TooLIP: How to Find New Instances of FiLIP Cipher With Smaller Key Size and New Filters

François Gérard[1], Agnese Gini[1], Pierrick Méaux[1]

University of Luxembourg, Esch-sur-Alzette, Luxembourg,
name.surname@uni.lu

Abstract. In this article, we propose a new tool to evaluate the security of instances of FiLIP cipher. TooLIP is user friendly, it automatically evaluates the cost of several attacks on user-defined Boolean functions. It allows to test new families of filters that are more homomorphic friendly for recent techniques of evaluations, and is designed to easily add new attacks, or change parameters in the considered attacks. To demonstrate our tool we apply it in three contexts. First we show how the keysize can be reduced for former instances with XOR-Threshold functions when the amount of encrypted plaintext obtained by the adversary is limited. Then, we use TooLIP to determine secure instances with filters in less variables for two new families of Boolean functions, leading to a more efficient evaluation and/or a reduced bandwidth. Finally, we apply it to find other instances with filters where we know only (bounds on) the algebraic immunity and resiliency.

Keywords: Automatic tool, Stream ciphers, FiLIP cipher, Boolean functions.

1 Introduction

With the growing interest of practical Fully Homomorphic Encryption (FHE), in the last decade multiple symmetric ciphers have been designed to be efficient in the context of Hybrid Homomorphic Encryption [NLV11] (HHE). After considering standard symmetric ciphers such as AES [GHS12,CLT14] or SIMON [LN14], dedicated symmetric schemes have been introduced in the context of HHE since traditional ciphers are hard to efficiently evaluate homomorphically. For example, the first of these ciphers, LowMC [ARS+15] has a drastically reduced number of multiplications to be more FHE-friendly. The cipher FLIP [MJSC16] introduces a new stream cipher paradigm where a significant part of the cipher evaluation can be performed without homomorphic cost. This strategy has been generalized with extendable output functions and is common in recent ciphers for HHE such as in *e.g.* [DEG+18,CHK+21,HKL+22,HKC+20,DGH+21,CHMS22,HMS23]. All these designs for HHE are still recent and improvable, as witnessed recently with improvements on building blocks of the symmetric scheme [HL20,CIR22] or new techniques of homomorphic evaluation [CDPP22,MPP23].

In this article we focus on FiLIP, introduced in [MCJS19b]. FiLIP is a binary stream cipher following the improved filter permutator paradigm, extending the paradigm of FLIP. The main advantage of this cipher is that both its security and performance when it is homomorphically evaluated depend on a sole Boolean function, the one used as filter. On the homomorphic side, it allows to obtain a small latency in the context of HHE, as experimented in [MCJS19b], and more recently in [HMR20] and [CDPP22] reaching a latency of only 2.62ms, using the multiples progresses on the so-called third generation of FHE schemes such as FHEW [DM15], TFHE [CGGI16] and FINAL [BIP+22]. On the symmetric security side, in addition to the security analysis relying on the cryptographic criteria of the filter, the security of the filter permutator has been studied in an idealized model in [CT19]. Moreover, the design is similar to a popular pseudorandom generator design by Goldreich [Gol00] which asymptotic security has been thoroughly investigated *e.g.* [AL16,AL18,CDM+18,YGJL22,Üna23].

In [MCJS19b], FiLIP is instantiated using two families of functions, *i.e.* direct sum of monomials and XOR-Threshold functions. The security of these instances is determined by a procedure bounding the complexity of different attacks via cryptographic criteria (namely, the algebraic immunity, resiliency, and nonlinearity). Producing an efficient algorithmic implementation of this procedure is not trivial, because it requires to evaluate the parameters of all the sub-functions that are obtained by fixing up to λ variables[1], together with their probability. In fact, the authors needed to introduce specific optimisations to compute the suggested parameters for aforementioned families. Therefore, a major bottleneck for finding alternative instances of FiLIP is evaluating their security. Our work aims to facilitate this task by introducing a tool designed exactly for this purpose.

[1] λ being the security parameter.

Having an automated tool to assess the security of constructions is always a plus to help the community in broadening the field of cryptography. Indeed, when designers of schemes, protocols, or any cryptographic application propose something new, they often face the challenge of finding secure parameters. While sometimes, an ad-hoc security evaluation has to be performed due to the specificity of the proposal, it is often the case that the new construction simply relies on well-known assumptions or designs with unusual parameters. Then, it greatly simplifies the task of the designers if they can solely specify those parameters to a tool and get a security estimation. A notable example in the field of lattice-based cryptography is the estimator first proposed in [APS15a]. Not only a common tool helps for simplicity, but it also importantly sets a common ground for everyone. Indeed, estimating the cost of an attack for an adversary is often made under specific assumptions on parameters of the attack, this might lead to unfair comparisons between works that were both analyzed in good faith. Because attacks, as well as the accuracy of their bounds, might evolve over time, a unified tool is often a collaborative effort and should also evolve, this is why the current ePrint version of [APS15a]([APS15b]), starts with a disclaimer to warn the reader that the state of the art advanced since its initial publication.

Our contributions. In this article, we propose straightforward Python scripts to estimate the security of FiLIP when used with a given filter function and register size. The code has been written with user-friendliness in mind. The goal is that the user who wishes to study a new function can simply write a new Python class with a specific interface evaluate the cost of well-known attacks. This new piece of code must provide methods that compute Boolean criteria such as algebraic immunity, resiliency, and nonlinearity and also what are the direct descendants of the function obtained when fixing an input variable to 0 or 1. While testing new functions on fixed attacks is probably the simplest use case of the tool, it is also possible to implement new attacks or to modify the existing one by changing parameters such as the number of ciphertexts that the adversary can observe. Since we put emphasis on user-friendliness and re-usability, the scripts are written in pure Python. This naturally comes with a performance penalty. However, this has not been problematic for the functions we tested and should not be a problem unless the security parameter is severely increased. Since the tool is quite modular, it should be possible to rewrite and integrate optimized code for some parts if efficiency is the bottleneck.

More precisely our contributions are the following. First, based on [MCJS19a], we describe a general framework in which the security of FiLIP can be bounded for a given filter function and register size N. Then, we propose a user-friendly tool called TooLIP that automatically runs several attacks on user-defined functions. The tool can be used to test new families of filters that are more homomorphic friendly for recent techniques of evaluations. To demonstrate the applicability of the tool, we use it in three contexts:

- First, we consider the improvements on XOR-THR filters, which are studied in [MCJS19a] and used in the context of HHE in [HMR20] and [CDPP22]. We show that for the same functions the register size N can be significantly reduced when we take into consideration a limit on the quantity of encrypted plaintext ($2^{\lambda/2}$ for our examples). A reduced N leads to bandwidth improvements in HHE. Then, we exhibit filters with fewer variables in the same family for the same security level, which would lead to better HHE general performances.
- We determine secure instances for two new families of filters, XOR-QUAD-THR and XOR-THR-THR functions. First we investigate the Boolean criteria of these two families, then we exhibit new instances of FiLIP with smaller register sizes as for XOR-THR functions or direct sum of monomials as used in [MCJS19b]. For example, we propose several instances with N lower than 500 for 80-bit security and lower than 1000 for 128-bit security.
- Finally, we investigate instances with arbitrary filters: functions in n variables such that the main cryptographic parameters are known but not the specific structure of the functions nor its descendants. For example we show that a 512-variable function with optimal algebraic immunity in direct sum with a 196-variable XOR function is sufficient for 128-bit security since $N \geq 2500$.

The code is available here[2]. It will be made public once the paper publicly appears.

2 Preliminaries

Additionally to usual notations we denote the set of integers $\{1, \ldots, n\}$ by $[1, n]$, and use $+$ instead of $\oplus$ for the addition in $\mathbb{F}_2$.

[2] https://drive.google.com/file/d/1FhySzEpBFNWE8S6-edZsiamTss1I22pd/view?usp=sharing

2.1 Boolean functions and cryptographic criteria

We give some preliminaries on Boolean functions and their cryptographic criteria, for a deeper introduction on these notions we refer to the book of Carlet [Car21].

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

Definition 2 (Balancedness and Resiliency). *A Boolean function $f \in \mathcal{B}_n$ is said to be balanced if $|f^{-1}(0)| = |f^{-1}(1)| = 2^{n-1}$. The function f is called k-resilient if any of its restrictions obtained by fixing at most k of its coordinates is balanced. We denote by $\mathsf{res}(f)$ the maximum resiliency (also called resiliency order) of f and set $\mathsf{res}(f) = -1$ if f is unbalanced.*

Definition 3 (Nonlinearity). *The nonlinearity $\mathsf{NL}(f)$ of a Boolean function $f \in \mathcal{B}_n$, where n is a positive integer, is the minimum Hamming distance (d_H) between f and all the affine functions in $\mathcal{B}_n$: $\mathsf{NL}(f) = \min_{g,\,\mathsf{deg}(g)\leq 1}\{\mathsf{d_H}(f,g)\}$, where $g(x) = a \cdot x + \varepsilon$, $a \in \mathbb{F}_2^n, \varepsilon \in \mathbb{F}_2$ (where $\cdot$ is an inner product in $\mathbb{F}_2^n$, any choice of inner product will give the same value of $\mathsf{NL}(f)$).*

Definition 4 (Algebraic Normal Form (ANF), degree and monomials). *We call Algebraic Normal Form of a Boolean function f its n-variable polynomial representation over $\mathbb{F}_2$ (i.e. belonging to $\mathbb{F}_2[x_1,\ldots,x_n]/(x_1^2+x_1,\ldots,x_n^2+x_n)$): $f(x_1,\ldots,x_n) = \sum_{I\subseteq[1,n]} a_I \left(\prod_{i\in I} x_i\right)$ where $a_I \in \mathbb{F}_2$.*

The (algebraic) degree of a non-zero function f is $\mathsf{deg}(f) = \max_{I\subseteq[1,n]}\{|I| \,|\, a_I = 1\}$. If $f = 0$, $\mathsf{deg}(f) = 0$.

Each term $\prod_{i\in I} x_i$ is called a monomial, and we refer to monomial functions as the ones having only one coefficient $a_I = 1$ in their ANF.

Definition 5 (Algebraic Immunity [MPC04], annihilators and $\Delta_{\mathsf{AN}}(f)$). *The algebraic immunity of a Boolean function $f \in \mathcal{B}_n$, denoted as $\mathsf{AI}(f)$, is defined as: $\mathsf{AI}(f) = \min_{g\neq 0}\{\mathsf{deg}(g) \mid fg = 0 \text{ or } (f+1)g = 0\}$.*

The function g is called an annihilator of f (or $f+1$). Additionally we denote $\mathsf{AN}(f) = \min_{g\neq 0}\{\mathsf{deg}(g) \mid fg = 0\}$, and $\Delta_{\mathsf{AN}}(f) = |\mathsf{AN}(f) - \mathsf{AN}(f+1)|$.

Definition 6 (Fast Algebraic Immunity, *e.g.* [Car21] page 94). *The fast algebraic immunity of a Boolean function $f \in \mathcal{B}_n$, denoted as $\mathsf{FAI}(f)$, is defined as: $\mathsf{FAI}(f) = \min\{2\mathsf{AI}(f), \min_{1\leq \mathsf{deg}(g)<\mathsf{AI}(f)} \mathsf{deg}(g) + \mathsf{deg}(fg)\}$.*

Families of Boolean functions. We give the definition of families of Boolean functions that are used to build FiLIP filters, and we recall their parameters.

Definition 7 (XOR function). *For any positive integers k, XOR_k is the k-variable Boolean function such that for all $x = (x_1,\ldots,x_k) \in \mathbb{F}_2^k$*

$$\mathsf{XOR}_k(x) = \sum_{i=1}^{k} x_i.$$

Definition 8 (QUAD function). *For any positive integers q, Q_q is the $2q$-variable Boolean function such that $\forall x = (x_1,\ldots,x_{2q}) \in \mathbb{F}_2^{2q}$*

$$\mathsf{Q}_q(x) = \sum_{i=1}^{q} x_{2i-1}x_{2i}.$$

Definition 9 (Threshold function). *For any positive integers d and n such that $d \leq n+1$ we define the Boolean function $\mathsf{T}_{d,n}$ as follows:*

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

Definition 10 (Direct sum). *Let f be a Boolean function of n variables and g a Boolean function of m variables, the direct sum h of f and g is defined by $h(x,y) = f(x) + g(y)$ where $x \in \mathbb{F}_2^n$ and $y \in \mathbb{F}_2^m$.*

Property 1 (Direct sum properties (e.g., [MJSC16] Lemma 3)). Let h be the direct sum of two functions f, in n variables, and g, in m variables. Then h has the following cryptographic properties:

1. Resiliency: $\mathsf{res}(h) = \mathsf{res}(f) + \mathsf{res}(g) + 1$.
2. Nonlinearity: $\mathsf{NL}(h) = 2^m\mathsf{NL}(f) + 2^n\mathsf{NL}(g) - 2\mathsf{NL}(f)\mathsf{NL}(g)$.

3. Algebraic Immunity: $\max(\mathsf{AI}(f), \mathsf{AI}(g)) \leq \mathsf{AI}(h) \leq \mathsf{AI}(f) + \mathsf{AI}(g)$.

We focus on subfamilies obtained by the direct sum construction:

Definition 11 (Direct Sum of Monomials and Direct Sum Vector [MJSC16]). *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. We define its direct sum vector (DSV)$\mathbf{m}_f = [m_1, m_2, \ldots, m_k]$, 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 12 (XOR-THR Function). *For any positive integers k, d and n such that $d \leq n+1$ we define $\mathsf{XOR}_k + \mathsf{T}_{d,n}$ for all $z = (x_1, \ldots, x_k, y_1, \ldots, y_n) \in \mathbb{F}_2^{k+n}$ as follows:*

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

The XOR and QUAD functions are common functions in different areas and their properties are considered as folklore. The cryptographic properties of threshold and Xor-Threshold functions have been characterized in [CM19,CM22]. We summarize the parameters they reach in the following:

Property 2. Let $k \in \mathbb{N}$, then $f = \mathsf{XOR}_k$ has the following properties: $\mathsf{res}(f) = k-1, \mathsf{NL}(f) = 0$, and $\mathsf{AI}(f) = 0$ if $k = 0$, 1 otherwise.

Let $q \in \mathbb{N}$, then $g = \mathsf{Q}_q$ has the following properties: $\mathsf{res}(g) = 0$, $\mathsf{NL}(g) = 2^{2q-1} - 2^{q-1}$, and $\mathsf{AI}(g) = 0$ if $q = 0$, 1 if $q = 1$, 2 otherwise.

Let $d, n \in \mathbb{N}^*$ and $k \in \mathbb{N}^*$ such that $d \leq n+1$ the function $h = \mathsf{XOR}_k + \mathsf{T}_{d,n}$ has the following properties:

1. Resiliency: $\mathsf{res}(h) = \begin{cases} k & \text{if } n = 2d-1, \\ k-1 & \text{otherwise.} \end{cases}$
2. Nonlinearity:

$$\mathsf{NL}(h) = \begin{cases} 2^{n+k-1} - 2^k \binom{n-1}{(n-1)/2} & \text{if } d = \frac{n+1}{2}, \\ 2^k \sum_{i=d}^{n} \binom{n}{i} & \text{if } d > \frac{n+1}{2}, \\ 2^k \sum_{i=0}^{d-1} \binom{n}{i} & \text{if } d < \frac{n+1}{2}. \end{cases}$$

3. Algebraic Immunity: if $k = 0$, $\mathsf{AI}(h) = \min(d, n-d+1)$ otherwise:

$$\mathsf{AI}(h) = \begin{cases} \frac{n+1}{2} & \text{if } d = \frac{n+1}{2}, \\ \min(d+1, n-d+2) & \text{otherwise.} \end{cases}$$

4. $\Delta_{\mathsf{AN}}(h)$: $\Delta_{\mathsf{AN}}(h) = |n - 2d + 1|$ if $k = 0$, 0 otherwise.

We recall the notion of descendant functions [MJSC16] introduced to study the guess and determine attacks on FLIP [DLR16], and the concept of bit-fixing stable functions.

Definition 13 (Depth-ℓ descendants). *Let $f(x_1, \ldots, x_n)$ be a n-variable function with $n > 1$ and $\ell \in \mathbb{N}$ such that $0 \leq \ell < n$. Let S_ℓ be the collection of subsets $I = \{i_1, \ldots, i_\ell\} \subseteq \{1, \ldots, n\}$ such that $|I| = \ell$. The depth ℓ descendants $\mathcal{D}_\ell(f)$ is the set of functions in $n-\ell$-variables that can be obtained by setting $x_{i_j} = b_j$ for $j = 1, \ldots, \ell$ and $x_j \in I$ for every possible pair of $(I, b_1, \ldots, b_\ell) \in S_\ell \times \mathbb{F}_2^\ell$.*

For I in $\{1, \ldots, n\}$ and $b \in \mathbb{F}_2^{|I|}$ we denote by $f_{I,b}$ the descendant of f where the variables indexed by I are fixed to the values of b.

Definition 14 (Bit-fixing stability).

Let $\mathcal{F}$ be a family of Boolean functions, $\mathcal{F}$ is called bit-fixing stable, or stable relatively to guessing and determining, if for all functions $f \in \mathcal{F}$ such that f is a n-variable function with $n > 1$, all its (depth-ℓ) descendants are such that $f_{I,b} \in \mathcal{F}$, or $f_{I,b} + 1 \in \mathcal{F}$, or $\mathsf{deg}(f_{I,b}) = 0$.

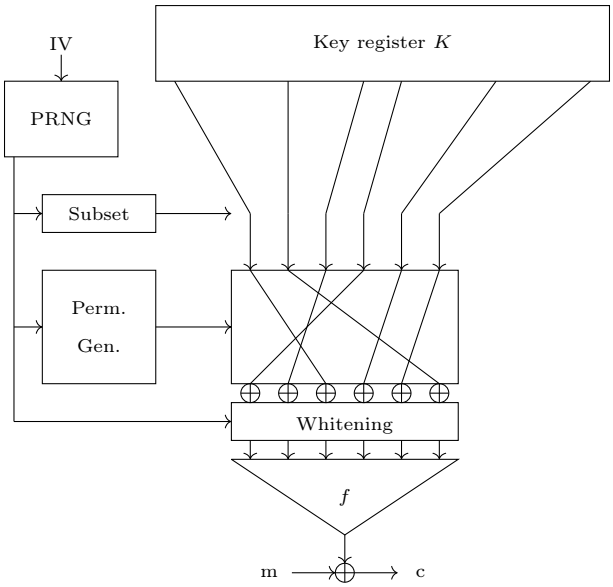

Fig. 1: Improved filter permutator paradigm.

2.2 Improved filter permutator and FiLIP

FiLIP is a family of stream ciphers following the Improved Filter Permutator paradigm (IFP) [MCJS19b], represented in Figure 1.

The IFP is composed of 6 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 a generator of subsets, a generator of binary strings called whitenings, and a filter function which produces the keystream.

For a security parameter λ, to encrypt $m \leq 2^{\frac{\lambda}{\kappa}}$ bits under a secret key $K \in \mathbb{F}_2^N$, the public parameters of the PRNG are chosen and then the following process is executed for each keystream bit b_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 b_i is computed, $b_i = f(P_i(S_i(K)) + w_i)$ through the n-variable Boolean filter f.

FiLIP [MCJS19b] stream cipher family is an instantiation of the IFP paradigm. The PRNG is instantiated with the forward secure PRG construction from [BY03] using AES as the underlying block cipher. So far, two kinds of instances have been introduced, with DSM functions, and with Xor-Threshold functions in the long version [MCJS19a], we recall them in Table 1.

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
n	N	DSM with $\mathbf{m}_f$	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	$[128, 64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 64]$	128

Table 1: FiLIP instances [MCJS19a].

2.3 Attacks applying on IFP

In this part we recall the attacks studied on the IFP and FiLIP so far, necessary to understand the functions implemented in the tool, we refer to [MCJS19b,MCJS19a] for the security analysis of the cipher. In a nutshell, the attack considered are adaptations of the ones applying on filtered Linear Feedback Shift Register (LFSR) as considered for FLIP [MJSC16], combined with guess and determine attacks, since these attacks been shown efficient on filter permutator designs [DLR16].

An IFP produces keystream bits by applying first linear operations and then a filter Boolean function. The usage of a PRNG avoids biases in the intermediate operations, *i.e.* subset selection, permutation and whitening, and leaves as principal target for the security analysis the filter function. Méaux *et al.* [MCJS19b] observed that resemblance to filter generators may expose IFPs to the attacks developed for these stream ciphers. Namely, they described a framework to bound the security of IFPs by analyzing the impact of known key-recovery attacks and their combinations, based on the evaluation of complexities via the estimation of Boolean criteria. The authors identify three main types of attack in the known-ciphertext model: algebraic-type attacks, correlation-type attacks, and guess-and-determine strategies. While the first two kinds of attack exploit properties of a given function, the guess-and-determine strategies consists in first guessing a few key bits and determining the security in such case, *i.e.* by using properties of functions in less variables. In fact, guess-and-determine strategies include algebraic and correlation type attacks as building blocks and profit from a time/data complexity trade-off.[3]

More precisely, let us consider an n-variable Boolean function and a fixed subset set of ℓ variables. Then, guessing ℓ bits of the input corresponds to pre-set those ℓ-variables and produce 2^ℓ *descendants*, *i.e.* 2^ℓ Boolean functions in $n-\ell$ variables. Hence, each descendant can be independently studied and contributes to the full strategy according to the impact of the chosen type of attack on the descendant itself. Therefore, to claim a level of security λ for a specific filter function, one has to take into account both data and time complexities of all the possible classes of descendants up to $\ell \leq \lambda$.

We will thoroughly explain the procedure to compute the complexity of a guess-and-determine strategy associated with one or more attacks in Section 3. While, we summarise here the algebraic-type and correlation-type attacks that we will use as building blocks. For both classes we consider a few significant algorithms, as suggested in [MCJS19b], and we identify the suitable Boolean criteria for estimating the data and time complexities.

Algebraic-type attacks. We refer to algorithms generating systems of equations from the key-stream and using algebraic techniques to retrieve the secret key as algebraic attacks. Algebraic attacks [CM03], fast algebraic attacks [Cou03a] and use of Gröbner bases, *e.g.* [Fau99,Fau02], belong to this class. For simplicity, we take into account the first two types as representative of this class.

Following [CM03], the so called *algebraic attack* (AA) aims to solve an over-defined system of equations, possibly via linearization, previously processed to reduce the degree. In fact, if f is the n-variable input Boolean function, one can find a non null function g such that the algebraic degree of gf is lower. Hence, since the equations in the system have form $f(y_i) = b_i$ where y_i is a linear transformation of the original variables, by multiplying the equations by g the degree system decreases and consequently the cost of the algorithm. The minimal reachable degree is determined by $\mathsf{AI}(f)$ the algebraic immunity of f (Definition 5), since for instance by multiplying by a non null annihilator g we obtain an equation $g(y_i)b_i = f(y_i)g(y_i) = 0$.

The linearization of a multivariate system of degree $\mathsf{AI}(f)$ in n variables produces a linear system in $D = \sum_{i=0}^{\mathsf{AI}(f)} \binom{n}{d}$ variables. The AA consists in solving such reduced system. Then, we consider a bound for the time complexity and the data complexity $O(D^\omega)$ and $O(D/\binom{n}{\mathsf{AI}(f)})$, respectively, where we denote by ω the constant of linear algebra.

Relaxing the condition on the minimality of the degree and introducing intermediate elimination steps can improve the attack's performance. This variation was proposed by Courtois in [Cou03a], we refer to it as *fast algebraic attack* (FAA). The main idea is to consider also functions g of degree $e < \mathsf{AI}(f)$ and such that gf has low degree d, and later remove monomials of degree larger than e, via Berlekamp-Massey algorithm. We refer to [Cou03a,ACG+06] for details. Letting $E = \sum_{i=0}^{e} \binom{n}{i}$ and $D = \sum_{i=0}^{d} \binom{n}{i}$, for filter functions we consider in this context we expect the time complexities of the algorithm to be $O(D\log_2(D) + ED\log_2(D) + E^\omega)$ and the data complexities being $O(D/\binom{n}{d}))$. In our tool we will consider only $e = 1$, *i.e.* $E = n+1$, and $d = \mathsf{AI}(f)$, which is always giving a lower bound of the complexities, and in this case the Boolean criterion to take into account remains the algebraic immunity.

[3] Notice that in this context time complexity refers to the expected running time of the algorithm, while data complexity refers to the size of the keystream bits' sample.

Correlation-type attacks. We refer to algorithms exploiting properties of the distribution's outputs as correlation-type attacks. More precisely, these attacks aim to distinguish the key-stream from a random sequence to learn properties of the functions and later apply more specific algorithms. Among the possible attacks we consider here those which take advantage of either proximity to affine functions or non balancedness of the filter function. In fact, we can characterize the effectiveness of the related attacks by studying its *nonlinearity* and *resiliency* of the filter function, respectively.

More precisely, if we consider the higher order correlation attack with XL algorithm [Cou03b], its complexity grows with the degree. Then an attacker would rather prefer working with functions of low degree if possible. If the filter function f has high degree, to reduce the complexity of this approach, one can search for a low degree approximation, *i.e.* a function g that has a low algebraic degree d and such that $f(x) = g(x)$ with probability $1 - \varepsilon$ for a real $\varepsilon \in [0, 1]$, and apply the XL algorithm to the system associated to this new function. We call this attack (HCA) and refer to [Cou03b] for the details. In particular, an affine approximation ($d \leq 1$) results into an attack having time complexity at least $O(n^{\omega}(1-\varepsilon)^{-n})$ and data complexity $O((1-\varepsilon)^{-n})$, (using the lower bounds from [MCJS19a] Section 9.1). Since the distance to the closest affine function is the nonlinearity, we consider $\varepsilon = \mathsf{NL}(f)/2^n$.

Furthermore, we recall that the resiliency of a Boolean function is the maximum value k such that every restrictions of f obtained by fixing at most k of its coordinates is balanced. If a function is unbalanced, the keystream generated can be distinguished from one uniformly at random by comparing the expectations of the output. If the function is balanced, we can similarly analyze descendants at level its resiliency plus one, to distinguish the distributions. The quantity of bits necessary to determine the presence of a bias in the keystream depends on the value is $O(\delta^{-2})$ where

$$\delta(f) = \frac{1}{2} - \frac{\mathsf{NL}(f)}{2^n} \tag{1}$$

We consider as lower bound from the time and data complexity of this simple correlation attack (CA) $2^{\mathsf{res}(f)}\delta(f)^{-2}$ and $\delta(f)^{-2}$, respectively (and refer to [MCJS19a], Section 9.1 for more details).

Remark 1. We explained in this section a few algorithms and determined "safe" (but meaningful) bounds on their complexity in order to evaluate the security of IFP. In the following, we will use the results about the algebraic and correlation-like attacks algorithms as building blocks inside the main guess-and-determine strategy. Although the list of algorithms we consider is not exhaustive, the design of the tool allows to plug other algorithms. Hence, different and future attack can also be included in the security evaluation.

3 Tool's description

3.1 Attacks and profiles

The problem of bounding the security of IFP was first addressed in [MCJS19a]. We recall the suggested procedure in this section. Previously, we discussed several attacks for which the complexity can be evaluated by computing the parameter of the filter function relatively to a criterion of Boolean functions. In practice, a well-know strategy is to perform guess-and-determine. In our case, instead of computing the complexity only on the initial function, one can first find a descendant (See Definition 13) by guessing a few bits of the input and then determine the complexity on this smaller instance of the problem. Since a single guess might not to lead to a significantly simple instance, one likely has to repeat the process multiple times and target a weakness that share various descendants. As the number of smaller instances is exponential in the number of guessed variables of the Boolean function, evaluating the complexity on all of them independently is unfeasible for concrete security parameters.

Nevertheless, we can optimize this computation by merging equivalent descendants. For instance, consider $f_n(x_1, \ldots, x_n) = \mathsf{XOR}_n = x_1 + x_2 + \cdots + x_n$. By setting any variable to 0, we always obtain a descendant of the form $f_{n-1}(y_1, \ldots, y_{n-1}) = y_1 + y_2 + \cdots + y_{n-1}$. Namely, we get n equivalent descendants. Similarly, after ℓ guesses we can have only two types of descendant $f_{n-\ell}$ and $f_{n-\ell} + 1$. While in this example the number of descendant stays moderate, for other functions it might not be the case. In fact, the merging procedure has a major impact on the running time of the algorithm evaluating the security.

Furthermore, one can group descendants according to a specific criterion since all the attacks' complexities depend only the number of variables and some criterion[4]. See Section 2.3. For the function f_n considered above, it actually means that all descendants at depth ℓ (that is to say, after fixing ℓ variables) are equivalent since the values of the criteria considered in the attacks are the same for $f_{n-\ell}$ and $f_{n-\ell} + 1$.

[4] If the complexity of the attack depends on more than one criterion, one has to categorize the function according to all of them. The overall strategy remains unchanged.

Eventually, the crucial point for evaluating the impact of an attack on the security of IFP, via this guess-and-determine strategy, is to compute the distribution of selected Boolean criteria over the set of all descendants up to the security level. Indeed, the core of the strategy consists in running the attack on the initial function and descendants that are obtained by guessing some bits. This implies that we have to consider the probability to find by guessing descendants leading to a certain complexity. For instance, a descendant at depth ν might be easy to attack but the probability of guessing it could be only $2^{-\nu}$, *i.e.* making this attack unpractical.

Since the probability of getting an object with a certain Boolean criterion (or criteria) is relevant to the computation of the expected complexity, we define the following:

Definition 15 ((π, ℓ)-profile). *Let π be a function defined over the set of all Boolean functions to an ordered discrete set. Let $p_j : \pi(\mathcal{D}_j(f)) \to [0, 1]$ be probability distribution function $p_j(a) = P(\pi(g) = a | g \in \mathcal{D}_j)$. The (π, ℓ)-profile of f is the sequence $C_0, \ldots, C_\ell$ of the cumulative functions of $p_0, \ldots, p_\ell$, respectively.*

For instance, if π is the function outputting the algebraic immunity, $\pi(\mathcal{D}_j)$ is the set of all possible values of the AI. Then, the (AI, ℓ)-profile at index $i \leq \ell$ contains the probability that a depth-i descendant picked uniformly at random has an AI of at most k for $k \in \pi(\mathcal{D}_i)$. The practical computation of the profile is discussed in Section 3.3.

To evaluate the complexity of an attack relatively to an IFP with filter function f in n variables and register size N, we guess L bits of the key register. Then, we also have to take into account the probability of ℓ out of L guessed bits actually being selected as input for the filter function, *i.e.*

$$P_{L=\ell} = \frac{\binom{L}{\ell} \cdot \binom{N-L}{n-\ell}}{\binom{N}{n}}.$$

Hence, for every value of the set of criteria $k \in \bigcup_{\ell=0}^{L} \pi(\mathcal{D}_\ell(f))$ the probability of getting a descendant with at most such k is $P(k, L) = \sum_{\ell=0}^{L} P_{L=\ell} \cdot C_\ell(k)$ For instance, if π is the algebraic immunity, this is the probability of getting a descendant with AI at most k by guessing L bits of the key.

Therefore, to analyze, up to λ bits of security, the guess-and-determine strategy with a building block attack associated to a function π encoding a set of criteria, time complexity function $\texttt{time}(\cdot)$ and data complexity function $\texttt{data}(\cdot)$ we do the following operations: for every $L \in \mathbb{N}$ such that $0 \leq L \leq \lambda$ and for each value $k \in \bigcup_{\ell=0}^{\lambda} \pi(\mathcal{D}_\ell(f))$ we compute:

- the time complexity given by $\mathtt{t}_{k,L} = 2^L \cdot \texttt{time}(\cdot)$;
- the data complexity is affected by the probability of getting a descendant $g \in \bigcup_{\ell=0}^{L} \pi(\mathcal{D}_\ell(f))$ having such that $\pi(g) \leq k$, *i.e.* $P(k, L)$, so we set it to $\mathtt{d}_{k,L} = P(k, L)^{-1} \cdot \texttt{data}(\cdot)$. Furthermore, since the adversary is limited to observe λ/κ ciphertexts, this complexity can actually be scaled by κ.

Finally, the lower bound on the guess-and-determine strategy is given by

$$\min_{k,L} \left\{ \max \left\{ \mathtt{t}_{k,L}, \kappa \cdot \mathtt{d}_{k,L} \right\} \right\}. \tag{2}$$

An algorithmic description can be found in Algorithm 1.

3.2 Implementation

We provide Python scripts to evaluate the security of new parameters for FiLIP. The code aims to be easy to modify and reuse to study new instances. In particular, the user simply has to describe new functions in order to study their security when used as filter in FiLIP. The code has three main components: the profiling module, the attack module and the function objects.

Function objects New families of functions are created by writing a class following a given interface. While it would be natural to have a hierarchy with an abstract class at the top representing a generic Boolean function, we do not provide one since we felt that it is unnecessary and describing the interface is enough. A documented blank class is provided as template for the user to create new functions with all the required methods. In a nutshell, the class will contains methods to evaluate the criteria used in the attacks (Algebraic Immunity, Resiliency and Nonlinearity), to compute the direct descendants of the function that are obtained by fixing a variable to 0 or 1 (as well as their probability) and to compare with other functions in order to detect equality. Instances of the class represent functions object with concrete parameters. For example `f = XOR_TR(2,5,10)` will create an object corresponding to $\mathsf{XOR}_2 + \mathsf{T}_{5,10}$ and `f.AI()` will return its Algebraic Immunity.

Profiling module The profiling module is computing the different profiles corresponding to each criterion of a function. A profile is represented by a matrix for which columns are indexed by the different values that can be taken by the criterion. The row i contains at index j the probability that descendants obtained by fixing i variables have a value below or equal to j for the given criterion. Hence, in the first row, where only the initial function is considered, all the indexes will have value 0, except the ones that are superior or equal to the criterion (in this case we have a 1). We note that if an attack depends on n criteria, we need a $n+1$ dimensional array. In our case we mostly consider matrices since most attacks depend on one criterion, the only exception is the attack depending on the Resiliency and Nonlinearity.
The profiling phase starts with the creation of a profile of appropriate size for each criterion. In most of the cases, the maximum value for the criterion is known without exploring the descendants. Indeed, Algebraic Immunity can only decrease and δ and ϵ are in the interval $[0, 0.5]$. The exception is resiliency that can increase when fixing a bit and thus the size of the array must correspond to the maximum resiliency among all descendants.
Then, up to a maximal depth in the (rooted directed) graph of descendants equal to the security parameter, we compute the descendants of each node at the current level, their probability, and update the profile accordingly by computing the criteria on each descendant. Since we want each row of the profile for criterion C to contain at index i the probability $Pr[C \leq i]$ and not $Pr[C = i]$, we need to eventually accumulate the values in each row.
Since, for a given function, two descendants are obtained by fixing a variable to 0 and 1, their amount should grow exponentially at each level. Obviously, this would make the computation of the profile impossible since we have to explore the descendants up to fixing a number of variables equal to the security parameter. This is why an important aspect of profiling is to detect when two descendants are equivalent and merge their node (this is why all the descendants form a graph and not a tree). In the implementation, we rely on Python dictionaries to store descendants while detecting collisions. This forces us to define the hashing operator on function objects in order to use the hash map structure offered by the dictionary.

Attack module Once the profiling phase is finished, the complexities of the attacks can be evaluated. All the attacks are very similar and a generic code could be written to implement the guess and determine strategy. For the sake of simplicity, and to potentially add some tweaks, we separated them in different classes having the same interface. Each instantiation of an attack requires a profile, a function and a register size. Being associated to one or several criteria, it requires methods to obtain data and time complexities with respect to the criteria. The main method actually computing bounds for the complexities implements Algorithm 1.

Configuration Since we want the code to be flexible in terms of modeling the attack scenario, we added a small configuration file containing the main constants. In particular, the user can easily modify

- κ the ratio between the time and the data security parameters,
- the granularity of the discretization for the criteria depending on the non-linearity,
- the value ω used in some algebraic and correlation attacks.

An algorithmic description can be found in Algorithm 1. Notice that the tool never actually runs the chosen building block attack.

Algorithm 1 Guess-and-determine attack

Require: A filter function f, a profile P, a depth λ a range for the criterion $[min_C, max_C]$ and a scaling factor κ modeling the limited amount of queries of the adversary.
Ensure: Complexity c

$lowest\_complexity \leftarrow \infty$
for $L = 0$ to λ **do**
  $c \leftarrow min_C$
  while $c \leq max_C$ **do**
    $time\_complexity \leftarrow L + \texttt{time}(\cdot)$
    $proba \leftarrow 0$
    for $\ell = 0$ to L **do**
      $proba_\ell \leftarrow \binom{L}{\ell} \cdot \binom{N-L}{n-\ell} / \binom{N}{n}$
      $proba \leftarrow proba + proba_\ell \cdot P[\ell][c]$
      if $proba \neq 0$ **then**
        $data\_complexity \leftarrow \texttt{data}(\cdot)/proba$
        $complexity \leftarrow \texttt{max}(\kappa \cdot data\_complexity, time\_complexity)$
        $lowest\_complexity \leftarrow \texttt{min}(complexity, lowest\_complexity)$
      end if
    end for
    $c \leftarrow c + 1$
  end while
end for
return $lowest\_complexity$

3.3 Implementation

We provide Python scripts to evaluate the security of new parameters for FiLIP. The code aims to be easy to modify and reuse to study new instances. In particular, the user simply has to describe new functions in order to study their security when used as filter in FiLIP. The code has three main components: the profiling module, the attack module and the function objects.

Function objects. New families of functions are created by writing a class following a given interface. While it would be natural to have a hierarchy with an abstract class at the top representing a generic Boolean function, we do not provide one since we felt that it is unnecessary and describing the interface is enough. A documented blank class is provided as template for the user to create new functions with all the required methods. In a nutshell, the class will contain methods to evaluate the criteria used in the attacks (Algebraic Immunity, Resiliency and Nonlinearity), to compute the direct descendants of the function that are obtained by fixing a variable to 0 or 1 (as well as their probability) and to compare with other functions in order to detect equality. Instances of the class represent functions object with concrete parameters. For example `f = XOR_TR(2,5,10)` will create an object corresponding to $\mathsf{XOR}_2 + \mathsf{T}_{5,10}$ and `f.AI()` will return its Algebraic Immunity.

Profiling module. The profiling module is computing the different profiles corresponding to each criterion of a function. A profile is represented by a matrix for which columns are indexed by the different values that can be taken by the criterion. The row i contains at index j the probability that descendants obtained by fixing i variables have a value below or equal to j for the given criterion. Hence, in the first row, where only the initial function is considered, all the indexes will have value 0, except the ones that are superior or equal to the criterion (in this case we have a 1). We note that if an attack depends on n criteria, we need a $n+1$ dimensional array. In our case we mostly consider matrices since most attacks depend on one criterion, the only exception is the attack depending on the Resiliency and Nonlinearity.
The profiling phase starts with the creation of a profile of appropriate size for each criterion. In most of the cases, the maximum value for the criterion is known without exploring the descendants. Indeed, Algebraic Immunity can only decrease and δ and ϵ are in the interval $[0, 0.5]$. The exception is resiliency that can increase when fixing a bit and thus the size of the array must correspond to the maximum resiliency among all descendants.
Then, up to a maximal depth in the (rooted directed) graph of descendants equal to the security parameter,

we compute the descendants of each node at the current level, their probability, and update the profile accordingly by computing the criteria on each descendant. Since we want each row of the profile for criterion C to contain at index i the probability $Pr[C \leq i]$ and not $Pr[C = i]$, we need to eventually accumulate the values in each row.
Since, for a given function, two descendants are obtained by fixing a variable to 0 and 1, their amount should grow exponentially at each level. Obviously, this would make the computation of the profile impossible since we have to explore the descendants up to fixing a number of variables equal to the security parameter. This is why an important aspect of profiling is to detect when two descendants are equivalent and merge their node (this is why all the descendants form a graph and not a tree). In the implementation, we rely on Python dictionaries to store descendants while detecting collisions. This forces us to define the hashing operator on function objects in order to use the hash map structure offered by the dictionary.

Attack module. Once the profiling phase is finished, the complexities of the attacks can be evaluated. All the attacks are very similar and a generic code could be written to implement the guess and determine strategy. For the sake of simplicity, and to potentially add some tweaks, we separated them in different classes having the same interface. Each instance of an attack requires a profile, a function and a register size. Being associated to one or several criteria, it requires methods to obtain data and time complexities with respect to the criteria. The main method actually computing bounds for the complexities implements Algorithm 1.

Configuration. Since we want the code to be flexible in terms of modelling the attack scenario, we added a small configuration file containing the main constants. In particular, the user can easily modify

- κ the ratio between the security parameter,
- the granularity of the discretization for the criteria depending on the non-linearity,
- the value ω used in some algebraic and correlation attacks.

4 Candidate secure functions

In this section, we discuss new functions that can be used to instantiate FiLIP as well as concrete parameters at various security levels. All of the results were obtained by using the tool presented in Section 3.

4.1 Former instances with smaller key size

Unlike in [MCJS19a], we decided to limit the amount of ciphertext data that the adversary can obtain while mounting the attack. In practice, we bound the number of queries to the square root of the computational capabilities of the adversary. For example, if the targeted security parameter λ is equal to 80, we only consider as successful attacks that require less than 2^{40} ciphertexts. This correspond to setting the κ parameter to 2 in Algorithm 1. Thus, it was natural to have a second look at some of the instances that were described in [MCJS19a] and to propose new parameters that would either reduce the register size or speed up the homomorphic evaluation of the filter. The results are presented in Table 2.

For a security of $\lambda = 80$ bits, we can obtain a function in 144 variables with a register size as small as $N = 530$, this size is almost four times smaller than what was proposed in [MCJS19a]. However, this requires a threshold function in more variables, that might be more expensive to evaluate. Thus we also propose an alternative with $N = 982$ and a threshold function in less variables.

For a security of $\lambda = 128$ bits, several trade-offs can be made depending on what the user of the filter finds convenient. In the table, we highlighted two functions $f_1 = \mathsf{XOR}_{65} + \mathsf{T}_{32,63}$ and $f_2 = \mathsf{XOR}_{58} + \mathsf{T}_{35,70}$ which both have a minimal number of variable ($n = \lambda$). Surprisingly, f_1 also exhibits a register size that is almost twice as small as the smaller register size possible for this value of λ in [MCJS19a].

Reducing the value of N can lead to a significant save in practice, for example, in the framework of [CDPP22], the bandwidth is directly proportional to the register size since they encrypt separately each key bit of FiLIP. In their case, this comes with no penalty cost on the runtime since it is mainly driven by the threshold size s which stays the same. In fact, f_1 will even be slightly faster to evaluate since less additions are required for the threshold part.

k	d	s	n	N	λ
40	52	104	144	530	80
100	24	44	144	982	80
65	32	63	128	2560	128
58	35	70	128	4096	128

k	d	s	n	N	λ
80	260	520	600	1200	128
70	93	186	256	1499	128
65	96	191	256	1461	128
70	61	122	192	1777	128
160	48	96	256	1987	128

Table 2: For N and n the function $\mathsf{XOR}_k + \mathsf{T}_{d,s}$ provides λ-bit of security.

4.2 Instances with new families of direct sum of threshold functions (DST)

We remarked that both kind of filters previously used to instantiate FiLIP belong to a larger family, that we call direct sum of thresholds, or DST. Indeed, a degree d monomial corresponds to the threshold function $\mathsf{T}_{d,d}$, hence DSM functions are the direct sums of multiple threshold functions of highest threshold. The Xor-Threshold functions are the sum of many thresholds $\mathsf{T}_{1,1}$ and one high threshold in more variables, usually a majority function. Considering direct sums of threshold for FiLIP filters has different interest. First, threshold function can be efficiently evaluated homomorphically (*e.g.* [HMR20,CDPP22,MPP23]), then the resiliency and nonlinearity can be exactly determined based on the properties of threshold functions, finally these functions are bit-fixing stable. We focus on two new subfamilies, interesting for their cryptographic properties and mainly for their friendliness regarding homomorphic evaluation.

Definition 16 (XOR-QUAD-THR function). *For any positive integers k, q, d and n such that $d \leq n+1$ we define $\mathsf{XOR}_k + \mathsf{Q}_q + \mathsf{T}_{d,n}$ for all $v = (x_1, \ldots, x_k, y_1, \ldots, y_{2q}, z_1, \ldots, z_n) \in \mathbb{F}_2^{k+2q+n}$ as:*

$$\begin{aligned}(\mathsf{XOR}_k + \mathsf{Q}_q + \mathsf{T}_{d,n})(v) &= x_1 + \cdots + x_k + \sum_{i=1}^{q} y_{2i-1}y_{2i} + \mathsf{T}_{d,n}(z_1, \ldots, z_n) \\ &= \mathsf{XOR}_k(x) + \mathsf{Q}_q(y) + \mathsf{T}_{d,n}(z).\end{aligned}$$

Definition 17 (XOR-THR-THR function). *For any positive integers k, d_1, d_2, n_1 and n_2 such that $d_1 \leq n_1 + 1$ and $d_2 \leq n_2 + 1$ we define $\mathsf{XOR}_k + \mathsf{T}_{d_1,n_1} + \mathsf{T}_{d_2,n_2}$ for all $v = (x_1, \ldots, x_k, y_1, \ldots, y_{n_1}, z_1, \ldots, z_{n_2}) \in \mathbb{F}_2^{k+n_1+n_2}$ as:*

$$\begin{aligned}(\mathsf{XOR}_k + \mathsf{T}_{d_1,n_1} + \mathsf{T}_{d_2,n_2})(v) =& x_1 + \cdots + x_k \\ &+ \mathsf{T}_{d_1,n_1}(y_1, \ldots, y_{n_1}) + \mathsf{T}_{d_2,n_2}(z_1, \ldots, z_{n_2}) \\ =& \mathsf{XOR}_k(x) + \mathsf{T}_{d_1,n_1}(y) + \mathsf{T}_{d_2,n_2}(z).\end{aligned}$$

Using Property 1 and Property 2 we can compute the resiliency, nonlinearity and a lower bound on the algebraic immunity of these functions. Since the AI value is often the bottleneck for the security, we rely on a better bound that we derive using a result from [Méa22] using the parameter $\Delta_{\mathsf{AN}}(f)$ (see Definition 5).

Property 3 ([Méa22], Corollary 3). Let $n, m \in \mathbb{N}^*$, $f \in \mathcal{B}_n$, $g \in \mathcal{B}_m$, and ψ the direct sum of f and g, the following bound holds on its algebraic immunity:

$$\mathsf{AI}(\psi) \geq \max\big(\mathsf{AI}(f) + \min\{\Delta_{\mathsf{AN}}(f), \mathsf{AI}(g)\}, \mathsf{AI}(g) + \min\{\Delta_{\mathsf{AN}}(g), \mathsf{AI}(f)\}\big).$$

We detail the property of these two subfamilies in the following proposition:

Proposition 1 (Properties of XOR-QUAD-THR and XOR-THR-THR functions). *Let $k, q, d, n > 0$, and $d \leq n$, then the function $f = \mathsf{XOR}_k + \mathsf{Q}_q + \mathsf{T}_{d,n}$ has the following properties:*

1. *Resiliency:* $\mathsf{res}(f) = \begin{cases} k & \text{if } n = 2d-1, \\ k-1 & \text{otherwise.} \end{cases}$
2. *Nonlinearity:* $\mathsf{NL}(f) = 2^{n+k}(2^{2q-1} - 2^{q-1}) + 2^{2q}\mathsf{NL}(\mathsf{XOR}_k + \mathsf{T}_{d,n}) - (2^{2q} - 2^q)\mathsf{NL}(\mathsf{XOR}_k + \mathsf{T}_{d,n})$.
3. *Algebraic Immunity:* $\mathsf{AI}(f) \begin{cases} \geq \frac{n+1}{2} & \text{if } n - 2d + 1 = 0, \\ \geq \min(d+1, n-d+2) & \text{if } |n-2d+1| = 1, \\ = \min(d+2, n-d+3) & \text{otherwise.} \end{cases}$

Let $k, d_1, d_2, n_1, n_2 > 0$, $d_1 \leq n_1$ and $d_2 \leq n_2$, then the function $g = \mathsf{XOR}_k + \mathsf{T}_{d_1,n_1} + \mathsf{T}_{d_2,n_2}$ has the following properties:

1. *Resiliency:* $\mathsf{res}(g) = \begin{cases} k+1 & \text{if } n_1 = 2d_1 - 1 \text{ and } n_2 = 2d_2 - 1, \\ k-1 & \text{if } n_1 \neq 2d_1 - 1 \text{ and } n_2 \neq 2d_2 - 1, , \\ k & \text{otherwise.} \end{cases}$
2. *Nonlinearity:* $\mathsf{NL}(g) = 2^{n_2}\mathsf{NL}(\mathsf{XOR}_k + \mathsf{T}_{d_1,n_1}) + 2^{k+n_1}\mathsf{NL}(\mathsf{T}_{d_2,n_2}) - 2\mathsf{NL}(\mathsf{XOR}_k + \mathsf{T}_{d_1,n_1})\mathsf{NL}(\mathsf{T}_{d_2,n_2})$.
3. *Algebraic Immunity:*

$$\begin{aligned} \mathsf{AI}(g) \geq \max\{ & \min(d_1, n_1 - d_1 + 1) + \\ & + \min(|n_1 - 2d_1 + 1|, \min(d_2, n_2 - d_2 + 1) + \varepsilon_g), \\ & \min(d_2, n_2 - d_2 + 1) + \\ & + \min(|n_2 - 2d_2 + 1|, \min(d_1, n_1 - d_1 + 1) + \varepsilon_f)\}. \end{aligned}$$

where $\varepsilon_g = 1$ if $n_2 \neq 2d_2 - 1$, 0 otherwise, and $\varepsilon_f = 1$ if $n_1 \neq 2d_1 - 1$, 0 otherwise.

Proof. For the resiliency and nonlinearity, the exact number are directly derived from Property 1, and Property 2, using the direct sum with $\mathsf{XOR}_k + \mathsf{T}_{d,n}$ and Q_q for f, $\mathsf{XOR}_k + \mathsf{T}_{d_1,n_1}$ and T_{d_2,n_2} for g.

Then we show the bounds on the algebraic immunity of $f = \mathsf{XOR}_k + \mathsf{Q}_q + \mathsf{T}_{d,n}$. If $n - 2d + 1 = 0$ then $\min(d, n - d + 1) = d = (n + 1)/2$ and using Property 1 Item 3, $\mathsf{AI}(f) \geq \mathsf{AI}(\mathsf{T}_{(n+1)/2,n})$, that is $\mathsf{AI}(f) \geq (n+1)/2$ using Property 2 Item 3. For the two other cases, we use Property 3, considering the direct sum of $\mathsf{T}_{d,n}$ and $\mathsf{XOR}_k + \mathsf{Q}_q$. If $|n - 2d + 1| = 1$ then $\mathsf{AI}(\mathsf{T}_{d,n}) = \min(n, n - d + 1)$ and $\Delta_{\mathsf{AN}}(\mathsf{T}_{d,n}) = 1$ (by Property 2), then the bound gives $\mathsf{AI}(f) \geq \min(n, n - d + 1) + \min(1, \mathsf{AI}(\mathsf{XOR}_k + \mathsf{Q}_q))$. Since $\mathsf{XOR}_k + \mathsf{Q}_q$ is not a constant function its AI is at least 1, therefore in this case $\mathsf{AI}(f) \geq \min(n + 1, n - d + 2)$. For the last case, $|n - 2d + 1| > 1$, using the same reasoning $\mathsf{AI}(f) \geq \min(n, n - d + 1) + \min(2, \mathsf{AI}(\mathsf{XOR}_k + \mathsf{Q}_q))$. Since by definition $k > 0$ and $q > 0$, the function $\mathsf{XOR}_k + \mathsf{Q}_q$ is the direct sum of $\mathsf{XOR}_1 + \mathsf{Q}_1 = \mathsf{XOR}_1 + \mathsf{T}_{2,2}$ and another function. Since $\mathsf{XOR}_1 + \mathsf{T}_{2,2}$ has algebraic immunity 2 using Property 2 Item 3, applying Property 1 Item 3 we get $\mathsf{AI}(\mathsf{XOR}_k + \mathsf{Q}_q) \geq 2$, allowing us to conclude $\mathsf{AI}(f) \geq \min(d+2, n-d+3)$. In this case we have $\mathsf{AI}(f) = \min(d + 2, n - d + 3)$ since $\mathsf{AI}(f) \leq \mathsf{AI}(\mathsf{T}_{d,n}) + \mathsf{AI}(\mathsf{XOR}_k + \mathsf{Q}_q)$ using the upper bound of Property 1 Item 3 since $\mathsf{AI}(\mathsf{T}_{d,n}) = \min(d, n-d+1)$ and $\mathsf{AI}(\mathsf{XOR}_k + \mathsf{Q}_q) = 2$ since it is a degree 2 function (of AI greater than 1).

Finally, we demonstrate the bounds on the algebraic immunity of $g = \mathsf{XOR}_k + \mathsf{T}_{d_1,n_1} + \mathsf{T}_{d_2,n_2}$. Using Property 3 with T_{d_1,n_1} and $\mathsf{XOR}_k + \mathsf{T}_{d_2,n_2}$ we obtain (also using Property 2 Item 3 and 4)):

$$\begin{aligned} \mathsf{AI}(g) &\geq \mathsf{AI}(\mathsf{T}_{d_1,n_1}) + \min(\Delta_{\mathsf{AN}}(\mathsf{T}_{d_1,n_1}), \mathsf{AI}(\mathsf{XOR}_k + \mathsf{T}_{d_2,n_2})) \\ &\geq \min(d_1, n_1 - d_1 + 1) + \min(|n_1 - 2d_1 + 1|, \min(d_2, n_2 - d_2 + 1) + \varepsilon_g). \end{aligned}$$

Similarly, using Property 3 with T_{d_2,n_2} and $\mathsf{XOR}_k + \mathsf{T}_{d_1,n_1}$ we obtain:

$$\mathsf{AI}(g) \geq \min(d_2, n_2 - d_2 + 1) + \min(|n_2 - 2d_2 + 1|, \min(d_1, n_1 - d_1 + 1) + \varepsilon_f),$$

which allows us to conclude. □

Remark 2. First, we note that XOR-QUAD-THR functions have already been suggested as filters for FiLIP, as mentioned in [MCJS19a] Section 9.2, but with no concrete instantiation up to our knowledge. Second, for functions using more threshold functions of different degrees more general results from [Méa22] could be used to improve the lower bound on the AI of direct sums, such as Lemma 6 and Theorem 1. We did not find improvements for the two subfamilies we consider.

We studied the security of several concrete functions in the two families with TooLIP. Table 5 shows results for XOR-THR-THR. The practical goal of studying XOR-THR-THR was mainly to assess the impact of splitting a threshold in two thresholds of half the size. It is thus natural to compare Table 5 and Table 2. We can see that for non minimal values of n, splitting the threshold opens the possibility to reduce the register size N. However, XOR-THR functions that were obtained by setting $n = \lambda$ seems to still offer better register size. That being said, having two smaller threshold to evaluate could be an efficiency gain (depending on the method of homomorphic evaluation). For $\lambda = 80$, we have a straightforward improvement.

Tables 3 and 4 gives secure parameters for XOR-QUAD-THR functions. The idea was to add quadratic terms to heavily increase the nonlinearity without adding too many variables. Since this could have a negative impact on the evaluation time of the function depending on the evaluation technique used, we first tried to (arbitrarily) limit the number of quadratic terms in QUAD to 10 (see Table 3). Then, to broaden the result,

we allowed the number of terms in QUAD to grow further (see Table 4). We can see that this eventually lead to even smaller register size than the one we had with XOR-THR and XOR-THR-THR. While 10 quadratic terms seems not enough to significantly improve over the XOR-THR-THR construction, 20 QUAD offers 128 bits of security for a register size of only 541. This is approximately 7 times smaller than the smallest function proposed in [MCJS19a]. Furthermore, it also allows us to propose 256-bit secure instances for a reasonable number of variables.

k	q	d	s	n	N	λ
40	10	52	104	164	329	80
100	10	21	44	164	329	80
70	10	93	186	276	789	128
65	10	96	191	276	780	128
70	10	61	122	212	834	128
160	10	48	96	276	864	128

Table 3: For N and n the function $\mathsf{XOR}_k + \mathsf{Q}_{10} + \mathsf{T}_{d,s}(n)$ provides λ-bit of security.

k	q	d	s	n	N	λ
60	20	85	170	270	541	128
60	20	65	150	250	873	128
50	40	80	160	290	617	128
50	60	65	150	320	640	128
60	40	65	150	290	581	128
60	60	50	100	280	560	128
60	60	60	120	300	600	128
160	15	60	120	310	5000	256
140	20	60	120	310	4800	256

Table 4: For N and n the function $\mathsf{XOR}_k + \mathsf{Q}_q + \mathsf{T}_{d,s}(n)$ provides λ-bit of security.

k	d_1	s_1	d_2	s_2	n	N	λ
40	26	52	26	52	144	360	80
100	11	22	11	22	144	397	80
54	19	38	19	38	130	6500	128
54	22	45	23	45	144	3072	128
70	30	61	31	61	192	841	128
65	47	95	48	96	256	830	128
70	46	93	47	93	256	736	128
60	26	53	27	53	166	1229	128
160	24	48	24	48	256	913	128

Table 5: For N and n the function $\mathsf{XOR}_k + \mathsf{T}_{d_1,s_1} + \mathsf{T}_{d_2,s_2}$ provides λ-bit of security.

4.3 FiLIP with arbitrary filter

We consider instances of FiLIP with arbitrary functions, in this case functions given by the their number of variables, algebraic immunity, resiliency and nonlinearity only. We use that "arbitrary" functions is a bit-fixing stable family (Definition 14), and we can track the parameters of the descendants using the following proposition.

Proposition 2 (Parameter degradation by bit-fixing). *Let $n \in \mathbb{N}^*$ and f be an n variable Boolean function, for all $i \in \{1, \ldots, n\}$ and $\varepsilon \in \{0, 1\}$ the following bound applies on the properties of $f_{i,\varepsilon}$:*

- *Resiliency:* $\mathsf{res}(f_{i,\varepsilon}) \geq \mathsf{res}(f) - 1$.
- *Algebraic immunity:* $\mathsf{AI}(f_{i,\varepsilon}) \geq \mathsf{AI}(f) - 1$.

Proof. First, we show the result on the resiliency. By contradiction we assume there exists i and ε such that $\mathsf{res}(f_{i,\varepsilon}) < \mathsf{res}(f) - 1$. By the definition of resiliency, it means that after fixing $\mathsf{res}(f) - 1$ variables, $f_{i,\varepsilon}$ is not balanced anymore, hence fixing $\mathsf{res}(f) - 1 + 1 = \mathsf{res}(f)$ variables (the ones for $f_{i,\varepsilon}$ and the i-th variable to ε) of f gives an unbalanced function, which leads to a contradiction.

Then, we prove the bound on the AI. By contradiction we assume there exists i and ε such that $\mathsf{AI}(f_{i,\varepsilon}) < \mathsf{AI}(f) - 1$. Therefore there exists an annihilator of $f_{i,\varepsilon}$ (respectively $f_{i,\varepsilon} + 1$), g_i, non null, of degree lower than $\mathsf{AI}(f) - 1$. Then we take the function $h = (\varepsilon + x_i) \cdot g_i$ which is not null since, and of degree at lower than $\mathsf{AI}(f)$. Since $(\varepsilon + x_i)f = f_{i,\varepsilon}$ we have that h is an annihilator of f (respectively $f + 1$), leading to a contradiction. □

Moreover, for the nonlinearity we use a lower bound derived from the AI given by Lobanov [Lob09]:

Property 4 ([Lob09]). Let f be an n variable Boolean function, then the following bound applies on its nonlinearity:

$$\mathsf{NL}(f) \geq 2 \sum_{i=0}^{\mathsf{AI}(f)-2} \binom{n-1}{i}$$

We consider two kinds of functions for our tests in this part. First, we use TooLIP on functions which are the direct sum of a XOR function and an arbitrary function with high algebraic immunity and no resiliency. Since a random function have high algebraic immunity and low resiliency with high probability (see [Did06]), this test gives an idea on the number of variables necessary to have a secure instance of FiLIP by taking a random function and adding it in direct sum with a XOR function. Then, we consider a type of function having both optimal algebraic immunity and resilience, a dahu, whose existence is conjectured and investigated in [DMR21]. The advantage of using such function is that no direct sum is necessary with a XOR function to reach a high resiliency.

Definition 18 (Dahus, adapted from [DMR21] Definition 14). *Let $n \geq 3$, we call dahu an n-variable Boolean function such that:* $\mathsf{AI}(f) + \mathsf{res}(f) + 1 = n$ *and* $\mathsf{AI}(f) = \lfloor (n+1)/2 \rfloor$.

Setting λ to 128, we obtain the following result:

- Writing $f[\alpha, n]$ an arbitrary function f in n variables with $\mathsf{AI}(f) = \alpha$ and $\mathsf{res}(f) = 0$, we fulfill the security goal for a function $f[213, 450] + \mathsf{XOR}_{256}$ and key register size $N = 6000$. For the function $f[256, 512] + \mathsf{XOR}_{196}$ we obtain the same result since $N \geq 2500$.
- A dahu in 512 variables, that is to say with algebraic immunity 256 and resiliency 255 is a secure filter for $N = 2500$. For $n = 450$ the same result apply for $N \geq 3000$.

5 Conclusion and open questions

In this article, we discussed new instances for FiLIP by studying the security of different families of filter functions. To automate this process, we propose a tool called TooLIP that runs several existing attacks on user defined functions. While the functions studied in this article are already implemented in the tool, it should be simple for an user to add new ones as long as computing (bounds on) the criteria and the descendant is feasible. In particular, we focused on XOR-THR-THR and XOR-QUAD-THR which gave instances that are more efficient in terms of bandwidth and/or evaluation time when FiLIP is used in an HHE protocol. We also provided results for arbitrary filters in case the function used as some unknown properties. These different contributions enable to easily update the state of the art on the attacks, and determine secure instances in new families of functions that become more homomorphic friendly.

Two main open questions arose during this work. First, in [MCJS19b] various modifications are performed on the algorithm to determine instances with DSM functions since the number of descendants prevents to use the generic algorithms. It would be interesting to see if similar improvements could be implemented generically with the tool, such as simplifying the graph of descendants by merging sub-paths when descendants with worse properties are identified. Second, the throughput of FiLIP is less competitive than its latency compared to other symmetric ciphers used in HHE due to the fact that it produces its keystream bit by bit. A multi-bit output version of FiLIP could have a better throughput, and it raises the question of the tool to be adaptable to such generalizations.

Acknowledgments. The authors were supported by the ERC Advanced Grant no.787390.

References

ACG+06. Frederik Armknecht, Claude Carlet, Philippe Gaborit, Simon Künzli, Willi Meier, and Olivier Ruatta. Efficient computation of algebraic immunity for algebraic and fast algebraic attacks. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*. Springer, Heidelberg, 2006.

AL16. Benny Applebaum and Shachar Lovett. Algebraic attacks against random local functions and their countermeasures. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*. ACM Press, 2016.

AL18. Benny Applebaum and Shachar Lovett. Algebraic attacks against random local functions and their countermeasures. *SIAM J. Comput.*, pages 52–79, 2018.

APS15a. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

APS15b. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. Cryptology ePrint Archive, Paper 2015/046, 2015.

ARS+15. Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Heidelberg, 2015.

BIP+22. Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. FINAL: faster FHE instantiated with NTRU and LWE. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022*, volume 13792, pages 188–215, 2022.

BY03. Mihir Bellare and Bennet S. Yee. Forward-security in private-key cryptography. In Marc Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 1–18. Springer, Heidelberg, 2003.

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

CDM+18. Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux, Mélissa Rossi, and Yann Rotella. On the concrete security of goldreich's pseudorandom generator. In *ASIACRYPT 2018, Part I*, LNCS. Springer, Heidelberg, 2018.

CDPP22. Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder V.L. Pereira. Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In *ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 563–577, 2022.

CGGI16. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 3–33. Springer, Heidelberg, 2016.

CHK+21. Jihoon Cho, Jincheol Ha, Seongkwang Kim, Byeonghak Lee, Joohee Lee, Jooyoung Lee, Dukjae Moon, and Hyojin Yoon. Transciphering framework for approximate homomorphic encryption. In *ASIACRYPT*, 2021.

CHMS22. Orel Cosseron, Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. Towards case-optimized hybrid homomorphic encryption - featuring the elisabeth stream cipher. In *Advances in Cryptology - ASIACRYPT*, pages 32–67, 2022.

CIR22. Carlos Cid, John Petter Indrøy, and Håvard Raddum. Fasta – a stream cipher for fast fhe evaluation. In *Topics in Cryptology – CT-RSA 2022*, pages 451–483, 2022.

CLT14. Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In *Public-Key Cryptography – PKC 2014*, pages 311–328, 2014.

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, 2003.

CM19. Claude Carlet and Pierrick Méaux. Boolean functions for homomorphic-friendly stream ciphers. *Algebra, Codes and Cryptology*, pages 166–182, 11 2019.

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.

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, 2003.

Cou03b. Nicolas Courtois. Higher order correlation attacks, XL algorithm and cryptanalysis of toyocrypt. In Pil Joong Lee and Chae Hoon Lim, editors, *ICISC 02*, volume 2587 of *LNCS*. Springer, Heidelberg, 2003.

CT19. Benoît Cogliati and Titouan Tanguy. Multi-user security bound for filter permutators in the random oracle model. *Des. Codes Cryptogr.*, 87(7):1621–1638, 2019.

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.

DGH+21. Christoph Dobraunig, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Pasta: A case for hybrid homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 731, 2021.

Did06. F. Didier. A new upper bound on the block error probability after decoding over the erasure channel. *IEEE Transactions on Information Theory*, 52(10):4496–4503, 2006.

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, 2016.

DM15. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, pages 617–640, 2015.

DMR21. Aurélien Dupin, Pierrick Méaux, and Mélissa Rossi. On the algebraic immunity - resiliency trade-off, implications for goldreich's pseudorandom generator. *IACR Cryptol. ePrint Arch.*, page 649, 2021.

Fau99. Jean-Charles Faugère. A new efficient algorithm for computing groebner bases. *Journal of Pure and Applied Algebra*, 139:61–88, june 1999.

Fau02. Jean-Charles Faugère. A new efficient algorithm for computing Grobner bases without reduction to zero. In *Workshop on application of Groebner Bases 2002*, Catania, Spain, 2002.

GHS12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417, pages 850–867, 2012.

Gol00. Oded Goldreich. Candidate one-way functions based on expander graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(90), 2000.

HKC+20. Jincheol Ha, Seongkwang Kim, Wonseok Choi, Jooyoung Lee, Dukjae Moon, Hyojin Yoon, and Jihoon Cho. Masta: An he-friendly cipher using modular arithmetic. *IEEE Access*, 8:194741–194751, 2020.

HKL+22. Jincheol Ha, Seongkwang Kim, Byeonghak Lee, Jooyoung Lee, and Mincheol Son. Rubato: Noisy ciphers for approximate homomorphic encryption. In *EUROCRYPT*. Springer-Verlag, 2022.

HL20. Phil Hebborn and Gregor Leander. Dasta - alternative linear layer for rasta. *IACR Trans. Symmetric Cryptol.*, 2020(3):46–86, 2020.

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*, volume 12578 of *LNCS*, pages 39–61. Springer, 2020.

HMS23. Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. The patching landscape of elisabeth-4 and the mixed filter permutator paradigm. *IACR Cryptol. ePrint Arch.*, page 1895, 2023.

LN14. Tancrède Lepoint and Michael Naehrig. A comparison of the homomorphic encryption schemes fv and yashe. In David Pointcheval and Damien Vergnaud, editors, *Progress in Cryptology – AFRICACRYPT 2014*, pages 318–335, Cham, 2014. Springer International Publishing.

Lob09. M. S. Lobanov. Exact relations between nonlinearity and algebraic immunity. *Journal of Applied and Industrial Mathematics*, 3(3):367–376, Jul 2009.

MCJS19a. 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.

MCJS19b. Pierrick Méaux, Claude Carlet, Anthony Journault, and François-Xavier Standaert. Improved filter permutators for efficient FHE: better instances and implementations. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *Progress in Cryptology - INDOCRYPT*, volume 11898 of *LNCS*, pages 68–91. Springer, 2019.

Méa22. Pierrick Méaux. On the algebraic immunity of direct sum constructions. *Discrete Applied Mathematics*, 320:223–234, 2022.

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

MPC04. Willi Meier, Enes Pasalic, and Claude Carlet. Algebraic attacks and decomposition of boolean functions. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 474–491, Berlin, Heidelberg, 2004. 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.

NLV11. Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *CCSW*, pages 113–124. ACM, 2011.

Üna23. Akın Ünal. Worst-case subexponential attacks on prgs of constant degree or constant locality. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 25–54, 2023.

YGJL22. Jing Yang, Qian Guo, Thomas Johansson, and Michael Lentmaier. Revisiting the concrete security of goldreich's pseudorandom generator. *IEEE Transactions on Information Theory*, 68(2):1329–1354, 2022.