the secret key $\boldsymbol{k}$
4: Collect $L$ faulty keystream words $\boldsymbol{z}'$ for the same $IV$ and compute $\Delta\boldsymbol{z} = \boldsymbol{z}' - \boldsymbol{z}$
5: $tmp \leftarrow Identify(\Delta\boldsymbol{z}, IV, XOF)$
6: $Path \leftarrow GenPath(tmp, L, IV, XOF)$
7: for $\delta$ from 1 to 15 do
8:     $S \leftarrow Filter(\Delta\boldsymbol{z}, Path, T', tmp, \delta, IV, XOF)$        ▷ Adjust the way to look up table
9:     for $sol \in S$ do
10:        if Elisabeth($sol$, $IV$) = $\boldsymbol{z}$ then
11:            return $sol$
12:        end if
13:    end for
14: end for
```

The first step of our second DFA is to compute and store a new table $T'$ for filtering in advance. Then, a word-based fault is injected and we can locate the position of the fault with Algorithm 4. Given the fault location $i$ and the value of difference $\Delta k_i$, we have

$$k_i' - k_i = \Delta k_i,\, k_j' = k_j, j \neq i,$$

As in the previous analysis, the output difference $\Delta z$ will be equal to the output difference of one $g(x_1, \dots, x_5)$. When the faulty word is $x_5$, the moment will be discarded immediately because the difference equation does not involve any key variable. Conversely, if the faulty word lies at one of $x_1, x_2, x_3, x_4$, we can use the table $T'$ to filter the candidate keys. In a random word error model, the value of faulty word is unknown. Therefore, we need to compute the difference for all possible faults. Since there are $4$ word positions and 15 non-zero difference, the size of table $T'$ is $4 \times 15 \times (2^4)^4 \approx 2^{22}$.

Next, we can utilize Algorithm 5 to find an effective path for filtering and filter the solution space with Algorithm 6. It is important to note that the index of the table $T'$ has changed; it now includes one additional column for the value of the difference compared to table $T$. Thus, we need to make some adjustments when looking up entries in the table. This change has little impact on the overall performance of the attack.

We also implemented the second DFA on our personal computer using a total of 15,000 keystream words. In our attack, we need to guess the exact value of difference and then filter the keys accordingly. When the guess is incorrect, the time spent on setting up and solving equations can be the same as, or even longer than, when the guess is correct. However, our filtering process terminates more quickly when the guess is wrong, as the set of candidate keys becomes empty after a few intersection operations. This feature highlights the advantage of our lookup table method for deriving the secret key. For each difference $\delta$, the filtering process took approximately 10 seconds. The entire process of our second DFA took about 150 seconds. Although the second DFA requires more time than the first, it demands fewer faults to inject and is easier to implement in practical applications. Thus, The attacker can select an appropriate attack based on practical conditions and complete the attack on Elisabeth-4.

## V. Conclusion

In our study, we introduce DFA against three recent HE-friendly stream ciphers: Masta, Pasta, and Elisabeth. Our results demonstrate that the secret keys of Masta and Elisabeth can be efficiently recovered within a practical time by introducing a random word-based fault into the state or key registers. Notably, the DFA of Elisabeth can also be completed with 4 bit-based faults. Furthermore, in the case of Pasta, injecting three random word-based faults enables the application of a theoretical DFA, showcasing a significant advantage over brute force attacks. With our DFA, the secret key of Elisabeth-4 can be recovered within several minutes. From our experimental results, it can be observed that complex round functions, truncated outputs, large finite fields, and long lengths of block all contribute to enhancing the resistance of cryptographic algorithms against DFA. Our comprehensive analysis reveals the vulnerabilities of HE-friendly stream ciphers to DFA, highlighting the need for closer scrutiny in the design of ciphers in this category.

## References

[1] R. L. Rivest, L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[2] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, "Ciphers for MPC and FHE," in *Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I 34*. Springer, 2015, pp. 430–454.

[3] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, and R. Sirdey, "Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression," Cryptology ePrint Archive, Paper 2015/113, 2015, https://eprint.iacr.org/2015/113. [Online]. Available: https://eprint.iacr.org/2015/113

[4] P. Méaux, A. Journault, F.-X. Standaert, and C. Carlet, "Towards stream ciphers for efficient FHE with low-noise ciphertexts," in *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I 35*. Springer, 2016, pp. 311–343.

[5] P. Méaux, C. Carlet, A. Journault, and F.-X. Standaert, "Improved filter permutators for efficient FHE: better instances and implementations," in *International Conference on Cryptology in India*. Springer, 2019, pp. 68–91.

[6] C. Dobraunig, M. Eichlseder, L. Grassi, V. Lallemand, G. Leander, E. List, F. Mendel, and C. Rechberger, "Rasta: a cipher with low ANDdepth and few ANDs per bit," in *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I 38*. Springer, 2018, pp. 662–692.

[7] F. Liu, S. Sarkar, W. Meier, and T. Isobe, "Algebraic attacks on Rasta and Dasta using low-degree equations," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2021, pp. 214–240.

[8] J. He, K. Hu, H. Lei, and M. Wang, "Massive superpoly recovery with a meet-in-the-middle framework: Improved cube attacks on Trivium and Kreyvium," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2024, pp. 368–397.

[9] C. Carlet, P. Méaux, and Y. Rotella, "Boolean functions with restricted input and their robustness; application to the FLIP cipher," *Cryptology ePrint Archive*, 2017.

[10] C. Carlet and P. Méaux, "A complete study of two classes of Boolean functions: direct sums of monomials and threshold functions," *IEEE Transactions on Information Theory*, vol. 68, no. 5, pp. 3404–3425, 2021.

[11] F. Liu, T. Isobe, and W. Meier, "Cryptanalysis of full LowMC and LowMC-M with algebraic techniques," in *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part III 41*. Springer, 2021, pp. 368–401.

[12] J. Ha, S. Kim, W. Choi, J. Lee, D. Moon, H. Yoon, and J. Cho, "Masta: an HE-friendly cipher using modular arithmetic," *IEEE Access*, vol. 8, pp. 194 741–194 751, 2020.

[13] C. Dobraunig, L. Grassi, L. Helminger, C. Rechberger, M. Schofnegger, and R. Walch, "Pasta: A case for Hybrid Homomorphic Encryption," in *TCHES 2023 Volume 2023/3*, 2023, 25th Conference on Cryptographic Hardware and Embedded Systems : CHES 2023, TCHES ; Conference date: 10-09-2023 Through 14-09-2023. [Online]. Available: https://ches.iacr.org/2023/program.php

[14] O. Cosseron, C. Hoffmann, P. Méaux, and F.-X. Standaert, "Towards case-optimized Hybrid Homomorphic Encryption: Featuring the Elisabeth stream cipher," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2022, pp. 32–67.

[15] L. Grassi, F. Liu, C. Rechberger, F. Schmid, R. Walch, and Q. Wang, "Minimize the randomness in Rasta-like designs: How far can we go?" *Cryptology ePrint Archive*, 2024.

[16] H. Gilbert, R. Heim Boissier, J. Jean, and J.-R. Reinhard, "Cryptanalysis of Elisabeth-4," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2023, pp. 256–284.

[17] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1997, pp. 37–51.

[18] J. J. Hoch and A. Shamir, "Fault analysis of stream ciphers," in *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*. Springer, 2004, pp. 240–253.

[19] S. Banik, S. Maitra, and S. Sarkar, "A differential fault attack on the Grain family of stream ciphers," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2012, pp. 122–139.

[20] P. Méaux and D. Roy, "Theoretical differential fault attacks on FLIP and FiLIP," *Cryptography and Communications*, pp. 1–24, 2024.

[21] D. Roy, B. Bathe, and S. Maitra, "Differential fault attack on Kreyvium & FLIP," *IEEE Transactions on Computers*, vol. 70, no. 12, pp. 2161–2167, 2020.

[22] R. Radheshwar, M. Kansal, P. Méaux, and D. Roy, "Differential fault attack on Rasta and FiLIP-dsm," *IEEE Transactions on Computers*, vol. 72, no. 8, pp. 2418–2425, 2023.

[23] L. Jiao, Y. Li, Y. Hao, and X. Gong, "Differential fault attacks on privacy protocols friendly symmetric-key primitives: RAIN and HERA," *IET Information Security*, vol. 2024, no. 1, p. 7457517, 2024.

[24] C. Dobraunig, D. Kales, C. Rechberger, M. Schofnegger, and G. Zaverucha, "Shorter signatures based on tailor-made minimalist symmetric-key crypto," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 843–857.

[25] J. Cho, J. Ha, S. Kim, B. Lee, J. Lee, J. Lee, D. Moon, and H. Yoon, "Transciphering framework for approximate homomorphic encryption," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2021, pp. 640–669.

[26] V. Strassen, "Gaussian elimination is not optimal," *Numerische mathematik*, vol. 13, no. 4, pp. 354–356, 1969.

[27] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, 1987, pp. 1–6.

[28] J. Alman and V. V. Williams, "A refined laser method and faster matrix multiplication," in *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2021, pp. 522–539.

[29] R. Duan, H. Wu, and R. Zhou, "Faster matrix multiplication via asymmetric hashing," in *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2023, pp. 2129–2138.

[30] V. V. Williams, Y. Xu, Z. Xu, and R. Zhou, "New bounds for matrix multiplication: from alpha to omega," in *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2024, pp. 3792–3835.