Theoretical differential fault attacks on FLIP and FiLIP

Pierrick Méaux[1] and Dibyendu Roy[2]

[1] Luxembourg university, Esch-sur-Alzette, Luxembourg, `pierrick.meaux@uni.lu`
[2] Indian Institute of Information Technology Vadodara, Gandhinagar, India, `dibyendu.roy@iiitvadodara.ac.in`

Keywords: Fault attacks · FLIP · FiLIP · Boolean functions

Abstract. In this article, we examine Differential Fault Attacks (DFA) targeting two stream ciphers, FLIP and FiLIP. We explore the fault model where an adversary flips a single bit of the key at an unknown position. Our analysis involves establishing complexity bounds for these attacks, contingent upon the cryptographic parameters of the Boolean functions employed as filters and the key size. Initially, we demonstrate how the concept of sensitivity enables the detection of the fault position using only a few keystream bits. This represents an enhancement over previous DFA methodologies applied to these ciphers. Subsequently, we leverage the properties of the filter's derivatives to execute attacks. This approach is universally applicable to any filter, and we delineate specific attack strategies for the two function families previously implemented in these ciphers.

1 Introduction

Since the late 1990s, there has been a growing demonstration and refinement of side-channel attacks targeting implemented cryptographic primitives, as exemplified in works such as [Koc96,BDL97,KJJ99]. These attacks illustrate the inadequacy of the black-box security model, which assumes an adversary only has access to the input and output of an algorithm. This realization has spurred the need to explore the security of cryptographic primitives within a broader model that better aligns with real-world contexts. Throughout the years, various side-channel attacks have been scrutinized. These attacks can be categorized as passive-solely observing the device executing the algorithms or active, involving manipulating the device's computations, often termed fault attacks. Among these, Differential Fault Attacks (DFA) form a subset, operating on comparing a cryptographic primitive's behaviour on an unaltered device with its behaviour after introducing a fault. Initially employed in public key encryption schemes, DFA have been demonstrated on symmetric encryption schemes, commencing with block ciphers [BS97], and and later on stream ciphers [HS04].

Stream ciphers, as symmetric encryption schemes, are typically characterized by public parameters like an Initial Value (IV) and a secret key. These elements are combined within a state to generate a keystream. This keystream is then applied to the plaintext to produce the ciphertext, often via a bit-by-bit XOR operation. Since the 1980s, numerous stream ciphers have been explored in various contexts, including the eSTREAM competition's standardization process [Rob08]. The focus of this article lies in investigating security within a specific attack and fault model. We operate within the known-plaintext/ciphertext model, where the adversary accesses the keystream. However, unlike the chosen plaintext model, the adversary lacks control over the chosen plaintexts or the IV used in the scheme. Regarding faults, we consider a potent attacker equipped with sophisticated tools capable of inducing differences in the cipher state. Specifically, this attacker can flip the value of a single bit within the cipher state without knowledge of its position. In this context of a differential fault attack, the adversary has access to two keystreams: one derived from the un-tampered device (s_i; $i = 1, \dots, m$) and another obtained from the device affected by a single fault (s'_i; $i = 1, \dots, m$). Leveraging these s_i and s'_i, the attacker endeavors to reconstruct the cipher's state, aiming ultimately to retrieve the secret key. If the attacker successfully retrieves the secret key within a feasible time frame, we label the cipher as prone to DFA.

In this particular type of DFA, the process typically involves these steps:

1. Injecting a fault: During the keystream generation phase, a fault is introduced randomly into the state of the cipher.
2. Locating the fault: Determining the position of the injected fault by analyzing both the normal and faulty keystream bits.
3. Forming equations: Constructing a system of equations utilizing the normal and faulty keystream bits.
4. Solving equations: Resolving the system of equations to retrieve the secret key.

Within this article, our focus encompasses steps 2 through 4, delving into the analysis of time and data complexity associated with these attacks. Since the attacker lacks prior knowledge about the fault's location, existing literature presents limited techniques to identify it. Among these, the most prevalent method is signature-based fault identification. This technique has been extensively explored across various stream ciphers, notably in works such as [BM13,MSS17,SSMC17] for identifying the location of the injected fault for Mickey [Bab08], Plantlet [MAM17], Grain [HJMM08], Acorn [Hon16] and Lizard [HKM17].

FLIP [MJSC16] and FiLIP [MCJS19a] are two families of stream ciphers tailored for the context of Hybrid Homomorphic Encryption (HHE) [NLV11]. In the realm of HHE, fault attacks are of particular interest as they target the client side, aiming to recover the symmetric scheme's key. This breach allows decryption of all data sent by the client during the protocol. Symmetric ciphers devised for HHE might be more prone to fault attacks due to their simpler algebraic structure due to the constraints necessary for efficient homomorphic evaluation. Both FLIP and FiLIP rely on distinct paradigms that leverage public wire-cross permutations alongside a single filter function. The core security of these ciphers predominantly derives from the cryptographic properties of the Boolean function utilized as the filter. FLIP employs Direct Sums of Monomials (DSM) as its filter, while FiLIP instantiates two function families: DSM and Xor-Thresholds. In recent studies [RBM20,RKMR23], practical Differential Fault Attacks (DFAs) have been proposed for FLIP and FiLIP instantiated with DSM functions. In [RBM20], Roy *et al.* propose a practical DFA on FLIP (and Kreyvium [CCF+16]). They show that the secret key of FLIP can be recovered by injecting only one bit fault in the key register. In [RKMR23], Radheshwar *et al.* propose a practical DFA on FiLIP instantiated with DSM functions (and Rasta [DEG+18]). The authors show that the secret key of both the ciphers can be recovered in feasible time by injecting only one bit fault in the key register. These DFAs on FLIP and FiLIP instantiated with DSM functions [RBM20,RKMR23] encountered difficulty in predicting the fault position due to the randomized wire-cross permutations within the state. Consequently, these attacks relied on guessing the fault's location. For each guess, a system of equations involving state bits was constructed. These equations were then solved, and the correct key was discerned using previously known keystream bits from the multitude of solutions.

1.1 Our contributions

This paper presents theoretical differential fault attacks on both FLIP and FiLIP, proving the complexity bounds of these attacks based on the cryptographic parameters of the Boolean function used as filters and the key size. Specifically:

– First, we revisit the DFA proposed in prior works [RBM20,RKMR23] by introducing a novel technique to identify the fault's location in the key register. We demonstrate that the concept of the influence of a coordinate is sufficient to determine the fault position provably. By examining this criterion on specific derivatives of the filter function used in FLIP (or FiLIP), we can pinpoint the fault location with a limited number of keystream bits and high probability. This technique corresponds to Algorithm 1 for both the ciphers. It exhibits an asymptotic complexity bounded by $\mathcal{O}(mn)$ both for FLIP and FiLIP with the filter function considered, where n denotes the key size and m denotes the number of keystream bits used. In practice, it significantly enhances prior DFAs [RBM20,RKMR23] by discarding fault positions with only a few keystream bits instead of necessitating n individual attacks for each possible fault position.

- Then, once the fault position is determined, we illustrate how conducting a DFA on FLIP (or FiLIP) with filter function f parallels executing a standard attack on a specific derivative of f. Additionally, we correlate the complexity of key recovery in the DFA of F(i)LIP to attacks on the paradigms of FLIP and FiLIP using other functions [MJSC16,MCJS19b]. This contribution offers insight into finding filter functions for FLIP and FiLIP paradigms that can mitigate these DFAs. Boolean functions where all derivatives (in vectors of Hamming weight one) are secure filters for F(i)LIP in the black-box model.
- Finally, for the key recovery phase, we introduce two strategies tailored to the specific filter functions of FLIP and FiLIP. For FLIP, we investigate the properties of the derivative of DSM and introduce a novel equation generation technique in Algorithm 2, applicable to all proposed instances of FLIP ciphers. We prove that in this scenario, the secret key of FLIP can be recovered with high probability at a complexity of $\mathcal{O}(nm)$. Additionally, empirical experiments showcase the practical aspect of the attack on an instance with $n = 530$, demonstrating that the secret key can be retrieved in less than a minute with a single-bit fault in the key register. For FiLIP, we develop a strategy tailored to Xor-threshold instances. Upon identifying the fault location, we demonstrate that with these filters, we can formulate a linear system of equations involving the secret key bits using Algorithm 3. This approach results in a successful DFA with high probability at a complexity of $\mathcal{O}(m^{\omega})$, where ω denotes the exponent for linear system inversion ($2 < \omega \leq 3$). We also perform concrete experiments on one instance of FiLIP$_{\text{XOR-THR}}$ to illustrate the practical impact of these attacks.

1.2 Comparison with former works

The precision in locating the fault significantly impacts the attack's performance. In both [RBM20] and [RKMR23], the attacker posits a hypothesis about the fault's location and subsequently solves the corresponding system of equations derived from XORing the actual and faulty keystreams. In this scenario, the attacker must solve n algebraic systems of equations for FLIP (or N for FiLIP), followed by testing the candidate key. Contrastingly, our techniques enable determining the fault location using a bounded number of equations, reducing the process to solving only one system. Given that the key size typically ranges between several hundred to a few thousand bits for these schemes, this distinction holds substantial significance. To illustrate, for a scheme comprising n key bits and denoting T_1 as the time complexity of fault detection and T_2 as the complexity of solving the faulty system, our techniques curtail the complexity from previous attacks, diminishing it from $\mathcal{O}(nT_2)$ to $\mathcal{O}(T_1 + T_2)$.

Comparing the time taken for attacks in [RBM20] and [RKMR23] on FLIP-530 and FiLIP instances highlights the efficiency gains of our fault location techniques. In [RBM20], for FLIP-530, the average time reported is $n/2 \times 262$ seconds, equivalent to approximately 19 hours on average. Conversely, our method achieves this task in less than a minute for the same instance. In [RKMR23], the attack's average runtime is $1508 \times N/2$ seconds, approximately 375 hours. Our fault location technique improves this by a factor of around $N/2$, significantly reducing the time required to a few minutes. Specifically, for an instance of FiLIP using Xor-Threshold functions, the attack completes in only 580 seconds. For a clearer comparison, we have summarized these timings in Table 1.

Scheme instance	FLIP-530	FLIP-530	FiLIP-DSM-1792	FiLIP-XThr-1024
Reference	[RBM20]	this work	[RKMR23]	this work
Time	19h	32s	751h	581s

Table 1: Comparison of the average time of DFA on FLIP and FiLIP

2 Preliminaries

We use the following notations in this article. We use $[n]$ to represent the set of integers from 1 to n, both included and $+$ instead of $\oplus$ for the addition over $\mathbb{F}_2$. For an element $a \in \mathbb{F}_2$ we denote by $\mathsf{w_H}(a)$ its Hamming weight $\mathsf{w_H}(a) = \#\{i \in [n] \,|\, a_i = 1\}$. $\mathsf{E}_{k,n}$ denotes the set $\{a \in \mathbb{F}_2^n \,|\, \mathsf{w_H}(a) = k\}$. The Hamming weight of a function refers to the Hamming weight of its truth table. For $j \in [n]$, $\mathbf{e}_j$ refers to the element of $\mathbb{F}_2^n$ with j-th coordinate being 1 and the $n-1$ others being 0. We denote by $\log$ the logarithm in basis 2.

We refer to wire-cross permutations of size n the re-arrangements of length-n vectors. For a wire-cross permutation $P \in \mathcal{S}_n$, for an element $x \in \mathbb{F}_2^n$ we denote by $P(x) \in \mathbb{F}_2^n$ the permutation of x and for an integer $i \in [n]$ we denote by $P(i)$ the position of the i-th element after applying P, *i.e.* $P(i) = \mathsf{supp}(P(\mathbf{e}_i))$.

Let $m, m' \in \mathbb{N}$, $m' \leq m$ and $p \in \mathbb{R}, 0 \leq p \leq 1$, we denote by $\mathsf{B}\left(m', m, p\right)$ the probability $\Pr\left[X \geq m'\right]$ when X is a random variable following the binomial law with probability of success p and number of tries m. The value of $\mathsf{B}\left(m', m, p\right)$ can be obtained from the cumulative distribution of such binomial law.

2.1 Boolean Functions

We recall general concepts on Boolean functions we use in this article; for a deeper introduction to Boolean functions and their cryptographic properties, we refer to the book [Car21].

Definition 1 (Boolean Function). *A Boolean function f with n variables is a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. The set of all Boolean functions in n variables will be denoted by $\mathcal{B}_n$.*

Definition 2 (Algebraic Normal Form (ANF)). *We call Algebraic Normal Form of a Boolean function f its n-variable polynomial representation over $\mathbb{F}_2$ (i.e. belonging to $\mathbb{F}_2[x_1, \ldots, x_n]/(x_1^2+x_1, \ldots, x_n^2+x_n)$):*

$$f(x) = \sum_{I \subseteq [n]} a_I \left(\prod_{i \in I} x_i \right) = \sum_{I \subseteq [n]} a_I x^I,$$

where $a_I \in \mathbb{F}_2$.

- *The algebraic degree of f is $\mathsf{deg}(f) = \max_{\{I \,|\, a_I = 1\}} |I|$ (with the convention that $\mathsf{deg}(0) = 0$).*
- *Any term $\prod_{i \in I} x_i$ in such an ANF is called a monomial and its degree equals $|I|$. A function with only one non-zero coefficient a_I, where I is non-empty, is called a monomial function.*

Definition 3 (Derivative). *Let $f \in \mathcal{B}_n$ and $a \in \mathbb{F}_2^n$, we call derivative in the direction a (or with input difference a) of f the Boolean function:*

$$\mathsf{D}_a f(x) = f(x) + f(x+a).$$

We introduce the different families of Boolean functions used to instantiate the filter functions of FLIP of FiLIP.

Definition 4 (Direct sum). *Let f be a Boolean function of n variables and g be a Boolean function of m variables, the direct sum h of f and g is defined by:*

$$h(x,y) = f(x) + g(y), \quad \text{where } x \in \mathbb{F}_2^n \text{ and } y \in \mathbb{F}_2^m.$$

Definition 5 (Direct Sum of Monomials). *Let f be a non-constant n-variable Boolean function. We call f a Direct Sum of Monomials (or DSM) if the following holds for its ANF:*

$$\forall (I,J) \text{ such that } a_I = a_J = 1,\ I \cap J \in \{\emptyset, I \cup J\}.$$

In other words, in the ANF of such functions, each variable appears at most once.

Definition 6 (Direct Sum Vector [MJSC16]). *Let f be a DSM, we define its direct sum vector (DSV):*

$$\mathbf{m}_f = [m_1, m_2, \dots, m_k],$$

of length $k = \mathsf{deg}(f)$, where m_i is the number of monomials of degree i, $i > 0$, in the ANF of f:

$$m_i = |\{I \subset [n]\}; a_I = 1 \text{ and } |I| = i\}|.$$

Definition 7 (Majority Function). *For any positive integer n we define the Boolean function MAJ_n as:*

$$\forall x = (x_1, \dots, x_n) \in \mathbb{F}_2^n, \quad \mathsf{MAJ}_n(x) := \begin{cases} 0 & \text{if } \mathsf{w_H}(x) \le \frac{n}{2}, \\ 1 & \text{otherwise.} \end{cases}$$

Definition 8 (Threshold Function). *For any positive integer $d \le n+1$ we define the Boolean function $\mathsf{T}_{d,n}$ as:*

$$\forall x = (x_1, \dots, x_n) \in \mathbb{F}_2^n, \quad \mathsf{T}_{d,n}(x) = \begin{cases} 0 & \text{if } \mathsf{w_H}(x) < d, \\ 1 & \text{otherwise.} \end{cases}$$

Definition 9 (Xor-Threshold Function). *For any positive integers k, d and n such that $d \le n+1$ we define $\mathsf{XOR}_k + \mathsf{T}_{d,n}$ for all $z = (x_1, \dots, x_k, y_1, \dots, y_n) \in \mathbb{F}_2^{k+n}$ as follows:*

$$(\mathsf{XOR}_k + \mathsf{T}_{d,n})(z) = x_1 + \dots + x_k + \mathsf{T}_{d,n}(y_1, \dots, y_n) = \mathsf{XOR}_k(x) + \mathsf{T}_{d,n}(y).$$

2.2 FLIP and FiLIP

(Improved) filter permutator paradigm of stream ciphers FLIP and FiLIP are families of stream ciphers following the Filter Permutator paradigm (FP) [MJSC16] and Improved Filter Permutator paradigm (IFP) [MCJS19a]. In Figure 1 we represent these two paradigms.

The filter permutator is composed of four parts: a register where the key is stored, a forward secure PseudoRandom Number Generator (PRNG) initialized with a public IV, a generator of wire-cross permutations, and a filter function which produces the keystream. For a security parameter λ, to encrypt $m \le 2^\lambda$ bits under a secret key $K \in \mathbb{F}_2^n$ (such that $\mathsf{w_H}(K) = n/2$), the public parameters of the PRNG are chosen and then the following process is executed for each keystream bit s_i (for $i \in [m]$):

- The PRNG is updated, its output determines the n to n elements permutation P_i at time i,
- the keystream bit s_i is computed as $s_i = f(P_i(K))$.

The IFP contains the four parts of the filter permutator and two generators, one to select subsets of the key and one to generate fixed size binary strings called whitenings. The main differences of these modifications to the FP paradigm are that the key register is bigger than the input size of the filter function, and that a pseudorandom binary string is added to the input of the filter functions. For a security parameter λ, to encrypt $m \leq 2^\lambda$ bits under a secret key $K \in \mathbb{F}_2^N$, the public parameters of the PRNG are chosen. Then the following process is executed for each keystream bit s_i (for $i \in [m]$):

- The PRNG is updated, its output determines the subset S_i of n-out-of-N elements, the permutation P_i from n to n elements at time i, and the whitening w_i, that is a n-size binary vector,
- the subset is applied to the key,
- the permutation is applied,
- the whitening is added,
- the keystream bit s_i is computed, $s_i = f(P_i(S_i(K)) + w_i)$.

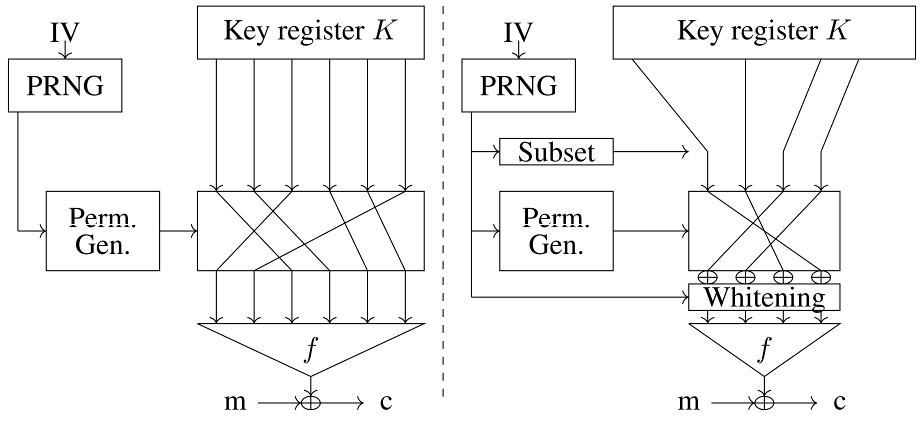

Fig. 1: Filter permutator and improved filter permutator paradigms.

FLIP FLIP [MJSC16] stream cipher family is an instantiation of the FP paradigm where the PRNG is the forward secure PRG construction from [BY03] using the AES as underlying block cipher. The filter functions are all DSM functions (see Definition 5). We recall them in Table 2.

n	$\mathbf{m}_f$	λ
530	$[50, 72, 8, 8, 8, 8, 8, 8, 8]$	80
662	$[50, 72, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]$	80
1394	$[90, 120, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8]$	128
1704	$[91, 124, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]$	128

Table 2: FLIP instances

FiLIP FiLIP [MCJS19a] stream cipher family is an instantiation of the IFP paradigm. The PRG is instanced, but for FLIP, there are more differences in the filter functions. Two kinds of instances have been introduced, with DSM functions and with Xor-Threshold functions in the long version [MCJS19b], we recall them in Table 3.

n	N	$\mathbf{m}_f$	λ
512	16384	$[89, 67, 47, 37]$	80
430	1792	$[80, 40, 15, 15, 15, 15]$	80
320	1800	$[80, 40, 0, 20, 0, 0, 0, 10]$	128
1216	16384	$[128, 64, 0, 80, 0, 0, 0, 80]$	128
1280	4096	$[128, 64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 64]$	128
n	N	$\mathsf{XOR}_k + \mathsf{T}_{d,m}$	λ
144	2048	$113, 16, 31$	80
144	2048	$81, 32, 63$	80
256	1536	$129, 64, 127$	80
512	1024	$257, 128, 255$	80
256	8192	$129, 64, 127$	128
512	4096	$257, 128, 255$	128

Table 3: FiLIP instances

3 DFA attack on filter permutators

3.1 Generalizing the attack of [RBM20] on FLIP

Using the description of FLIP (Section 2.2), the i-th keystream bit can be written as $s_i = f(P_i(x))$. When a fault has been injected in the j-th position of the key register, the i-th produced output can be written as $s'_i = f(P_i(x + \mathbf{e}_j))$. The DFA adversary is able to get the two keystreams and hence to obtain for each output:

$$s"_i = s_i + s'_i = f(P_i(x)) + f(P_i(x + \mathbf{e}_j)) = \mathsf{D}_{P_i(\mathbf{e}_j)} f(P_i(x)),$$

where $\mathsf{D}_{P_i(\mathbf{e}_j)} f(P_i(x))$ denotes the derivative of the function $f(P_i(x))$ in the direction $P_i(\mathbf{e}_j)$. Therefore, the DFA adversary can obtain a system of equations which corresponds to the keystream of an instance of FLIP with the same IV, the same key with one variable less (x_j), and each time one of the n filter functions $\mathsf{D}_{\mathbf{e}_{j'}} f$; particular derivative of the initial filter function.

In [RBM20], the attack is presented on FLIP_{530}, where the filter function is a DSM of degree 9. The attack uses the simple algebraic structure of the functions $\mathsf{D}_{\mathbf{e}_j} f$ for $j \in [n]$; monomial functions of degree at most 8, to solve the system of equations given by $s"_i$ with a SAT-solver. The attack is performed for the n possible locations of the fault: for each $j \in [n]$ the system given by the $s"_i$ is solved with the SAT-solver, if the solution is compatible with the Hamming weight of K (*i.e.* $n/2$) then it is kept in the set of solutions $\mathcal{S}$. After solving the n systems, if $|\mathcal{S}| > 1$, the attacker generates a large number of keystream bits from each potential solution and compares it with the keystream given by s_i which were previously collected for the unknown key. Assuming the time complexity of the SAT-solver is T, and that it dominates the complexity of the keystream comparisons to discard false solutions, the total time complexity is $\mathcal{O}(nT)$.

In the following, we show how to improve the complexity of the attack by determining first the position of the fault, and then applying the attack on $\mathsf{D}_{\mathbf{e}_j} f$. In this case the time complexity decreases to $\mathcal{O}(T_1+T_2)$, where T_1 is the complexity of determining the position j of the fault, and T_2 can be taken as the complexity of the best attack against FLIP with filter function $\mathsf{D}_{\mathbf{e}_j} f$. In Section 3.2 we show how the complexity can be estimated from a filter permutator with filter function f, by means of cryptographic properties of f. In Section 3.3 we focus on the filter functions used for the cipher FLIP, namely direct sum of monomials, and give precise attack complexity bounds in this case.

3.2 DFA on filter permutators and filter properties

Determining the fault position First, to determine where the fault occurs we recall the concept of the influence of a coordinate and introduce the notion of 0-influence set and 1-influence set of a function.

Definition 10 (Influence *e.g.* [KKL88]). *Let $f \in \mathcal{B}_n$, the influence of a coordinate $i \in [n]$ on f is:*

$$\mathsf{inf}_i(f) = \frac{\#\{x \in \mathbb{F}_2^n \mid f(x) \neq f(x+\mathbf{e}_i)\}}{2^n}.$$

Additionally we denote by $\mathsf{inf}_{i,S}(f)$ the influence of i on f when its inputs are taken from the set $S \subset \mathbb{F}_2^n$ only: $\mathsf{inf}_{i,S}(f) = \#\{x \in S \mid f(x) \neq f(x+\mathbf{e}_i)\}/|S|$.

The influence measures how much the variation of a coordinate affects the output. The following definition focuses on the two extreme cases where a coordinate has no influence, or full influence, on the output.

Definition 11. *Let $f \in \mathcal{B}_n$, we call 0-influence set $\mathsf{Inf}^0(f) = \{i \in [n] \mid \mathsf{inf}_i(f) = 0\}$ and 1-influence set $\mathsf{Inf}^1(f) = \{i \in [n] \mid \mathsf{inf}_i(f) = 1\}$.*

The following proposition allows to link the values of the stream obtained by the DFA adversary and the 0-influence set (respectively 1-influence set) of the filter:

Proposition 1. *Let $f \in \mathcal{B}_n$, $j' \in [n]$, P be a (wire-cross) permutation of $\mathbb{F}_2^n$, and $s"(x) = \mathsf{D}_{\mathbf{e}_{j'}} f(P(x))$ then:*

- *If $s"(x) = 0$ then $P^{-1}(j') \notin \mathsf{Inf}^1(f)$,*
- *if $s"(x) = 1$ then $P^{-1}(j') \notin \mathsf{Inf}^0(f)$.*

Proof. For both the cases we show the contrapositive. If $P^{-1}(j') \in \mathsf{Inf}^1(f)$ then from Definition 11 $\mathsf{inf}_{j'}(f(P(x))) = 1$ therefore $s"(x) = \mathsf{D}_{\mathbf{e}_{j'}} f(P(x)) = 1$. By similar arguments, $P^{-1}(j') \in \mathsf{Inf}^0(f)$ implies $s"(x) = \mathsf{D}_{\mathbf{e}_{j'}} f(P(x)) = 0$, finishing the proof. □

Then, using the stream $s"_i$ and the influence sets of f we propose Algorithm 1 to determine the fault location.

The principle of the algorithm is to use each stream value $s"_i$ to discard potential positions of j, applying Proposition 1. Accordingly, the algorithm is useful for an attacker provided the two influence sets are not both null. We study the complexity of Algorithm 1 in the following proposition.

Proposition 2. *Let f be an n-variable function f with influence sets $\mathsf{Inf}^0(f)$ and $\mathsf{Inf}^1(f)$. A DFA adversary can recover the fault position $j \in [n]$ of FLIP$_f$ using m keystream bits $s"_i$ with the time complexity $\mathcal{O}(mn)$ and the success probability $Pr\left[L = \{j\}\right]$.*

```
Input: Keystream size $m$, binary stream $s"_i$ for $i \in [m]$, permutations $P_i$ for $i \in [m]$, and $f$'s
       influence sets $\mathsf{Inf}^0(f)$ and $\mathsf{Inf}^1(f)$.
Output: a set $S \subseteq [n]$ containing the fault position $j$.
L := $[n]$ ; // The candidate list of fault positions
cpt := 1 ;
while Cardinal(L)>1 & cpt $\leq m$ do
    if $s"_{cpt} = 0$ then
        for $k \in \mathsf{Inf}^1(f)$ do
            Compute $P_i(k)$;
            Remove $P_i(k)$ from $L$;
        end
    end
    if $s"_{cpt} = 1$ then
        for $k \in \mathsf{Inf}^0(f)$ do
            Compute $P_i(k)$;
            Remove $P_i(k)$ from $L$;
        end
    end
    cpt ← cpt +1;
end
return L
```

Algorithm 1: Determining the fault position.

– *If* $0 < \ell = \min(|\mathsf{Inf}^0(f)|, |\mathsf{Inf}^1(f)|)$ *then* $Pr\left[L = \{j\}\right] \geq 1 - (n-1)\left(1 - \frac{\ell}{n-1}\right)^m$.

– *If* $\mathsf{Inf}^\varepsilon(f) \neq 0$ *and* $|\mathsf{Inf}^{1-\varepsilon}(f)| = 0$ *for* $\varepsilon \in \{0, 1\}$ *then:*

$$Pr\left[L = \{j\}\right] \geq \mathsf{B}\left(m', m, p\right)\left(1 - (n-1)\left(1 - \frac{|\mathsf{Inf}^\varepsilon(f)|}{n-1}\right)^{m'}\right),$$

where $m' \in [m]$ *and* $p = \varepsilon + (1 - 2\varepsilon)\frac{1}{n}\sum_{j'=1}^{n} \mathsf{inf}_{j', \mathsf{E}_{\frac{n}{2}, n}}(f)$.

Proof. We begin with the case where $\mathsf{Inf}^0(f)$ and $\mathsf{Inf}^1(f)$ are both not empty. Denoting $\ell = \min(|\mathsf{Inf}^0(f)|, |\mathsf{Inf}^1(f)|)$, each equation alone allows removing at least ℓ possible positions for j. Since the permutations P_i are uniformly distributed we can estimate the number of equations needed before having the set L containing only j using the "balls-in-bins" approach. Each equation gives at least ℓ random positions of $[n] \setminus \{j\}$, then the probability for one of the $n - 1$ positions (seen as bins), called j', to have not been removed after m equations (seen as m launches of at least ℓ balls) is: $\Pr\left[j' \in L\right] \leq \left(1 - \frac{\ell}{n-1}\right)^m$. Having only j in L after m equations corresponds to having $n - 1$ other positions appearing in at least one equation, hence by union bound after m equations $\Pr\left[L = \{j\}\right] \geq 1 - (n-1)\left(1 - \frac{\ell}{n-1}\right)^m$.

When only one of the influence sets is non-null, only one part of the equations allows removing potential positions. In the case $\mathsf{Inf}^1(f) \neq \emptyset$ (respectively $\mathsf{Inf}^0(f) \neq \emptyset$) only the equations such that $s"_i = 0$ (respectively $s"_i = 1$) allow to decrease L in Algorithm 1. Hence, we can study the probability of the algorithm returning j similarly as before by pondering it by the probability $\Pr\left[s"_i = 1\right]$. More precisely, the probability of getting at least m' over m equations such that $s"_i = 0$ (respectively $s"_i = 1$) is given by a binomial law with parameter $p = 1 - \mathsf{Pr}\left[s"_i = 1\right]$ (respectively ($p = \mathsf{Pr}\left[s"_i = 1\right]$). Hence we compute $\mathsf{Pr}\left[s"_i = 1\right]$.

We know that $s"_i = \mathsf{D}_{P_i(\mathbf{e}_j)} f(P_i(x))$ then the uniform distribution of the permutations P_i implies:

$$\begin{aligned}
\mathsf{Pr}\left[s"_i = 1\right] &= \sum_{j'=1}^{n} \mathsf{Pr}\left[P_i(j) = j'\right] \cdot \mathsf{Pr}\left[\mathsf{D}_{\mathbf{e}_{P_i(j)}} f(P_i(x)) = 1 \mid P_i(j) = j'\right] \\
&= \frac{1}{n} \sum_{j'=1}^{n} \mathsf{Pr}\left[\mathsf{D}_{\mathbf{e}_{j'}} f(P_i(x)) = 1 \mid P_i(j) = j'\right] \\
&= \frac{1}{n} \sum_{j'=1}^{n} \mathsf{Pr}\left[f(P_i(x)) \neq f(P_i(x + \mathbf{e}_j))\right] \\
&= \frac{1}{n} \sum_{j'=1}^{n} \mathsf{inf}_{j', \mathsf{E}_{\frac{n}{2}, n}}(f) = p_f,
\end{aligned}$$

where the probability is taken over the random wire-cross permutations P_i. Thereafter, in the case $\mathsf{Inf}^1(f) \neq \emptyset$ (respectively $\mathsf{Inf}^0(f) \neq \emptyset$), the algorithm recovers j with probability:

$$\mathsf{Pr}\left[L = \{j\}\right] \geq \mathsf{B}\left(m', m, p\right) \left(1 - (n-1)\left(1 - \frac{|\mathsf{Inf}^{\varepsilon}(f)|}{n-1}\right)^{m'}\right),$$

where $m' \in [m]$, and $p = 1 - p_f$ (respectively $p = p_f$).

We finish with the time complexity of Algorithm 1. Since for each equation used by the algorithm it computes at most $\max(|\mathsf{Inf}^0(f)|, |\mathsf{Inf}^1(f)|)$ indexes to remove from L, its time complexity is (roughly) upper bounded by $\mathcal{O}(nm)$. $\square$

Note that even if Algorithm 1 does not return j but a bigger set containing it, a key recovery attack is still possible. An attacker can use the algorithm until obtaining a set of n' potential positions (with $1 < n' < n$), and then apply the strategy presented in [RBM20]: solve the equation system for the n' positions and determine the correct one. It leads to a time complexity of $\mathcal{O}(nm + n'T)$ where T denotes the time complexity of solving the equation system.

Attack on $\mathsf{D}_{\mathbf{e}_j} f$ Once the faulted position j is known, for each keystream element $s"_i$ the attacker knows which one of the derivatives gives this value. Since the permutations are uniformly distributed $P_i(j)$ is uniformly distributed. Hence, each n derivative has the same probability of being obtained. One attack strategy consists of considering only the equations corresponding to the derivative g and mounting the attack on the filter permutator with this filter function. In this case, the attacker obtains a system of equations with shape $s"_i = g(P_i(x))$, in the $n - 1$ non-faulted variables of the key, and can therefore apply any attack on the filter permutator in $n - 1$ variables and filter function g.

Different attacks are considered for this paradigm, and listed in [MJSC16] (Section 3.4), such as algebraic attacks [CM03] and fast algebraic attacks [Cou03a], correlation attacks [Sie84] and high order correlation attacks [Cou03b]. All these attacks can be improved with guess and determine techniques, as illustrated by the cryptanalysis of Duval *et al.* [DLR16] on a preliminary version of FLIP, and attacks exploiting the Hamming weight invariance of the filter's input are explored in [CMR17]. Accordingly, denoting by T_2 and D_2 the time and data complexity of the key recovery attack of FLIP instantiated with g

and an $n-1$-bit secret key, the time complexity of the part of this DFA after determining the fault position is $\mathcal{O}(T_2)$. Regarding the data complexity, it is $\mathcal{O}(p_g^{-1}D_2)$, where p_g denotes the probability of obtaining the derivative g.

3.3 Differential properties of DSM and attack on FLIP

Derivative of DSM functions

Proposition 3. *Let $n \in \mathbb{N}^*$, $f \in \mathcal{B}_n$ a DSM and $j \in [n]$, the following holds on $\mathsf{D}_{\mathbf{e}_j} f$:*

- *If x_j does not appear in the ANF of f, then $\mathsf{D}_{\mathbf{e}_j} f$ is the constant function 0.*
- *If x_j appears in a monomial of degree 1 of f then $\mathsf{D}_{\mathbf{e}_j} f$ is the constant function 1.*
- *If x_j appears in a monomial of degree $t > 1$ of f then $\mathsf{D}_{\mathbf{e}_j} f$ is a monomial function of degree $t-1$.*

Proof. We focus on the expression of $f(x+\mathbf{e}_j)$ in the different cases. In the first case, since x_j does not appear in the ANF of f we get $f(x+\mathbf{e}_j) = f(x)$ hence $\mathsf{D}_{\mathbf{e}_j} f = 0$.

In the second case, f can be written as the direct sum of x_j and g in $n-1$ variables (that we will denote by x'), hence $f(x+\mathbf{e}_j) = x_j + 1 + g(x')$ and therefore

$$\mathsf{D}_{\mathbf{e}_j} f(x) = x_j + g(x') + x_j + 1 + g(x') = 1.$$

In the third case, f can be written as the direct sum of $x_j \prod_{k=1}^{t-1} x_{i_k}$ (where $i_1, \cdots, i_{t-1}$ are distinct indices in $[n]$) and g in $n-t$ variables (that we denote by x'), hence $f(x+\mathbf{e}_j) = (x_j+1)\prod_{k=1}^{t-1} x_{i_k} + g(x')$ and therefore:

$$\mathsf{D}_{\mathbf{e}_j} f(x) = x_j \prod_{k=1}^{t-1} x_{i_k} + g(x') + x_j \prod_{k=1}^{t-1} x_{i_k} + \prod_{k=1}^{t-1} x_{i_k} + g(x') = \prod_{k=1}^{t-1} x_{i_k}.$$

Hence $\mathsf{D}_{\mathbf{e}_j} f$ is a monomial function of degree $t-1$, concluding the proof. □

Proposition 4. *Let $n \in \mathbb{N}^*$ and $f \in \mathcal{B}_n$ a DSM with DSV $\mathbf{m}_f = [m_1, \cdots, m_k]$:*

- $|\mathsf{Inf}^0(f)| = n - \sum_{i=1}^{k} m_i$.
- $|\mathsf{Inf}^1(f)| = m_1$.
- $\mathsf{inf}_{i,\mathsf{E}_{\frac{n}{2},n}}(f)$ *takes the value:*

$$\begin{cases} 0 & \text{If } x_i \text{ does not appear in the ANF of } f \\ 1 & \text{If } x_i \text{ appears in a monomial of degree } 1\text{, i.e. the } m_1 \text{ part.} \\ \frac{\binom{n-(d-1)}{\frac{n}{2}-(d-1)}}{\binom{n}{\frac{n}{2}}} & \text{If } x_i \text{ appears in a monomial of degree } d\text{, i.e. the } m_d \text{ part.} \end{cases}$$

Proof. We begin with the two first items. From the definition of the influence (Definition 10) for any function f we get:

$$\mathsf{inf}_i(f) = \frac{\#\{x \in \mathbb{F}_2^n \mid f(x) \neq f(x+\mathbf{e}_i)\}}{2^n} = \frac{2\mathsf{w_H}(\mathsf{D}_{\mathbf{e}_i} f)}{2^n}.$$

Hence $\mathsf{inf}_i(f) = 0 \Leftrightarrow \mathsf{w_H}(\mathsf{D}_{\mathbf{e}_i} f) = 0$ which means $\mathsf{D}_{\mathbf{e}_i} f$ is the null function, and $\mathsf{inf}_i(f) = 1 \Leftrightarrow \mathsf{w_H}(\mathsf{D}_{\mathbf{e}_i} f) = 2^{n-1}$ which means $\mathsf{D}_{\mathbf{e}_i} f$ is the constant function 1. Combining it with Proposition 3, since f is a DSM, $\mathsf{inf}_i(f) = 0$ if and only if x_i does not appear in its ANF, which is the case for $n - \sum_{i=1}^{k} m_i$ by definition of the DSV. On the other side, $\mathsf{inf}_i(f) = 1$ if and only if x_i appears in a monomial of degree 1, which is the case for exactly m_1 variables by definition of the DSV. It allows to conclude on $|\mathsf{Inf}^0(f)|$ and $|\mathsf{Inf}^1(f)|$.

For the last part, by definition:

$$\mathsf{inf}_{i,\mathsf{E}_{\frac{n}{2},n}}(f) = \frac{\#\{x \in \mathsf{E}_{\frac{n}{2},n} \mid \mathsf{D}_{\mathbf{e}_i} f(x) = 1\}}{\#\mathsf{E}_{\frac{n}{2},n}} = \frac{\#\{x \in \mathsf{E}_{\frac{n}{2},n} \mid \mathsf{D}_{\mathbf{e}_i} f(x) = 1\}}{\binom{n}{\frac{n}{2}}}.$$

Using Proposition 3, $\mathsf{D}_{\mathbf{e}_i} f$ is either the null function (giving $\mathsf{inf}_{i,S}(f) = 0$ for any set S), the constant function 1 (giving $\mathsf{inf}_{i,S}(f) = 1$ for any set S), or a monomial function of degree $d-1$. In the latter case, over the $\binom{n}{n/2}$ elements of $\mathsf{E}_{n/2,n}$, only the ones with the $d-1$ variables of the degree $d-1$ monomial being ones give $\mathsf{D}_{\mathbf{e}_i} f = 1$, which corresponds to the $\binom{n-(d-1)}{n/2-(d-1)}\binom{d-1}{d-1}$ elements having $d-1$ variables equal to one in the degree $d-1$ monomial and the $n/2-(d-1)$ other variables equal to one in the non selected variables.

□

Corollary 1. *Let $f \in \mathcal{B}_n$ a DSM, if $\mathbf{m}_f = [m_1, \cdots, m_k]$ then:*

$$\frac{1}{n}\sum_{i=1}^{n} \mathsf{inf}_{i,\mathsf{E}_{\frac{n}{2},n}}(f) = \frac{1}{n}\sum_{i=1}^{k} i m_i \frac{\binom{n-(i-1)}{\frac{n}{2}-(i-1)}}{\binom{n}{\frac{n}{2}}} \geq \frac{m_1}{n}.$$

Based on the derivatives of DSM functions we consider a dedicated attack on FLIP. First the fault position j is determined with Algorithm 1 using that $|\mathsf{Inf}^1(f)| \neq 0$ for these filters (Proposition 4). Then, we consider the following attack strategy:

Strategy S1: The attacker collects the equations where $\mathsf{D}_{P_i(\mathbf{e}_j)} f$ is a degree 1 function, since these equations are of type $s"_i = x_k$ it directly gives the value of x_k. When all variables except x_j have been determined, x_j is obtained from the Hamming weight of the key (fixed to $n/2$).

We describe the attack scenario as Algorithm 2 and study its complexity in the following proposition.

Proposition 5. *Let $n \in \mathbb{N}^*$ and f be an n-variable DSM with DSV $\mathbf{m}_f = [m_1, \cdots, m_k]$, a DFA adversary implementing strategy S1 can recover the key of FLIP_f using m keystream bits $s"_i$ with the time complexity $\mathcal{O}(nm)$ and the success probability $p_s \geq p_1 p_2$, where p_1 is the probability of success of Algorithm 1 and*

$$p_2 = \mathsf{B}\left(m', m, \frac{2m_2}{n}\right)\left(1 - (n-1)\left(1 - \frac{1}{n-1}\right)^{m'}\right),$$

with $m' \in [m]$.

Proof. The time complexity and success probability are obtained by analyzing the complexity of Algorithm 1 and then Algorithm 2. The time complexity and success probability for Algorithm 1 come from Proposition 2, and we denote it T_1 and p_1 in the remaining part of the proof.

The cost of Algorithm 1 apart, the time complexity of Algorithm 2 is dominated by the while loop, where each time it does at most 3 tests and 4 affectations/additions. Hence, the while loop leads to a complexity of $\mathcal{O}(m)$, and then the total complexity remains in $\mathcal{O}(mn)$. The success probability can be estimated as $p_s = p_1 p_2$ where p_2 refers to the probability of determining entirely K provided j, *i.e.* the probability of determining the $n-1$ key bits during the while loop. We bound this probability with a similar manner as

```
Input: Keystream size $m$, binary stream $s"_i$ for $i \in [m]$, permutations $P_i$ for $i \in [m]$, $n$ and $\mathbf{m}_f$.
Output: key K.
K := $[x_1, \dots, x_n]$ ; // key with undetermined values
j := Algorithm 1 on inputs ($m$, $s"_i$ for $i \in [m]$, $P_i$ for $i \in [m]$, $[m_1]$, $\emptyset$) ;
i := 1 ;
w := 0 ;
cpt := 0 ;
// the counter of determined key values
while $cpt < n-1$ & $i \leq m$ do
    // Tests if $P_i(j)$ belongs to the $m_2$ part
    if $\mathsf{D}_{P_i(\mathbf{e}_j)} f$ is a degree-1 monomial $x_k$ then
        // Tests if $x_k$ is undetermined
        if $x_k \in K$ then
            $K[k] \leftarrow s"_i$ ;
            cpt ← cpt+1 ;
            if $x_k = 1$ then
                w ← w+1 ;
            end
        end
    end
    i ← i+1 ;
end
if $cpt = n-1$ then
    K[j]← $n/2-$w ;
end
return K ;
```

Algorithm 2: Attack strategy S1.

in the proof of Proposition 2, using the "balls in bins" approach. Since the permutations P_i are uniformly distributed j appears in a degree 2 monomial with probability $2m_2/n$ and in this case $\mathsf{D}_{P_i(\mathbf{e}_j)} f = x_k$ where k is uniformly distributed in $[n] \setminus \{j\}$. Accordingly, the probability of having at least m' over m equations with j in the degree-2 part for $m' \in [m]$ is given by $\mathsf{B}\left(m', m, 2m_2/n\right)$. Over m' such equations, each variable x_k with $k \in [n] \setminus \{j\}$ appears with probability $1/(n-1)$, the variable x_k appears in none of the m' equations with probability $\left(1 - \frac{1}{n-1}\right)^{m'}$, and by union bound all variables appear at least once with probability at least $1 - (n-1)\left(1 - \frac{1}{n-1}\right)^{m'}$. Therefore:

$$p_2 = \mathsf{B}\left(m', m, 2m_2/n\right) \left(1 - (n-1)\left(1 - \frac{1}{n-1}\right)^{m'}\right), \qquad \text{and } p_s = p_1 p_2.$$

□

To conclude this part, we give the relevant parameters of the 4 FLIP instances relatively to the criteria we described in this section, and the estimated complexity of the DFA attacks we described in Table 4, for an example of choice for m and m'. In all these cases we get that with high probability with less than 2^{20}

samples, the adversary determines the fault position and recovers the key. In Section 3.4, we test the attack in practice, and show that a lower number of samples is sufficient on average, reducing it from at least 2^{15} to at most 2^8.

instance n	$\mathbf{m}_f$	$\lvert\mathsf{Inf}^1(f)\rvert$	$\frac{1}{n}\sum_{j'=1}^{n}\mathsf{inf}_{j',\mathsf{E}_{\frac{n}{2},n}}(f)$	$\log(m)$	$\log(m')$	$p_s \geq$
530	$[50, 72, 8_7]$	50	0.259	15	13	0.999
662	$[50, 72, 4_{13}]$	50	0.196	16	13	0.997
1394	$[90, 120, 8_{14}]$	90	0.162	17	14	0.989
1704	$[91, 124, 5_{21}]$	91	0.132	18	15	0.999

Table 4: FLIP instances and DFA attacks

3.4 Concrete attack on one instance of FLIP

To illustrate our DFA on FLIP we consider one instance of FLIP and perform experiments. We consider FLIP with $n = 530$ and $\mathbf{m}_f = [50, 72, 8_7]$ and simulate the attack. We first collect m many normal keystream bits, then we inject a single bit fault at a random location of the key register and collect m many faulty keystream bits for the same key and IV. We use Algorithm 1 to determine the location of the injected fault. Depending upon the number of selected keystream bits m and the number of tries, Algorithm 1 returns a set with possible fault location (see Table 5). In practice we observed that $m = 2^8$ is sufficient for Algorithm 1 to return the correct location of the injected fault. After the location of the injected fault is identified, we run Algorithm 2 to recover the secret key. Experimentally, we observe that if we consider 20000 normal and faulty keystream bits then Algorithm 2 returns the correct secret key in approximately 32 seconds. The complete experiment was performed using SageMath software [Sag17] installed in a laptop with processor of 2.80GHz clock, 16 GB RAM and linux (Ubuntu 22.10) environment.

Table 5: Success rate of Algorithm 1 for FLIP_{530}

m	# tries	$\lvert L\rvert < 5$	$\lvert L\rvert = 1$
2^6	100	11	0
2^7	100	100	95
2^8	100	100	100

4 DFA attack on improved filter permutators

We generalize the approach of the previous section to improved filter permutators. Following the previous section's architecture, we first explain how a similar differential fault attack can be mounted even with a subset selection and a whitening. Then, we analyze the complexity of finding the error position in the case of IFP with a result analogous to Proposition 2, and we reduce the complexity of the attack to the one of IFP whose filter is a derivative of the initial filter. In Section 4.3, we focus on the instances of FiLIP: we study the derivatives of Xor-Threshold functions and investigate the attack complexity on FiLIP-DSM and FiLIP-XOR-THR. Finally, to illustrate this DFA strategy, we implement a concrete attack on an instance of FiLIP.

4.1 Generalizing the attack to IFP

In the following, we show that the DFA attack on FP can be generalized to IFP and analyzed similarly. Without the fault injection the i-th output of FiLIP can be written as $s_i = f(P_i(S_i(x)) + \omega_i)$, where (in contrast with FLIP) S_i is a n-out-of-N subset, and w_i a public vector from $\mathbb{F}_2^n$. Since P_i, S_i and ω_i are known, each keystream bit can be written as:

$$s_i = g(P_i'(x + \omega_i')),$$

where:

- $g \in \mathcal{B}_N$ is the direct sum of f and the null function on $N - n$ variables.
- $\tilde{P}_i$ is the N to N wire-cross permutation consisting in applying P_i over S_i to get the first n bits and mapping the $[N] \setminus S_i$ other indexes from $n+1$ to N.
- $\tilde{w_i}$ is the whitening vector of length N defined as $\tilde{w_i}(j) = w_i(P_i[S_i[j]])$ if $j \in S_i$ and 0 otherwise.

Then, when a fault has been injected in the j-th position of the key register, the i-th produced output can be written as $s_i' = g(\tilde{P}_i(x + \tilde{\omega_i} + \mathbf{e}_j))$ and therefore:

$$s"_i = s_i + s_i' = g(\tilde{P}_i(x + \tilde{\omega_i})) + g(\tilde{P}_i(x + \tilde{\omega_i} + \mathbf{e}_j)) = \mathsf{D}_{\tilde{P}_i(\mathbf{e}_j)} g(\tilde{P}_i(x + \tilde{\omega_i})).$$

Note that $g(x)$ and $g(x + \tilde{\omega_i})$ have the same cryptographic properties; hence we can apply the same attack as in Section 3, and in this case, the complexity of the attacks depend on the properties of $\mathsf{D}_{\mathbf{e}_{j'}} g$.

4.2 DFA on improved filter permutators and filter properties

Determining the fault position

Proposition 6. *Let g be an N-variable function with influence sets $\mathsf{Inf}^0(g)$ and $\mathsf{Inf}^1(g)$, a DFA adversary can recover the fault position $j \in [N]$ of FiLIP$_f$ using m keystream bits $s"_i$ with the time complexity $\mathcal{O}(mN)$, and the success probability $Pr[L = \{j\}]$.*

- *If $0 < \ell = \min(|\mathsf{Inf}^0(g)|, |\mathsf{Inf}^1(g)|)$ then $Pr[L = \{j\}] \geq 1 - (N-1)\left(1 - \frac{\ell}{N-1}\right)^m$.*
- *If $|\mathsf{Inf}^\varepsilon(g)| \neq 0$ and $|\mathsf{Inf}^{1-\varepsilon}(g)| = 0$ for $\varepsilon \in \{0,1\}$ then:*

$$Pr[L = \{j\}] \geq \mathsf{B}(m', m, p)\left(1 - (N-1)\left(1 - \frac{\ell}{N-1}\right)^{m'}\right),$$

where $m' \in [m]$ and $p = \varepsilon + (1-2\varepsilon)\frac{1}{N}\sum_{j'=1}^{N} \mathsf{inf}_{j'}(f)$.

Proof. The proof is very similar to the one of Proposition 2, seen over g (the N-variable function) rather than f, the only difference is the addition of a whitening ω_i which randomizes the input of f (the n-variable function).

We begin with the case where $\mathsf{Inf}^0(g)$ and $\mathsf{Inf}^1(g)$ are both not empty. Denoting $\ell = \min(|\mathsf{Inf}^0(g)|, |\mathsf{Inf}^1(g))|$, each equation alone allows removing at least ℓ possible positions for j. Since the permutations $\tilde{P}_i$ are uniformly distributed, we can estimate the number of equations needed before having the set L containing only j using the "balls-in-bins" approach. Each equation gives at least ℓ random positions of $[N] \setminus \{j\}$, then the probability for one of the $N-1$ positions (seen as bins), called j', to have not been removed after m equations (seen as m launches of at least ℓ balls) is: $\Pr\left[j' \in L\right] \leq \left(1 - \frac{\ell}{N-1}\right)^m$. Having only j in L after m equations corresponds to having $N-1$ other positions appearing in at least one equation, hence by union bound after m equations $\Pr\left[L = \{j\}\right] \geq 1 - (N-1)\left(1 - \frac{\ell}{N-1}\right)^m$.

When only one of the influence sets is non-null, only one part of the equations allows removing potential positions. In the case $\mathsf{Inf}^1(g) \neq \emptyset$ (respectively $\mathsf{Inf}^0(g) \neq \emptyset$) only the equations such that $s"_i = 0$ (respectively $s"_i = 1$) allow to decrease L in Algorithm 1. Hence, we can study the probability of the algorithm returning j similarly as before by pondering it by the probability $\Pr\left[s"_i = 1\right]$. More precisely, the probability of getting at least m' over m equations such that $s"_i = 0$ (respectively $s"_i = 1$) is given by a binomial law with parameter $p = 1 - \Pr\left[s"_i = 1\right]$ (respectively $(p = \Pr\left[s"_i = 1\right])$. Hence we compute $\Pr\left[s"_i = 1\right]$.

We know that $s"_i = \mathsf{D}_{\tilde{P}_i(\mathbf{e}_j)} g(\tilde{P}_i(x + \tilde{\omega_i}))$ therefore since P_i, S_i and ω_i are uniformly distributed:

$$\begin{aligned}
\Pr\left[s"_i = 1\right] &= \Pr\left[g(\tilde{P}_i(x + \tilde{\omega_i}) \neq g(\tilde{P}_i(x + \tilde{\omega_i} + \mathbf{e}_j)\right] \\
&= \Pr\left[g(\tilde{P}_i(x + \tilde{\omega_i}) \neq g(\tilde{P}_i(x + \tilde{\omega_i} + \mathbf{e}_j) \mid \tilde{P}_i(j) \in S_i\right] \cdot \Pr\left[\tilde{P}_i(j) \in S_i\right] \\
&\quad + \Pr\left[g(\tilde{P}_i(x + \tilde{\omega_i}) \neq g(\tilde{P}_i(x + \tilde{\omega_i} + \mathbf{e}_j) \mid \tilde{P}_i(j) \notin S_i\right] \cdot \Pr\left[\tilde{P}_i(j) \notin S_i\right] \\
&= \frac{n}{N} \Pr\left[f(S_i(\tilde{P}_i(x + \omega_i))) \neq f(S_i(\tilde{P}_i(x + \tilde{\omega_i} + \mathbf{e}_j)))\right] + 0,
\end{aligned}$$

where the last equation comes from the fact that S_i is taken uniformly at random from the set of n-out-of-N subsets, and the $N-n$ last inputs of g have no influence. Then, denoting by x' the restriction of x over S_i, $\Pr\left[s"_i = 1\right]$

$$\begin{aligned}
&= \frac{n}{N} \sum_{j'=1}^{n} \Pr\left[f(S_i(\tilde{P}_i(x + \omega_i))) \neq f(S_i(\tilde{P}_i(x + \tilde{\omega_i} + \mathbf{e}_j))) \mid S_i(\tilde{P}_i(j)) = j'\right] \cdot \Pr\left[S_i(\tilde{P}_i(j)) = j'\right] \\
&= \frac{1}{N} \sum_{j'=1}^{n} \Pr\left[f(P_i)(x' + \omega_i) \neq f(P_i)(x' + \omega_i + \mathbf{e}'_j)\right] = \frac{1}{N} \sum_{j'=1}^{n} \mathsf{inf}_{j'}(f) = p_g,
\end{aligned}$$

where the second equation comes from the fact that both S_i and P_i are taken uniformly at random. The last equation comes from the fact that the influence of a variable is invariant to the addition of a constant, $\forall a \in \mathbb{F}_2^n, \forall i \in [n]$:

$$\begin{aligned}
\mathsf{inf}_i(f(x)) &= \frac{\#\{x \in \mathbb{F}_2^n \mid f(x) \neq f(x + \mathbf{e}_i)\}}{2^n}, \\
&= \frac{\#\{x \in \mathbb{F}_2^n \mid f(x + a) \neq f(x + a + \mathbf{e}_i)\}}{2^n} = \mathsf{inf}_i(f(x + a)).
\end{aligned}$$

Hence, in the case $\mathsf{Inf}^1(g) \neq \emptyset$ (respectively $\mathsf{Inf}^0(g) \neq \emptyset$), the algorithm recovers j with probability:

$$\Pr\left[L = \{j\}\right] \geq \mathsf{B}\left(m', m, p\right)\left(1 - (N-1)\left(1 - \frac{\ell}{N-1}\right)^{m'}\right),$$

where $m' \in [m]$, and $p = 1 - p_g$ (respectively $p = p_g$).

We finish with the time complexity of Algorithm 1. Since for each equation used by the algorithm it computes at most $\max(|\mathsf{Inf}^0(g)|, |\mathsf{Inf}^1(g)|)$ indexes to remove from L, its time complexity is (roughly) upper bounded by $\mathcal{O}(Nm)$. □

Remark 1. All instances of IFP use $N > n$, then $\mathsf{Inf}^0(g)$ is not empty, it contains at least $N - n$ elements. Moreover, for all filter functions f that are the direct sum of an XOR part and another function such as all candidate instances [MCJS19a,MCJS19b,HMR20,CDPP22,MPP23], the XOR_k part provides already k elements in $\mathsf{Inf}^1(g)$. Accordingly, for such filter functions the first case of Proposition 6 always applies, both 0-influence set and 1-influence set can be used.

Attack on $\mathsf{D}_{\mathbf{e}_{j'}}g$. Using Algorithm 1 and Proposition 6 an adversary can determine the fault position and then perform an attack on the system given by $\mathsf{D}_{\mathbf{e}_{j'}}g$, using the general approach described in Section 3.2 adapted to the IFP. It gives a time complexity of $\mathcal{O}(T_1 + T_2)$, where T_1 is the complexity of determining the position j of the fault, and T_2 can be taken as the complexity of the best attack against FiLIP with filter function $\mathsf{D}_{\mathbf{e}'_j}g$.

4.3 Differential properties of Xor-Threshold and attack on FiLIP

Derivative of Xor-Threshold functions FiLIP instances use filter functions which are DSM or Xor-Threshold functions, since the influence properties of DSM are already given in Proposition 4, we study the ones of Xor-Threshold in the following. The standard cryptographic properties (algebraic immunity, nonlinearity, resilience, $\cdots$) of these functions were studied in [CM22].

Proposition 7. *Let $f \in \mathcal{B}_n$ a Xor-Threshold $\mathsf{XOR}_k + \mathsf{T}_{d,m}$ and $i \in [n]$, the following holds on $\mathsf{D}_{\mathbf{e}_i}f$:*

- *If x_i does not appear in the ANF of f, then $\mathsf{D}_{\mathbf{e}_i}f$ is the constant function 0.*
- *If x_i appears in the XOR_k part then $\mathsf{D}_{\mathbf{e}_i}f$ is the constant function 1.*
- *If x_i appears in the threshold part $\mathsf{T}_{d,m}$ of f then $\mathsf{D}_{\mathbf{e}_i}f$ is a the indicator function of the set $\mathsf{E}_{d-1,m-1}$.*

Proof. The first and second cases are the same as in Proposition 3. We focus on the third case. In this case, we write x as (y, z) to split the part y in the xor and the part z in the threshold and write $x + \mathbf{e}_i$ as $(y, z + \mathbf{e}_j)$. Thereafter $f(x + \mathbf{e}_i) = x + \mathsf{T}_{d,m}(z + \mathbf{e}_j)$ and $f(x + \mathbf{e}_i)$ differs from $f(x)$ if and only if $\mathsf{T}_{d,m}(z + \mathbf{e}_j) \neq \mathsf{T}_{d,m}(z)$. Since the Hamming weights of z and $z + \mathbf{e}_j$ have a difference of 1, the threshold function gives a different output if and only if the Hamming weight of z without the j-th variable is $d - 1$. Thereafter $\mathsf{D}_{\mathbf{e}_i}f(y, z) = \mathbb{1}_{\mathsf{E}_{d-1,m-1}}(z_1, \ldots, z_{j-1}, z_{j+1}, \ldots, z_m)$, where $\mathbb{1}_{\mathsf{E}_{d-1,m-1}}$ denotes the characteristic function of the set $\mathsf{E}_{d-1,m-1}$.

□

Proposition 8. *Let $n \in \mathbb{N}^*$ and $f \in \mathcal{B}_n$ be a Xor-Threshold function $\mathsf{XOR}_k + \mathsf{T}_{d,m}$ with $2 \leq d \leq m$:*

- *$|\mathsf{Inf}^0(f)| = n - k - m$.*
- *$|\mathsf{Inf}^1(f)| = k$.*

Proof. As shown in Proposition 4's proof, $\mathsf{inf}_i(f) = 0$ if and only if $\mathsf{D}_{\mathbf{e}_i}f$ is the null function, and $\mathsf{inf}_i(f) = 1$ if and only if $\mathsf{D}_{\mathbf{e}_i}f$ is the constant function 1. Using Proposition 7 $\mathsf{D}_{\mathbf{e}_i}f$ is null when x_i does not appear in the ANF of f (the $n-m-k$ mute variables), $\mathsf{D}_{\mathbf{e}_i}f$ is the constant function when x_i appears in the XOR_k part. Since $2 \leq d \leq m$, the function $\mathbb{1}_{\mathsf{E}_{d-1,m-1}}$ is not constant, therefore we can conclude $|\mathsf{Inf}^0(f)| = n - k - m$ and $|\mathsf{Inf}^1(f)| = k$.

□

The attack strategy of Section 3.3 naturally extends to FiLIP-DSM instances, in the following we describe one strategy tailored to FiLIP-XOR-THR based on the derivatives of these functions. First the fault position j is determined with Algorithm 1 using that both $|\mathsf{Inf}^1(g)| \neq 0$ and $|\mathsf{Inf}^0(g)| \neq 0$ for these filters (Proposition 8). Then, we consider the following attack strategy S2 which takes in consideration that for a Xor-Threshold function $\mathsf{D}_{\mathbf{e}_i} f$ is often constant and gives few usable equations to determine the key variables.

Strategy S2: The attacker collects the equations where $\mathsf{D}_{\tilde{P}_i(\mathbf{e}_j)} g(\tilde{P}_i(x + \tilde{\omega_i}))$ is the indicator of an hypercube slice $\mathsf{E}_{k,n'}$ and $s"_i = 1$. These equations correspond to the cases where the $n' + 1$ variables with the associated whitening in input to the threshold function, minus the faulted position, give a vector of Hamming weight exactly k. For one such equation, renaming these variables $y_1, \ldots, y_{n'}$ and the associated (public) whitening $z_1, \ldots, z_{n'}$ the attacker can derive the linear equation $\sum_{i=1}^{n'} y_i + z_i = (k \mod 2)$. Then, the attacker collects (at least) $N-1$ such equations and solves the linear system. Finally, the faulted variable or a few undetermined variables, if the system is redundant, are recovered by taking equations given by s_i where these variables appear in the XOR part.

Proposition 9. *Let f be an n-variable function $\mathsf{XOR}_k + \mathsf{T}_{d,n'}$. Assuming recovering a linear system of $N-1$ equations with m samples in the first phase is sufficient to invert the remaining system in the second phase, a DFA adversary can recover the key of FiLIP_f using $m \geq N$ keystream bits $s"_i$ with complexity $\mathcal{O}(m^\omega)$, where ω is the exponent for the linear system exponent ($2 < \omega \leq 3$), and probability of success $p_s = p_1 p_2$, where p_1 is the probability of success of Algorithm 1 and*

$$p_2 = \mathsf{B}\left(N-1, m, \frac{n'\binom{n'-1}{d-1}}{N 2^{n'-1}}\right),$$

with $m' \in [m]$.

Proof. The time complexity and success probability are obtained by analyzing the complexity of Algorithm 1 and then Algorithm 3. The time complexity and success probability for Algorithm 1 come from Proposition 2. We denote it by T_1 and p_1 in the rest of the proof.

The cost of Algorithm 1 apart, the time complexity of Algorithm 3 is dominated by the linear system inversions. The two while loops have at most m iterations, doing one test and deriving one linear equation in the first loop, doing up to $t \leq N$ tests and deriving one linear equation in the second loop. Since the cost of solving the two linear systems with at most m equations in (at most) N binary variables is $\mathcal{O}(m^\omega)$. It dominates over the cost of the two loops and the cost of Algorithm 1 which is $T_1 = \mathcal{O}(mN)$.

The success probability can be estimated as $p_s = p_1 p_2$ where p_2 refers to the probability of determining entirely K provided j. We assume that the probability p_2 is the same as the probability of obtaining the $N-1$ equations; that is, we consider that if the attacker gets a linear system of $N-1$ equations with m samples in the first loop, then inverting this system and performing the second loop with the m samples to find the last variables always succeed. Thereafter, we bound the probability of getting the $N-1$ linear equations. Since S_i and P_i are uniformly distributed, the probability that $P_i(S_i(j))$ belongs to $[k+1, n]$ is n'/N, and then since w_i is uniformly distributed, the probability that the input of the threshold function without x_j has Hamming weight $d-1$ is $\binom{n'-1}{d-1}/2^{n'-1}$. Finally, the probability that at least $N-1$ equations over m satisfy the condition is:

$$\mathsf{B}\left(N-1, m, \frac{n'\binom{n'-1}{d-1}}{N 2^{n'-1}}\right).$$

□

```
Input: Keystream size $m$, binary stream $s"_i$ for $i \in [m]$, subsets $S_i$, permutations $P_i$ and whitening
    $\omega_i$ for $i \in [m]$, $n$, $N$ and the parameters $k, d, n'$ of the Xor-Threshold function $f$.
Output: key K.
K := $[x_1, \ldots, x_N]$ ; // key with undetermined values
j := Algorithm 1 on inputs ($m$, $s"_i$ for $i \in [m]$, $\tilde{P}_i$ for $i \in [m]$, $[k]$, $[n+1, N]$) ;
i := 1 ;
cpt := 0 // the counter of linear equation produced
while $cpt < N - 1$ & $i \leq m$ do
    // Tests if $\tilde{P}_i(j)$ falls in the threshold part and $s"i = 1$
    if $j \in S_i$, $P_i(S_i(j)) \in [k+1, n]$ & $s_i" = 1$ then
        Stores the linear equation $\sum_{\ell=1}^{n-1} y_\ell + z_\ell = d - 1 \mod 2$ // where $y_\ell$ and $z_\ell$ are the
          $n' - 1$ variables different from the fault entering the threshold with
          $x_j$ and $z_\ell$ their associated whitening
    end
    i ← i+1 ;
end
Solve the linear system ;
Update $K$ ;
L := [$t \in K$ such that $x_t$ is still undetermined] ;
$\ell$ := 1 ;
while $\ell \leq i$ do
    // Keep the linear equations where no undetermined variables are in the
      threshold part
    if $\forall t \in L$, $\tilde{P}_i(t) \notin [k+1, n]$ then
        // Since all the variables in the threshold part are known, the
          remaining equation is a linear equation with variables from $L$ only.
        Store the linear equation obtained by evaluating $s_\ell$.
    end
    $\ell$ ← $\ell$+1 ;
end
Solve the linear system ;
return K ;
```

Algorithm 3: Attack strategy S2.

We finish this part by estimating the probability of success of Strategy S2 on the different instances of FiLIP with Xor-Threshold functions (Table 3). As in Proposition 9 we assume that the $N-1$ linear equations obtained are independent and allow to recover the full key. The estimated probabilities of success are presented in Table 6, for an example of choice for m. Note that if the $N-1$ linear equations are not sufficient to recover the full key, we can collect $m' > N-1$ linear equations; it only impacts p_2 for the success probability where the "$N-1$" term is changed by "m'". In these cases, with a high probability using less than 2^{20} samples, the adversary recovers the fault position and determines the key.

instance N	k, d, n'	$\lvert\mathsf{Inf}^0(f)\rvert$	$\lvert\mathsf{Inf}^1(f)\rvert$	$\log(m)$	$p_s \geq$
2048	$113, 16, 31$	1904	113	20	0.999
2048	$81, 32, 63$	1904	81	20	0.999
1536	$129, 64, 127$	1280	129	19	0.999
1024	$257, 128, 255$	512	257	17	0.999
8192	$129, 64, 127$	7936	129	23	0.999
4096	$257, 128, 255$	3584	257	21	0.999

Table 6: FiLIP instances and DFA attacks

4.4 Concrete attack on one instance of FiLIP

To illustrate our DFA on FiLIP we consider one instance of FiLIP, the one with $N = 1024$ (the smallest value of N for the instances from this family) and filter $\mathsf{XOR}_{257} + \mathsf{T}_{128,255}$ (See Table 3). We simulate the DFA attack by first collecting m normal keystream bits, and then collecting m faulted keystream bits for the same key and IV after flipping one of the key bits. Then, we use Algorithm 1 to determine the location of the fault. We display in Table 7 the size of L we observed depending on the value of m. For this filter, in practice, $m = 2^5$ is sufficient to detect the position of the fault. Then, we simulate the second part of the DFA with a simplified version of Algorithm 3, In the simulation, the faulty bit is always selected in the filter, and we use an SAT-solver that has already been implemented to solve the linear system. The experiment using SageMath software [Sag17] and $m = 45000$ took approximately 290 seconds on a laptop with processor of 2.80 GHz clock, 16 GB RAM and linux (Ubuntu 22.10) environment. For this filter, the probability of the faulty bit to be selected is $\frac{512}{1024} = 0.5$. Hence, we expect a timing of approximately 580 seconds for Algorithm 3.

Table 7: Success rate of Algorithm 1 for FiLIP with filter $\mathsf{XOR}_{257} + \mathsf{T}_{128,255}$

m	# tries	$\lvert L\rvert < 5$	$\lvert L\rvert = 1$
16	100	84	19
24	100	100	93
32	100	100	100

5 Conclusion and open questions

In this article we studied new theoretical differential fault attacks on the stream ciphers FLIP and FiLIP considering the fault model where one random bit of the key is flipped before the keystream generation. We presented DFA strategies for both ciphers which time and data complexity can be bounded using cryptographic parameters of the derivatives (in the direction of Hamming weight one vectors) of the filter function. More precisely, the sensitivity notions are used to detect the fault position with few keystream bits, improving upon the DFA presented in [RBM20] and [RKMR23]. Then, the different black-box attacks described on the filter permutator model (for FLIP) and improved filter permutator model (for FiLIP) on a filter f can be applied to the derivative of the filter in our context. Based on the properties of the derivatives of DSM and Xor-Threshold functions, we proposed specialized algorithms for the different instances of FLIP and FiLIP. Finally, we illustrated the practicability of our theoretical attacks with an example on FLIP-530 and FiLIP with filter $\mathsf{XOR}_{257} + \mathsf{T}_{128,255}$.

Two main open questions arise from this work:

- **DFA countermeasure**. First, the complexities of the attack directly come from (standard) cryptographic parameters of derivatives of the filter function. One interesting direction to avoid the proposed DFA on FLIP and FiLIP is, therefore, to find Boolean functions with strong derivatives. Since the functions used for these ciphers are already subject to different constraints, such as being cheap to evaluate homomorphically and having good parameters even after fixing a limited number of variables, finding an appropriate function satisfying this additional requirement is an engaging challenge. Alternatively, finding functions with a small 0-influence set and a small 1-influence set will make the fault localization part of the attack more costly.
- **Fault model.** Second, the fault model we considered flips only one bit; a natural generalization consists of studying the attacks where the injected fault flips t bits, with t bounded but potentially unknown. In this case, the attacks we presented could be generalized, relying on the properties of other derivatives of the filter function. For example, if t is sufficiently small, the derivatives of f in vectors of Hamming weight at most t could still be studied, potentially leading to successful differential fault attacks for this more general model.

6 Acknowledgments

Pierrick Méaux was supported by the ERC Advanced Grant no. 787390. Dibyendu Roy would like to acknowledge the grant from the project entitled "Design and Analysis of Fully Homomorphic Encryption (FHE) Supported Stream Ciphers" with project number SRG/2021/000196 awarded by Science & Engineering Research Board (SERB), Department of Science & Technology, Govt. of India.

References

Bab08. Matthew Babbage, Steveand Dodd. *The MICKEY Stream Ciphers*, pages 191–209. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

BDL97. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

BM13. Subhadeep Banik and Subhamoy Maitra. A differential fault attack on mickey 2.0. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013*, pages 215–232, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

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

BY03. Mihir Bellare and Bennet Yee. Forward-security in private-key cryptography. In Marc Joye, editor, *Topics in Cryptology — CT-RSA 2003*, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

Car21. Claude Carlet. *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press, 2021.

CCF$^+$16. Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, 2016.

CDPP22. Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder V. L. Pereira. Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 563–577. ACM, 2022.

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

CM22. Claude Carlet and Pierrick Méaux. A complete study of two classes of boolean functions: direct sums of monomials and threshold functions. *IEEE Transactions on Information Theory*, 68(5):3404–3425, 2022.

CMR17. Claude Carlet, Pierrick Méaux, and Yann Rotella. Boolean functions with restricted input and their robustness; application to the FLIP cipher. *IACR Trans. Symmetric Cryptol.*, 2017(3), 2017.

Cou03a. Nicolas Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 176–194. Springer, Heidelberg, August 2003.

Cou03b. Nicolas T. Courtois. Higher order correlation attacks, xl algorithm and cryptanalysis of toyocrypt. In Pil Joong Lee and Chae Hoon Lim, editors, *Information Security and Cryptology — ICISC 2002*, pages 182–199, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

DEG$^+$18. Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A cipher with low anddepth and few ands per bit. In *CRYPTO 2018*, pages 662–692, 2018.

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

HJMM08. Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. The grain family of stream ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 2008.

HKM17. Matthias Hamann, Matthias Krause, and Willi Meier. LIZARD - A lightweight stream cipher for power-constrained devices. *IACR Trans. Symmetric Cryptol.*, 2017(1):45–79, 2017.

HMR20. Clément Hoffmann, Pierrick Méaux, and Thomas Ricosset. Transciphering, using filip and TFHE for an efficient delegation of computation. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *Progress in Cryptology - INDOCRYPT 2020 - 21st International Conference on Cryptology in India, Bangalore, India, December 13-16, 2020, Proceedings*, volume 12578 of *Lecture Notes in Computer Science*, pages 39–61. Springer, 2020.

Hon16. Wu Hongjun. A lightweight authenticated cipher. https://competitions.cr.yp.to/round3/acornv3.pdf, 2016.

HS04. Jonathan J. Hoch and Adi Shamir. Fault analysis of stream ciphers. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, pages 240–253, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

KJJ99. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

KKL88. J. Kahn, G. Kalai, and N. Linial. The influence of variables on boolean functions. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 68–80, 1988.

Koc96. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

MAM17. Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On ciphers that continuously access the non-volatile key. *IACR Transactions on Symmetric Cryptology*, 2016(2):52–79, Feb. 2017.

MCJS19a. P. Méaux, C. Carlet, A. Journault, and F. Standaert. Improved filter permutators for efficient FHE: better instances and implementations. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *INDOCRYPT*, volume 11898 of *LNCS*, pages 68–91, 2019.

MCJS19b. Pierrick Méaux, Claude Carlet, Anthony Journault, and François-Xavier Standaert. Improved filter permutators: Combining symmetric encryption design, boolean functions, low complexity cryptography, and homomorphic encryption, for private delegation of computations. Cryptology ePrint Archive, Report 2019/483, 2019.

MJSC16. Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards stream ciphers for efficient fhe with low-noise ciphertexts. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 311–343, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

MPP23. Pierrick Méaux, Jeongeun Park, and Hilder V. L. Pereira. Towards practical transciphering for FHE with setup independent of the plaintext space. *IACR Cryptol. ePrint Arch.*, page 1531, 2023.

MSS17. S. Maitra, A. Siddhanti, and S. Sarkar. A differential fault attack on plantlet. *IEEE Transactions on Computers*, 66(10):1804–1808, 2017.

NLV11. Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, page 113–124, New York, NY, USA, 2011. Association for Computing Machinery.

RBM20. Dibyendu Roy, Bhagwan Bathe, and Subhamoy Maitra. Differential fault attack on kreyvium flip. *IEEE Transactions on Computers*, 2020.

RKMR23. R Radheshwar, Meenakshi Kansal, Pierrick Méaux, and Dibyendu Roy. Differential fault attack on rasta and filip-dsm. *IEEE Transactions on Computers*, pages 1–8, 2023.

Rob08. Matthew Robshaw. *The eSTREAM Project*, pages 1–6. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

Sag17. Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.1)*, 2017. `https://www.sagemath.org`.

Sie84. Thomas Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE*, IT-30(5):776–780, 1984.

SSMC17. Akhilesh Siddhanti, Santanu Sarkar, Subhamoy Maitra, and Anupam Chattopadhyay. Differential fault attack on grain v1, ACORN v3 and lizard. In Sk Subidh Ali, Jean-Luc Danger, and Thomas Eisenbarth, editors, *Security, Privacy, and Applied Cryptography Engineering - 7th International Conference, SPACE 2017, Goa, India, December 13-17, 2017, Proceedings*, volume 10662 of *Lecture Notes in Computer Science*, pages 247–263. Springer, 2017.