Differential Fault Attack on Rasta and FiLIP$_{\text{DSM}}$

R Radheshwar*[1], Meenakshi Kansal†[2], Pierrick Méaux‡[3], and Dibyendu Roy§[1]

[1]Indian Institute of Information Technology Vadodara, Gandhinagar, India
[2]Rashtriya Raksha University, Gandhinagar, India
[3]University of Luxembourg, Luxembourg

Abstract

In this paper we propose Differential Fault Attack (DFA) on two Fully Homomorphic Encryption (FHE) friendly stream ciphers Rasta and FiLIP$_{\text{DSM}}$. Design criteria of Rasta rely on affine layers and nonlinear layers, whereas FiLIP$_{\text{DSM}}$ relies on permutations and a nonlinear filter function. Here we show that the secret key of these two ciphers can be recovered by injecting only 1 bit fault in the initial state. Our DFA on full round (# rounds = 6) Rasta with 219 block size requires only one block (i.e., 219 bits) of normal and faulty keystream bits. In the case of our DFA on FiLIP-430 (one instance of FiLIP$_{\text{DSM}}$), we need 30000 normal and faulty keystream bits.

Keywords. Differential Fault Attack, Rasta, FiLIP.

1 Introduction

In recent time, it has been observed that symmetric key cipher with low multiplicative depth can provide a significant amount of communication efficiency for protocols using Fully Homomorphic Encryption (FHE). Due to which, developing adapted symmetric key ciphers, for example with low multiplicative depth, became one of the recent trend in research. For this purpose, in 2015 Albrecht et al. [3] presented the first design of block cipher with very low multiplicative depth. Consideration of FHE friendly stream ciphers started with the seminal work by Canteaut et al. [6]. They proposed the design of the first such cipher namely Kreyvium, inspired by the stream cipher Trivium based on nonlinear feedback shift registers. In this article we will study two recent FHE friendly stream ciphers, Rasta [11] and FiLIP$_{\text{DSM}}$ [20].

In 2018, Dobraunig et al. [11] introduced a new family of stream ciphers named Rasta. Rasta belongs to the kind of stream ciphers which have low multiplicative depth and suits in multiparty computation or FHE setup. The design of Rasta family differs significantly from the other stream ciphers, borrowing some ideas from block cipher designs. This family of stream ciphers is based on applying rounds combining an affine layer and a nonlinear layer, where the affine layers are pseudorandom and publicly derived independently of the key. For an r-round Rasta we apply an affine layer $(r + 1)$ times and a nonlinear layer r times on the secret key in an iterative way. The final output is further xor'ed with the secret key for generating a block of keystream bits.

*202273001@iiitvadodara.ac.in
†meenakshi.kansal@rru.ac.in
‡pierrick.meaux@uni.lu
§dibyendu.roy@iiitvadodara.ac.in

The design of Rasta has attracted a lot of attention, and since then various ciphers reusing parts of the design have been proposed, such as Masta, Dasta, Pasta, and Fasta. On the cryptanalysis side, recently in Asiacrypt 2021, Liu et al. [15] have proposed an algebraic attack on Rasta. They have used linearization technique and exploited the low degree expression of the inverse nonlinear layer of Rasta for their attack. It results in smaller complexities that the one considered by the authors for algebraic attacks by linearization. For most of the parameters studied it allows to attack one more round than the analysis given in [11]. For example, the instance of Rasta with block size 219, the linearization attack considered by the authors could attack only 2 rounds, instead of 3 with the recent attack, over the 6 rounds advocated for 80-bit security.

In 2019 Méaux et al. [20] introduced a new family of stream ciphers called $\text{FiLIP}_{\text{DSM}}$. The design of $\text{FiLIP}_{\text{DSM}}$ tracks back to the family of stream ciphers FLIP [21] which follows the filter permutator paradigm. This paradigm introduced in 2016 for FHE applications differ significantly from other stream cipher designs. It consists of a key register, a Pseudorandom Generator (PRG), and a filter which is a Boolean function f. Each keystream bit is obtained by applying f to a wire-cross permutation of the key, the permutation being publicly derived from the PRG. FiLIP is the stream cipher family coming from the improved filter permutator paradigm. The difference from the initial paradigm is the use of two different sizes of register and more objects publicly derived from the PRG. In FiLIP, the key size is N and the filter acts on $n < N$ variables. For each keystream bit a n-out-of-N subset of the key register is taken, permuted, and added to a pseudorandom binary vector of size n, before applying f. The subscript in $\text{FiLIP}_{\text{DSM}}$ refers to the family of functions used for the filter. Here DSM holds for Direct Sum of Monomials.

The design of FLIP and $\text{FiLIP}_{\text{DSM}}$ grasped a lot of attention in the crypto research community due to their uncommon design and performance for FHE. In the FHE context the first family $\text{FiLIP}_{\text{DSM}}$ has been used in [20] and [14], and lately another family $\text{FiLIP}_{\text{XOR-THR}}$ in [14]. Recently, the improved filter permutator paradigm has been generalized to groups rather than bits [10]. On the cryptanalysis side, a preliminary instance of FLIP [19] has been attacked by Duval et al. [12]. Thereafter multiple articles [8, 9, 16, 17, 23] have analysed the design of FLIP. The cryptographic parameters of the filter functions used in $\text{FiLIP}_{\text{DSM}}$ are analyzed in [7].

The Differential Fault Attack (DFA) on stream ciphers has been introduced by Hoch and Shamir [13]. DFA is a known plaintext attack technique where the attacker is allowed to inject a fault into the state of the cipher at one clock during the keystream generation phase of the cipher. Whereas in the case of classical differential attack [5] we assume that the attacker can introduce a difference in the public parameter of the cipher (for a stream cipher the public parameter is usually the initialization vector). It can be noticed that DFA is an attack model which is stronger than the classical differential attack as we are assuming that the attacker can introduce differences into the state of the cipher during the keystream generation phase. As the difference is introduced during the keystream generation phase of the cipher, the attacker can notice a distinguishable impact of the introduced difference in the generated keystream bits. In DFA the attacker first collects the required number of normal keystream bits corresponding to one unknown key. After that the attacker injects a fault at a random location of the state of the cipher and collects the required number of fault affected keystream bits. As the location of the injected fault is not known to the attacker, the attacker can perform statistical tests on the keystream bits to identify the location of the injected fault. One such statistical test is available in the literature, which is known as signature based fault identification. Interested readers can refer to [4, 18, 23, 24, 25] to understand the signature based fault identification technique in detail. For our work we have not followed any statistical technique for identifying the location of the injected fault as the required statistical data looks random in nature. If the location of the injected fault cannot be identified using any technique then the attacker will guess the location of the injected fault and performs the DFA. Guessing the

location of the injected fault will be practical if the number of required fault is minimal and the state size of the cipher is small.

In 2021, Roy et al. [23] have proposed DFA on two FHE friendly stream ciphers Kreyvium and FLIP. They have shown that the secret key of both Kreyvium and FLIP can be recovered respectively by injecting 3 and 1 bit fault in the state of these two ciphers in practical time. In the case of DFA on Kreyvium, Roy et al. used the signature based fault identification technique to identify the location of the injected fault. For the DFA on FLIP the identification of the location of the fault was not possible using the signature based fault identification technique. As the number of required fault was only one, the authors have guessed the location of the injected fault for performing a DFA on FLIP.

1.1 Our Contribution and Organization of the Article

In this article we consider two recent FHE friendly stream ciphers Rasta & $\text{FiLIP}_{\text{DSM}}$ and perform detailed security analysis relatively to DFA. Our major contributions in this article are as follows:

- We first propose a DFA on Rasta in Section 2. There we show that the secret key of Rasta can be recovered by injecting a single bit fault into the initial state of the cipher. Along with the generic technique of DFA on Rasta we provide an experimental verification of our attack on one instance of the Rasta family of ciphers. We show that a single bit fault reveals the secret key of Rasta with block size $n = 219$, $r = 6$ rounds in approximately 95 hours on a laptop with a processor of 2.80 GHZ clock and 16 GB RAM, assuming the fault location. For our experiment we need only one block of keystream bits i.e., 219 bits.

- In Section 3 of our article we propose a DFA on $\text{FiLIP}_{\text{DSM}}$. In the case of $\text{FiLIP}_{\text{DSM}}$ the secret key of the cipher can be recovered by injecting 1 bit fault. For an experimental attack we consider an instance of $\text{FiLIP}_{\text{DSM}}$ which provides 80-bit security. We consider an instance of $\text{FiLIP}_{\text{DSM}}$ where $N = 1792$, $n = 430$ and the nonlinear filter function is $m_F = [80, 40, 15, 15, 15, 15]$. Experimentally, assuming the fault location, we show that the secret key of the considered instance can be recovered on a laptop with a processor of 2.80 GHZ clock and 16 GB RAM in approximately 751 hours.

Table 1: Summary of Our Work

Cipher	Parameters	# Faults	# Keystreams	Time (in hrs)
Rasta	$n = 219$, $r = 6$	1	219 bits	95
$\text{FiLIP}_{\text{DSM}}$	$N = 1792$, $n = 430$	1	30000 bits	751

1.2 Design Specification of Rasta

Rasta [11] is a FHE friendly family of stream ciphers. Rasta is based on few important parameters namely block size n and block counter i. To generate an n block keystream bits corresponding to block counter i, the cipher first generates a pseudorandom permutation $\pi_{N,i}$ using nonce N and block counter i. The permutation $\pi_{N,i}$ takes secret key K (of size n) as input and produces a block of size n. This block is further xor'ed with K to generate keystream bits $K_{N,i}$ of size n. The

pictorial representation of the keystream block generation process is given in Figure 1. Each time a fresh nonce N and block counter i are used to generate a new block of keystream bits.

For an r round Rasta, the permutation $\pi_{N,i}$ is a composition of $(r+1)$ affine layers $A_{j,N,i}$ $(j = 0, \ldots, r)$ and r nonlinear layer S. Here one can note that each affine layer $A_{j,N,i}$ depends on the nonce N and block counter i. For an r round Rasta the mathematical form of $\pi_{N,i}$ will be as per the Equation (1).

$$\pi_{N,i} = A_{r,N,i} \circ S \circ A_{r-1,N,i} \circ S \circ \cdots \circ A_{1,N,i} \circ S \circ A_{0,N,i}. \tag{1}$$

The affine layer $A_{j,N,i}$ is a mapping from n bits to n bits and the algebraic form is as per Equation (2).

$$A_{j,N,i}(x) = M_{j,N,i} \cdot x + C_{j,N,i}. \tag{2}$$

Here $M_{j,N,i}$ is a pseudorandom $n \times n$ invertible binary matrix and $C_{j,N,i}$ is an $n \times 1$ pseudorandom binary matrix. These matrices are generated using an eXtendable-Output Function (XOF) which takes nonce N and block counter i as input. The nonlinear layer S is also a mapping from n bits to n bits whose algebraic form is described in Equation (3).

$$\begin{aligned} &S(x_0, \ldots, x_{n-1}) = (y_0, \ldots, y_{n-1}) \\ &y_i = x_i + x_{i+2 \pmod n} + x_{i+1 \pmod n} x_{i+2 \pmod n}; \\ &\text{for } i = 0, \ldots, n-1. \end{aligned} \tag{3}$$

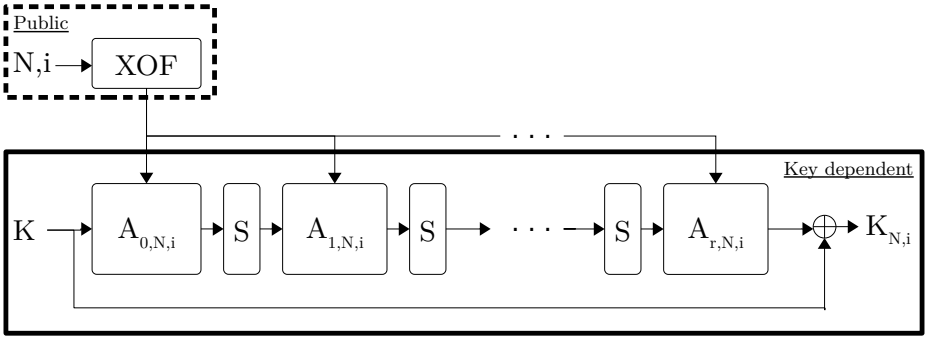

Figure 1: Design Specification of Rasta

1.2.1 Pseudorandom Invertible Matrix Generation Technique of Rasta

Here we describe the algorithmic sketch of the invertible pseudorandom binary matrix generation technique which is used in the main article of Rasta [11]. This algorithm (Algorithm 1) is proposed by Randall in 1993 [22]. Algorithm 1 starts with one $n \times 1$ matrix where all the entries are filled with 1 and two $n \times n$ zero matrices T, A. The algorithm returns an invertible binary square matrix $M = (m_{ij})_{n \times n}$. In this algorithm we have used one function $random(0, 1)$ which returns either 0 or 1 randomly. For our experiment we have used Algorithm 1 for generation of pseudorandom invertible binary matrices.

Liu et al. [15] have used a different algorithm for generating invertible matrix in their implementation [1] to attack Rasta. Their technique of invertible matrix generation is not efficient. The

[1] https://github.com/LFKOKAMI/AlgebraicAttackOnRasta.git

Algorithm 1: Random Invertible $n \times n$ Matrix Generation Algorithm [22]

```
index = (1)_{1×n};
T = A = (0)_{n×n};
for i = 0 to n − 1 do
    f = n − i ;
    while (f = n − i) do
        for f = 0 to n − i do
            if (random(0, 1) == 1) then
            end
        end
    end
    r = 0;
    c = 0;
    for r = 0 to n − 1 do
        if (index[r] == 1) then
            c = c + 1
        end
        if (f + 1 == c) then
            index[r] = 0 ;
            T[r][r] = 1 ;
            A[i][r] = 1 ;
        end
    end
    j = r + 1 ;
    while (j < n) do
        if (index[j] == 1) then
            T[j][r] = random(0, 1);
        end
        j = j + 1;
    end
    j = i + 1;
    while (j < n) do
        A[j][r] = random(0, 1);
        j = j + 1;
    end
end
M = A × T;
return M
```

authors have randomly filled 0 or 1 in an $n \times n$ matrix and if the determinant of the matrix is equal to 1 then their algorithm returns an invertible matrix. This process of generation of invertible matrix is not efficient, as every time we need to check the determinant of the matrix for deciding whether the matrix is invertible or not. Checking determinant every time is the most inefficient part of their algorithm. This is not the case for Algorithm 1. We have also experimentally verified that the matrix generation process of Liu et al. [15] takes much higher time than the Algorithm 1 for

higher values of n (e.g., for $n = 219$, Algorithm 1 took approx 2.1 seconds, where as Liu et al. [15] took 50.9 seconds). Due to this fact we have used Algorithm 1 for the generation of pseudorandom invertible matrices for our experiment.

1.3 Design Specification of FiLIP$_{\text{DSM}}$

FiLIP$_{\text{DSM}}$ [20] is a new family of stream ciphers which was proposed in Indocrypt 2019 which is adapted to FHE evaluation. The FiLIP$_{\text{DSM}}$ family of stream ciphers are designed via tweaking the design principle of FLIP family of stream ciphers [21]. FiLIP$_{\text{DSM}}$ is the first family of stream ciphers which is based on improved filter permutators. The design principle of FiLIP family of stream ciphers is simple and appealing. This family of ciphers suits in FHE setup because of their simple design and low noise growth.

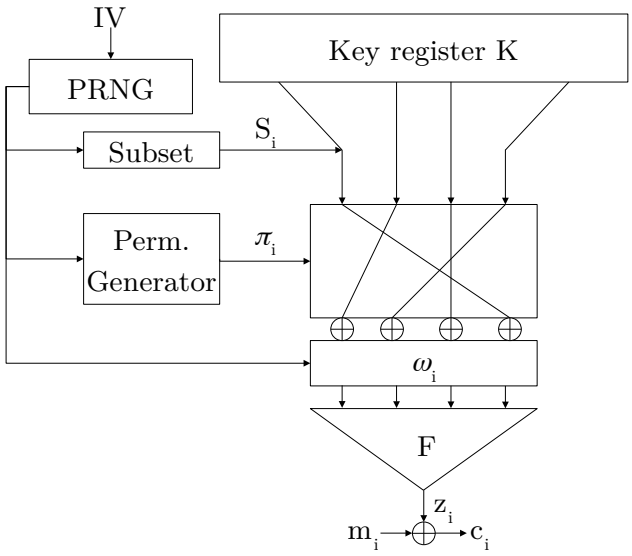

Figure 2: Design of FiLIP$_{\text{DSM}}$

FiLIP$_{\text{DSM}}$ [20] is based on one key register, one pseudorandom number generator (PRNG) and one nonlinear filter function. The key register takes the N bits secret key K as input. Here the weight of K must be $\frac{N}{2}$ (i.e., the number of 1's in K is $\frac{N}{2}$). Let the secret key K be denoted by $(k_1, \ldots, k_N)$. The PRNG takes the public parameter IV (initialization vector) as an input and it produces the followings at i-th clocking:

1. A subset S_i from $\{k_1, \ldots, k_N\}$ with $|S_i| = n$. Here n is fixed and $n \leq N$ for a particular instance of FiLIP$_{\text{DSM}}$.

2. One pseudorandom permutation π_i on $\{1, \ldots, n\}$.

3. One pseudorandom n bit string ω_i (called whitening).

In every clock (say i-th), firstly a subset S_i of $\{k_1, \ldots, k_N\}$ with $|S_i| = n$ is selected. After that one pseudorandom permutation π on $\{1, \ldots, n\}$ is generated by the PRNG. The permutation permutes the position of the elements of S_i and produces a new n bit tuple. This tuple is further bitwise xor'ed with ω_i to generate a new n bit string which is considered as an input to the nonlinear filter

function F. Upon getting the n bit input string the nonlinear filter function produces keystream bit z_i. The pictorial sketch of $\text{FiLIP}_{\text{DSM}}$ can be found in Figure 2.

The nonlinear filter function F is a d degree Boolean function involving n number of variables with zero as the constant term. The algebraic normal form of F is a direct sum of certain number of monomials (each monomial involves different set of variables) of different degree $\leq d$. Here F is represented as $m_F = [m_1, m_2, \ldots, m_d]$, where each m_i denotes the number of monomials of degree i. It means $n = \sum_{i=1}^{d} i m_i$. For example if F is $[1, 1, 2]$ then $F = x_1 + x_2x_3 + x_4x_5x_6 + x_7x_8x_9$. For different instances of N, n, and m_F different models of $\text{FiLIP}_{\text{DSM}}$ can be created (see [20] for such instances). One such instance is $N = 1792$, $n = 430$, $m_F = [80, 40, 15, 15, 15, 15]$ which provides 80 bit security. In this paper, we consider the same instance and show that $\text{FiLIP}_{\text{DSM}}$ is prone to DFA.

1.4 Underlying Assumptions for DFA

Here we provide the underlying assumptions of our DFA. This attack model relies on the following assumptions.

1. The attacker can restart the cipher with the same key and the same other public parameters as input.
2. The attacker can inject fault at a specific moment during the keystream generation of the cipher.
3. The attacker has the required tool (such as laser shots, clock glitches etc.) for the injection of fault.
4. The number of fault injections has to be minimal.

After the injection of the fault the attacker works as follows to recover the secret key.

1. If possible identify the location of the injected fault. If the identification of the location of the fault is not possible then guess the location.
2. Construct a system of equations involving the state bits as unknown from the normal and faulty keystream bits.
3. Solve the system to find the unknown state bits.
4. Find the secret key from the state bits.

2 DFA on Rasta

This section describes our DFA on Rasta [11]. Rasta is based on r rounds. Each round is composed of two layers, one is an affine layer and the other one is a nonlinear layer. The affine layer is controlled by the XOF function. The XOF function takes IV as input to generate $r + 1$ random looking binary matrices and binary vectors. The pseudorandom binary matrices have size $n \times n$ and they are invertible. The pseudorandom binary vectors have length n, where n denotes the block size of Rasta. The secret key of length n-bit is considered as input to the first affine layer then the output is further processed into the nonlinear layer. The pictorial representation of this process is described in Figure 1.

For mounting the DFA on Rasta the attacker injects a single bit fault in the key K. The injected location of the fault is unknown to the attacker. From the design specification of Rasta one can observe that many pseudorandom binary matrices are involved in the keystream bits generation process. Due to which the fault propagates through the different layers to the keystream in a random looking manner. This process of fault propagation cannot be controlled by any attacker as these matrices are generated using a XOF. Due to this we cannot identify the location of the injected fault by computing the signature. The signature becomes random looking in nature. The same difficulty occurred in Roy et al.'s DFA on FLIP [23]. To overcome this problem, the authors have exhaustively tried for all possible fault location and mounted a successful DFA on FLIP.

In the case of Rasta, we considered a similar approach as a first experiment. First we injected one bit fault and generated a system of equations for each possible fault locations. For our initial experiments we have considered Rasta with block size $n = 219$, round $r = 6$ which provides 80-bit security (for more detail see [11]). We injected a single bit fault (say at the first bit of K), then collected the normal and faulty keystream bits z and $\widetilde{z}$ for one block. Using these keystream bits we generated normal keystream bits equations and faulty keystream bits equations in SageMath 9.1 [2]. These equations were further fed to a SAT solver to find a possible solution for K. In this first experiment the SAT solver did not return any solution in feasible amount of time. Thus, we looked for a different technique for mounting a DFA on Rasta which we elaborate shortly.

We first see how the injected fault will propagate into the state of the cipher after the injection of the fault. We assume the fault is at one bit, say in the first bit of the secret key K. Corresponding to this difference, we define $\Delta_0 = (1, 0, \ldots, 0)$ i.e., 1 is at the first location and 0 at the other locations. Let $\widetilde{K}$ denotes the fault affected key i.e., $\widetilde{K} = K + \Delta_0$. After the application of the first affine layer the expression of the difference ($\Delta_1^{(1)}$) between normal and faulty state will be

$$\begin{aligned}
\Delta_1^{(1)} &= A_{0,N,i}(K) + A_{0,N,i}(\widetilde{K}) \\
&= (M_{0,N,i} \cdot K + C_{0,N,i}) + (M_{0,N,i} \cdot \widetilde{K} + C_{0,N,i}) \\
&= M_{0,N,i} \cdot K + M_{0,N,i} \cdot \widetilde{K} \\
&= M_{0,N,i} \cdot (K + \widetilde{K}) \\
&= M_{0,N,i} \cdot \Delta_0. \qquad (4)
\end{aligned}$$

After the application of an affine layer the output will enter into the nonlinear layer S. The $\Delta_1^{(1)} = (\delta_1^{(1)}, \ldots, \delta_n^{(1)})$ is the input difference for the nonlinear layer S. Let $X_1 = (x_1^{(1)}, \ldots, x_n^{(1)})$ and $\widetilde{X_1}$ be the normal and fault affected states respectively. Here $\widetilde{X_1} = (\widetilde{x_1}^{(1)}, \ldots, \widetilde{x_n}^{(1)})$ with $\widetilde{x}_i^{(1)} = (x_i^{(1)} + \delta_i^{(1)})$ for $i = 1, \ldots, n$. If $\Delta_1^{(2)} = (\delta_1^{(2)}, \ldots, \delta_n^{(2)})$ denotes the output difference after the application of the nonlinear layer then the expression of each $\delta_i^{(2)}$ will be as follows.

$$\begin{aligned}
\delta_i^{(2)} &= S(x_i) + S(x_i + \delta_i^{(1)}) \\
&= (x_i^{(1)} + x_{(i+2) \pmod n}^{(1)} + x_{(i+1) \pmod n}^{(1)} x_{(i+2) \pmod n}^{(1)}) \\
&\quad + \big(x_i^{(1)} + \delta_i^{(1)} + x_{(i+2) \pmod n}^{(1)} + \delta_{(i+2) \pmod n}^{(1)} \\
&\quad + (x_{(i+1) \pmod n}^{(1)} + \delta_{(i+1) \pmod n}^{(1)}) \\
&\qquad (x_{(i+2) \pmod n}^{(1)} + \delta_{(i+2) \pmod n}^{(1)})\big) \\
&= \delta_i^{(1)} + \delta_{(i+2) \pmod n}^{(1)} + x_{(i+1) \pmod n}^{(1)} \delta_{(i+2) \pmod n}^{(1)} \\
&\quad + x_{(i+2) \pmod n}^{(1)} \delta_{(i+1) \pmod n}^{(1)} \qquad (5) \\
&\quad + \delta_{(i+1) \pmod n}^{(1)} \delta_{(i+2) \pmod n}^{(1)}.
\end{aligned}$$

Using the mathematical formulae derived in Equations (4) and (5) we can determine the state difference expression after any number of iterations of affine and nonlinear layers of Rasta with the knowledge of Δ_0 even though we do not know the secret key. That means if $\Delta_l^{(2)}$ denotes the state difference after l number of iterations of affine and nonlinear layers then after one more round the difference expression will be $\Delta_{l+1}^{(1)} = M_{l,N,i} \cdot \Delta_l^{(2)}$ and $\Delta_{l+1}^{(2)} = (\delta_1^{(l+1)}, \ldots, \delta_n^{(l+1)})$ where the expression of $\delta_i^{(l+1)}$ will be as per Equation (7).

$$\Delta_{l+1}^{(1)} = M_{l,N,i} \cdot \Delta_l^{(2)} \tag{6}$$

$$\Delta_{l+1}^{(2)} = (\delta_1^{(l+1)}, \ldots, \delta_n^{(l+1)}) \tag{7}$$

$$\begin{aligned}\delta_i^{(l+1)} = {} & \delta_i^{(l)} + \delta_{(i+2) \pmod n}^{(l)} \\ & + x_{(i+1) \pmod n}^{(l)} \delta_{(i+2) \pmod n}^{(l)} \\ & + x_{(i+2) \pmod n}^{(l)} \delta_{(i+1) \pmod n}^{(l)} \\ & + \delta_{(i+1) \pmod n}^{(l)} \delta_{(i+2) \pmod n}^{(l)}.\end{aligned}$$

Here $x_{(i+1)(\mod n)}^{(l)}$ denotes the normal state bit variable of the $(i+1)$-th location after the application of the affine layer in the $(l+1)$-th round. We will be using these mathematical formulae for our experiments.

We are now in a position for describing our DFA on Rasta. As per the standard rule of DFA we first collect the normal keystream bits $z_i; i = 0, \ldots, n-1$, for an unknown key K. After collecting the normal keystream bits we inject one bit fault at a random location of the key register of Rasta. Here the location of the injected fault is unknown to us. After injecting the fault we collect one block of the fault affected keystream bits $\widetilde{z}_i; i = 0, \ldots, n-1$. Using these keystream bits we move towards the offline phase of our attack. We first construct equations corresponding to the normal keystream bits where the secret key corresponds to the n unknown binary variables. We use Algorithm 2 for the construction of the equations corresponding to normal keystream bits. In Algorithm 2 we have considered n block keystream bits which are known to us and the n bit unknown variable which compose the secret key K. During the process of keystream generation, in each round there is an application of a nonlinear layer (S) just after the application of a linear layer where each output bit is a 2 degree function of the input bits (see Equation (3)). Thus, the degree of each state bit will increase rapidly. To tackle this we introduce new variables $y_l^{(j+1)}$ corresponding to each degree-two expression and we incorporate new equations of the form $y_l^{(j+1)}+$ the degree-two expression in the set of equation Sys. This optimization helps us to generate the keystream bit equations efficiently.

Now we move forward to generate equations corresponding to the difference between normal and faulty keystream bits. To construct such equations we consider the difference propagation equations corresponding to $\Delta_{l+1}^{(1)}$ and $\Delta_{l+1}^{(2)}$ derived in Equations (6) and (7). We describe this process of difference equations generation in Algorithm 3. We start the equation generation process with the information of normal and faulty keystream bits. As we have already mentioned that the location of the injected fault is not known to us, we generate a system of equations for each possible fault location and recover the correct key via solving each system. In Algorithm 3 we assume that the fault is injected at the i-th location of the key register (i can be any integer from 1 to n). We consider the initial difference $\Delta_0 = (0, \ldots, 1, \ldots, 0)$, where 1 is at the i-th position and other locations are having 0. If the initial state is K and fault affected state is $\widetilde{K}$ then $\widetilde{K} = K + \Delta_0$. From the Equations (6) and (7) we already noticed the form of the state differences after application of the affine layer and nonlinear layer. In Algorithm 3 we introduce new variables ($\delta y_l^{(j+1)}$) corresponding

Algorithm 2: Normal Keystream Bits Equations Generation for Rasta with r Rounds

1 Normal Keystream Bits $z = (z_1, \ldots, z_n)$;
2 Unknown $K = (k_1, \ldots, k_n)$;
3 $X_0 = K$;
4 $Sys = \{\}$;
5 **for** $j = 0$ *to* $r - 1$ **do**
6 $\quad X_{j+1} = A_{j,N,i}(X_j)$;
$\quad$ `/* The invertible matrix is generated using Algorithm 1. */`
7 $\quad$ **for** $l = 1$ *to* n **do**
8 $\quad\quad Eq : y_l^{(j+1)} + x_l^{(j+1)} + x_{(l+2) \pmod n}^{(j+1)} + x_{(l+1) \pmod n}^{(j+1)} x_{(l+2) \pmod n}^{(j+1)} = 0$;
$\quad\quad$ `/* Here` $x_i^{(j+1)}$ `is the` i`-th element of` X_{j+1}`. */`
9 $\quad\quad Sys = Sys \cup \{Eq\}$;
10 $\quad$ **end**
11 $\quad X_{j+1} = (y_1^{(j+1)}, \ldots, y_n^{(j+1)})$;
12 **end**
13 $X_{r+1} = A_{r,N,i}(X_r)$;
14 $Z = X_{r+1} + K + z$;
15 $Sys = Sys \cup \{Z\}$;
16 **return** Sys

to each nonlinear layer computation on the difference. Related to each $\delta y_l^{(j+1)}$ we incorporate a new equation as described in line 10 of Algorithm 3 (this technique is similar to the one used in Algorithm 2). The introduction of new variable corresponding to each nonlinear layer helps significantly in finding the expression of difference between normal and faulty keystream bits in terms of the secret key K.

After generating the system of equations (Sys and Sys_d) using Algorithm 2 and Algorithm 3, we pass these two systems to the SAT solver for finding the solution corresponding to the secret key K. If the SAT solver returns a solution corresponding to the considered fault location i then we put the solution in the set of possible keys. Using each possible key we generate a few keystream bits and match with the previously collected keystream bits corresponding to the unknown key. In this way we can filter out the correct key from the list of possible keys. We simulate this process on one instance of Rasta with 219 block size and 6 rounds, the experimental result is described below.

DFA on Rasta ($n = 219$, $r = 6$, 80 bit security). We have simulated our DFA technique on Rasta with block size $n = 219$ and it has $r = 6$ rounds. As per the designer's claim this instance provides 80-bit security. We first collect one block of keystream bits (i.e., 219 bits) z. Further we inject a single bit fault at a random location of the key register. After that we collect one block of faulty keystream bits $\widetilde{z}$. Further we generate equations involving the unknown secret key K for the normal keystream bits using Algorithm 2. We use Algorithm 3 for the generation of equations involving secret key for the difference between normal and faulty keystream bits. We have used SageMath 9.1 [2] software for the generation of these equations. After the generation of both these systems of equations we have used SAT solver for finding the solution for the key. The SAT solver returns a unique solution for the correct fault location in approximately 1555 seconds. The complete experiment has been performed in a laptop with a processor of 2.80GHz clock, 16 GB

Algorithm 3: Keystream Bits Difference Equations Generation for Rasta with r Rounds

Normal Keystream Bits $z = (z_1, \ldots, z_n)$;
Faulty Keystream Bits $\widetilde{z} = (\widetilde{z_1}, \ldots, \widetilde{z_n})$;
Unknown $K = (k_1, \ldots, k_n)$;
$\Delta_0 = (0, \ldots, 1, \ldots, 0)$, i-th bit is 1 and rest are 0;
$\widetilde{K} = K + \Delta_0$;
$X_0 = K$;
$\Delta_0^{(1)} = \Delta_0$;
for $j = 0$ *to* $r - 1$ **do**
    $\Delta_{j+1}^{(1)} = M_{j,N,i} \cdot \Delta_j^{(1)}$;
    /* Here $M_{j,N,i}$ is the same matrix which is used in line 6 of Algorithm 2. */
    for $l = 1$ *to* n **do**
        $Eq : \delta y_l^{(j+1)} + \delta_i^{(l)} + \delta_{(i+2) \pmod n}^{(l)} + x_{(i+1) \pmod n}^{(l)} \delta_{(i+2) \pmod n}^{(l)} + x_{(i+2) \pmod n}^{(l)} \delta_{(i+1) \pmod n}^{(l)} + \delta_{(i+1) \pmod n}^{(l)} \delta_{(i+2) \pmod n}^{(l)} = 0$;
        /* Here $x_m^{(l)}$ is the normal state bit as per the Algorithm 2 and $\delta_i^{(l)}$ is the i-th bit of $\Delta_{j+1}^{(1)}$. */
        $Sys_d = Sys_d \cup \{Eq\}$;
    end
    $\Delta_{j+1}^{(2)} = (\delta y_1^{(j+1)}, \ldots, \delta y_n^{(j+1)})$;
end
$\Delta_{r+1}^{(1)} = M_{r,N,i}(\Delta_r^{(2)})$;
$Eq : \Delta_{r+1}^{(1)} + \Delta_0 + z + z^f = 0$;
$Sys_d = Sys_d \cup \{Eq\}$;
return Sys_d

RAM and Linux (Ubuntu 22.04) environment. As the location of the injected fault is not known to us, we need to simulate this process for all 219 possible locations of fault. Since checking the consistency of the keystream generated by a candidate key requires negligible time relatively to the SAT solver. The entire attack on Rasta will take approximately 219×1555 seconds ≈ 95 hours in our specified laptop.

3 DFA on FiLIP$_{\text{DSM}}$

In this section we demonstrate our DFA on FiLIP$_{\text{DSM}}$ [20]. From the design specification of FiLIP$_{\text{DSM}}$ it can be noticed that this cipher is quite different from the other standardized stream ciphers [1]. The state of the cipher is updated via random selection of n bits from N bits, pseudorandom wire-cross permutation and addition of a pseudorandom n-bit length vector. The keystream bit is generated using these n length pseudorandom bit string. All these design rationals make FiLIP$_{\text{DSM}}$ different from other feedback oriented stream ciphers.

Usually, the first step of DFA has two stages on (feedback oriented) stream ciphers. The first stage is the injection of a fault in the state of the cipher, and then the identification of the location of the injected fault. If we inject a fault in the state of a feedback oriented stream cipher then that

injected fault starts affecting the other state bits via the feedback as the feedback bit is computed using a deterministic function. This helps the attacker to get a pattern in the keystream bits (for most of the ciphers) which is thereafter used to identify the location of the injected fault. For $\text{FiLIP}_{\text{DSM}}$, the state of the cipher is updated via random selection of n out of N key bits and further these n bits are permuted using a pseudorandom permutation. Thus, we obtain an unpredictable movement of the fault in the state. Due to which we do not get any pattern in the keystream bits after injecting fault in the state of the cipher. Therefore finding the location of the injected fault is difficult for $\text{FiLIP}_{\text{DSM}}$ even if the number of faults remains constant for all the clocks. Moreover, there is another issue in $\text{FiLIP}_{\text{DSM}}$, the fault affected bit may not be involved in every keystream bit computation, since n many bits are selected randomly from the N length key register for the keystream bit computation. Now to tackle these issues we follow a different technique which we discuss in the next paragraph.

In the first step of our DFA on $\text{FiLIP}_{\text{DSM}}$ we inject a single bit fault in the key register of length N. As the detection strategy for the location of the fault does not apply for $\text{FiLIP}_{\text{DSM}}$, we directly go for the equation generation part of the DFA. Here we generate a system of equations for all possible (i.e., N) locations of the fault. For each system we will apply the same procedure, solving the system of equations for recovering the key $K = (k_1, \ldots, k_N)$ of weight $\frac{N}{2}$. For an unknown key K we will first collect l many normal keystream bits. These bits are denoted by $z_1, \ldots, z_l$. After the injection of a single bit fault (location is not known) we collect l many faulty keystream bits. These faulty keystream bits are denoted by $\widetilde{z_1}, \ldots, \widetilde{z_l}$. For every possible fault location r ($r = 1, \ldots, N$) we need to prepare the corresponding system of equations, which will be done offline. For simplicity of our discussion, we consider $r = 1$ i.e., we discuss the scenario for k_1 (fault is at first bit of the key register) and the same will be followed for all other values of r. As the secret key K is loaded in the key register and the state of the register is not updated, the fault will remain fixed at k_1 only. Among the N bits of the key register, n bits are selected randomly in every clock. These n bits will be involved in the keystream bit computation for this clock. If the faulty bit k_1 is a part of the selected bits then it will affect the keystream bit computation of that clock, otherwise the corresponding keystream bit will be fault free. Thereafter, the question is with what probability the bit k_1 will be selected from $K = (k_1, \ldots, k_N)$ during the selection of n out of N bits. Here we detail the expression of that probability. We can select n bits from N bits in $\binom{N}{n}$ number of ways. Among these $\binom{N}{n}$ many selections there will be $\binom{N-1}{n-1}$ number of possibilities where k_1 will be present. Thus, the probability that the selected n bits from the key register contains k_1 is $p = \frac{\binom{N-1}{n-1}}{\binom{N}{n}} = \frac{n}{N}$. We know that when the faulty bit k_1 is present in the selected subset of n bits from the key register then the keystream bit is affected by the injected fault, otherwise the keystream bit is fault free which does not help us in mounting the DFA on $\text{FiLIP}_{\text{DSM}}$. As the selection of subsets of bits from the key register is done using the public parameter IV, we take those clockings where the considered fault location bit (here k_1) is selected for the keystream bit computation.

The design specification of $\text{FiLIP}_{\text{DSM}}$ says that the nonlinear filter function F is a direct sum of certain monomials involving different number of variables. If the nonlinear filter function F has the representation $m_F = [m_1, \ldots, m_d]$ then F will have m_1 number of linear monomials, m_2 number of degree 2 monomials, m_3 number of degree 3 monomials and so on till degree d, with each variable appearing in only one monomial. In the generic sense F has m_i number of monomials of degree i. In the following we assume that the faulty bit is selected for the keystream bit computation for t-th clock. Let z_t and $\widetilde{z_t}$ denote the normal and faulty keystream bits respectively. If the faulty bit is involved in a linear monomial in the keystream bit computation then $z_t + \widetilde{z_t} = 1$. These kind of equations do not involve any key variable thus they are discarded immediately. If the faulty bit is involved in any degree two term of the keystream bit then $z_t + \widetilde{z_t}$ will be a single monomial

of degree 1. More generally, if the faulty bit is involved in any degree i term of the keystream bit computation then $z_t + \widetilde{z_t}$ will be a single monomial of degree $(i-1)$. After constructing these equations using l many normal and faulty keystream bits we use a SAT solver to find a solution for K. If the SAT solver returns a solution whose weight is $\frac{N}{2}$ then we consider that solution as a possible key and put in a set I. We generate a few keystream bits using all possible key K_i from the set I and match it with the known keystream bits for the unknown key K. If there is a match then K_i is the correct key. As the location of the fault is not known to us, we repeat the same process for all possible values of $r \in \{1, \ldots, N\}$ until we recover the unknown key K. If we assume that the SAT solver takes in average T seconds to find a solution, then the total time for recovering the secret key is $\approx \frac{NT}{p}$, where p is the probability that the faulty bit is selected in the keystream bit computation.

DFA on FiLIP-430 **($N = 1792, n = 430$, 80-bit security).** We consider an instance of $\text{FiLIP}_{\text{DSM}}$, i.e., FiLIP-430 to experimentally verify our DFA. At first we inject a single bit fault at the key register of length 1792. Since the fault detection technique does not apply, we try all possible fault locations for our attack. Thus without loss of generality we can assume that fault is injected at the first bit (k_1) of the key register. According to the defined parameters of FiLIP-430, a subset of bits with cardinal 430 will be selected randomly. These 430 bits are going to be involved in the keystream bit computation. As this selection is probabilistic, the probability that k_1 will be selected for keystream bit computation is $\frac{\binom{1791}{429}}{\binom{1792}{430}} = \frac{430}{1792} \approx 0.24$. As these selection of subsets of size 430 is done using IV, for our attack we are going to consider only those cases where the faulty bit is selected for the keystream bit computation. With these considerations we collect normal (z) and faulty keystream bits ($\widetilde{z}$) for the unknown key K. Now we compute the corresponding normal and faulty keystream bit expressions where the secret key bits are the unknown variables. These expressions are further xor'ed (say the xor'ed expression is E) to create an equation of the form $E = z + \widetilde{z}$. After forming the system of equations involving the unknown variables, we pass the entire system to the SAT solver for finding a solution. By doing the experiment we observed that for small number of equations we get multiple solutions for the system. A system with average 30000 equations provides a unique solution which is also valid i.e., the SAT solver returns a solution whose Hamming weight is 896. In practice, checking the Hamming weight of a candidate key and verifying the consistency of the keystream generated by the candidate key requires negligible time relatively to the SAT solver. The entire process took approximately 362 seconds on a laptop with processor of 2.80GHz clock, 16 GB RAM and Linux (Ubuntu 22.04) environment. This average time is based on the assumptions that the fault location is correct and the faulty bit is selected every time. If we incorporate all these metrics then the actual required time for a successful DFA on FiLIP-430 is approximately $\approx (\frac{362 \times 1792}{0.24})$ seconds ≈ 751 hours on our specified laptop.

4 Conclusion

In this paper we have proposed DFA on two most recent stream ciphers designed for FHE, namely Rasta and $\text{FiLIP}_{\text{DSM}}$. We have shown that the secret key of both the ciphers can easily be recovered in practical time just by injecting a single bit fault into the key register of the cipher. From our analysis on these two stream ciphers designed for FHE we strongly believe that FHE supported stream ciphers are vulnerable under DFA and the design criteria of these kinds of ciphers need attention.

References

[1] eSTREAM: the ECRYPT Stream Cipher Project. `https://www.ecrypt.eu.org/stream/`.

[2] SageMath: `https://www.sagemath.org/`.

[3] M. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, M. Ciphers for MPC and FHE. In the Proceeding of Advances in Cryptology 2015 (Eurocrypt 2015), LNCS, vol. 9056, pp. 430–454, Springer, 2015.

[4] B. Bathe, S. Tiwari, R. Anand, D. Roy and S. Maitra. Differential Fault Attack on Espresso. In the Proceeding of International Conference on Cryptology in India 2020 (Indocrypt 2020), LNSC, vol. 13143, pp. 271–286, Springer, 2020.

[5] E. Biham and O. Dunkelman. Differential cryptanalysis in stream ciphers. No. CS Technion report CS-2007-10. Computer Science Department, Technion, 2007. Available at: https://eprint.iacr.org/2007/218.pdf.

[6] 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. Journal of Cryptology, vol. 31, no. 3, pp. 885–916, Springer, 2018.

[7] 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, 2022.

[8] C. Carlet, P. Méaux and Y. Rotella. Boolean Functions with Restricted Input and their Robustness; Application to the FLIP Cipher. IACR Trans. Symmetric Cryptology, vol. 3 (2017), pp. 192–227, 2017.

[9] B. Cogliati, and T. Tanguy. T. Multi-user Security Bound for Filter Permutators in the Random Oracle Model. Designs, Codes and Cryptography, vol. 87, pp. 1621–1638, Springer, 2019.

[10] O. Cosseron, C. Hoffmann, P. Méaux, and F. Standaert. Towards Globally Optimized Hybrid Homomorphic Encryption - Featuring the Elisabeth Stream Cipher. Cryptology ePrint Archive, Paper 2022/180, 2022.

[11] 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 the Proceeding of Annual International Cryptology Conference 2018 (Crypto 2018), LNCS, vol. 10991, pp. 662–692, Springer, 2018.

[12] S. Duval, V. Lallemand and Y. Rotella. Cryptanalysis of the FLIP Family of Stream Ciphers. In the Proceeding of Annual International Cryptology Conference 2016 (Crypto 2016), LNSC, vol. 9814, pp. 457–475, Springer, 2016.

[13] J. J. Hoch and A. Shamir. Fault Analysis of Stream Ciphers. In the Proceeding of International Workshop on Cryptographic Hardware and Embedded Systems, LNCS, vol. 3156, pp. 240–253, Springer, 2004.

[14] C. Hoffmann, P. Méaux, and T. Ricosset. Transciphering, Using FiLIP and TFHE for an Efficient Delegation of Computation. In the Proceeding of International Conference on Cryptology in India 2020 (Indocrypt 2020), LNSC, vol. 12578, pp. 39–61, Springer, 2020.

[15] F. Liu, S. Sarkar, W. Meier and T. Isobe. Algebraic Attacks on Rasta and Dasta Using Low-degree Equations. In the Proceeding of International Conference on the Theory and Application of Cryptology and Information Security 2021 (Asiacrypt 2021), LNSC, vol. 13090, pp. 214–240, Springer, 2021.

[16] S. Maitra, B. Mandal, T. Martinsen, D. Roy and P. Stănică. Tools in Analyzing Linear Approximation for Boolean Functions Related to FLIP. In the Proceeding of International Conference on Cryptology in India 2018 (Indocrypt 2018), LNSC, vol. 11356, pp. 282–303, Springer, 2018.

[17] S. Maitra, B. Mandal, T. Martinsen, D. Roy and P. Stănică. Analysis on Boolean Function in a Restricted (Biased) Domain. IEEE Transactions on Information Theory, vol. 66, no. 2, pp. 1219–1231, 2020.

[18] S. Maitra, A. Siddhanti and S. Sarkar. A Differential Fault Attack on Plantlet. IEEE Transactions on Computers, vol. 66, no. 10, pp. 1804–1808, 2017.

[19] P. Méaux. Symmetric Encryption Scheme Adapted to Fully Homomorphic Encryption scheme. In: Journées Codage et Cryptographie - JC2 2015 - 12éme édition des Journées Codage et Cryptographie du GT C2, 5 au 9 octobre 2015, La Londeles-Maures, France, 2015, `http://imath.univ-tln.fr/C2/`.

[20] P. Méaux, C. Carlet, A. Journault and F. Standaert. Improved Filter Permutators for Efficient FHE: Better Instances and Implementations. In the Proceeding of International Conference on Cryptology in India 2019 (Indocrypt 2019), LNSC, vol. 11898, pp. 68–91, Springer, 2019.

[21] P. Méaux, A. Journault, F.-X. Standaert and C. Carlet. Towards Stream Ciphers for Efficient FHE with Low-noise Ciphertexts. In the Proceeding of Advances in Cryptology 2016 (Eurocrypt 2016), LNSC, vol. 9665, pp. 311–343, Springer, 2016.

[22] D. Randall. Efficient Generation of Random Nonsingular Matrices. Random Structures & Algorithms, vol. 4, no. 1, pp. 111–118, Wiley Online Library, 1993.

[23] D. Roy, B. Bathe and S. Maitra. Differential Fault Attack on Kreyvium & FLIP. IEEE Transactions on Computers, vol. 70, no. 12, pp. 2161–2167, 2021.

[24] S. Sarkar, P. Dey, A. Adhikari and S. Maitra. Probabilistic Signature based Generalized Framework for Differential Fault Analysis of Stream Ciphers. Cryptography and Communications, vol. 9, no. 4, pp. 523–543, Springer, 2017.

[25] A. Siddhanti, S. Sarkar, S. Maitra and A. Chattopadhyay. Differential Fault Attack on Grain v1, ACORN v3 and Lizard. In the Proceeding of International Conference on Security, Privacy, and Applied Cryptography Engineering 2017 (SPACE 2017), LNSC, vol. 10662, pp. 247–263, Springer, 2017.