IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2025, No. 3, pp. 745–780. DOI:10.46586/tches.v2025.i3.745-780

Secure and efficient transciphering for FHE-based MPC

Diego F. Aranha[1], Antonio Guimarães[2], Clément Hoffmann[3] and Pierrick Méaux[4]

[1] Aarhus University, Aarhus, Denmark.
dfaranha@cs.au.dk
[2] IMDEA Software Institute, Madrid, Spain.
antonio.guimaraes@imdea.org
[3] NTT Social Informatics Laboratories, Tokyo, Japan†.
clement.hoffmann@ntt.com
[4] University of Luxembourg, Esch-sur-Alzette, Luxembourg
pierrick.meaux@uni.lu

Abstract. Transciphering (or Hybrid-Homomorphic Encryption, HHE) is an established technique for avoiding ciphertext expansion in HE applications, saving communication and storage resources. Recently, it has also been shown to be a fundamental component in the practical construction of HE-based multi-party computation (MPC) protocols, being used both for input data and intermediary results (Smart, IMACC 2023). In these protocols, however, ciphers are used with keys that are jointly generated by multiple (possibly malicious) parties, which may require additional security assumptions that have been so far overlooked in the HHE literature. In this paper, we formalize this issue as a security against related-key attacks (RKA) problem and provide efficient solutions for it. We start by presenting an efficient method for homomorphically evaluating Mixed-Filter-Permutator (MFP) ciphers in leveled mode, enabling speedups of up to thousands of times compared to previous literature. For the multi-party scenario, we focus specifically on the Margrethe cipher (Hoffmann *et al.*, INDOCRYPT 2023). We show that, contrary to other commonly used HHE ciphers (e.g. FLIP), Margrethe is out-of-the-box secure for any protocols that allow malicious parties to learn up to two related key streams, enabling security for the vast majority of static MPC protocols. For other cases, we quantify the loss of security based on the number of related key streams (which often depends on the number of malicious parties and specific protocol). Performance-wise, our implementation of Margrethe takes just 3.9 ms to transcipher 4-bit messages, being significantly faster than the state of the art in terms of latency.

Keywords: Transciphering, Related Key Attacks, Fully Homomorphic Encryption, FHE-based MPC

1 Introduction

Among the many challenges for making Fully Homomorphic Encryption (FHE) practical, minimizing ciphertext expansion (*i.e.*, the ratio between the size of a ciphertext and the message it encrypts) appears as a particularly interesting problem. For some applications, it is a relevant but ultimately optional concern; for many others, it represents a fundamental challenge in their practicability. FHE-based multi-party computation (MPC) is a prime

†This work was mostly conducted while the author was working in UCLouvain, Louvain-la-Neuve, Belgium

Received: 2025-01-15 Accepted: 2025-03-15 Published: 2025-06-05

example of the latter, as the core idea behind it is precisely to avoid the high communication costs incurred by traditional MPC approaches. In this context, large ciphertext expansion may entirely undermine its purpose by making it more communication-intensive than alternative solutions.

Transciphering [NLV11] (or Hybrid Homomorphic Encryption, HHE) is an established solution for eliminating ciphertext expansion in HE schemes. It consists of encrypting data using a symmetric cipher that can be later homomorphically evaluated to recover data in an FHE ciphertext. In this way, data can be stored and communicated without any expansion, and the ciphertext only expands at processing time. For FHE-based MPC, this represents the ideal solution, as the impact of ciphertext expansion is completely negated. On the other hand, the employment of symmetric ciphers in multi-party protocols is not as straightforward as one may assume based on their use in general FHE applications.

Multi-party protocols require HHE ciphers to be used with keys that are jointly generated by different (possibly malicious) parties, which exposes them to possible vulnerabilities that are rarely considered in their design. Particularly, if an adversary learns outputs of a cipher that are generated by different but somewhat related secret keys, it can perform Related-Key Attacks (RKA), which may lead to efficient key recovery depending on the specific cipher and key relations. RKA security is so far a completely unstudied topic in HHE literature, but similar attacks have been demonstrated for popular HHE ciphers. The closest examples are differential fault attacks [RBM20, RKMR23, MR24] against FLIP [MJSC16], Kreyvium [CCF+18], Rasta [DEG+18], and FiLIP [MCJS19], which, as we show in this work, could be performed as an RKA without requiring fault injection.

1.1 Overview and contributions

Let $\mathsf{Sym} : \mathbb{K} \times \mathbb{Z} \mapsto \{0,1\}^\kappa$ be a stream cipher that receives a key K in some key space $\mathbb{K}$ and initialization vector (IV) z, producing a κ-bit keystream, and let $\Phi : \cdots \mapsto \mathbb{K}$ be some Key Derivation Function (KDF). The output of two instances $\mathsf{Sym}(k_i, z)$ and $\mathsf{Sym}(k_j, z')$ are *related keystreams* if one key is a function of the other, *i.e.* $k_i = \Phi(k_j, \dots)$. In the context of FHE-based MPC, key derivations are given by the combination of key shares. Let $\Phi : \mathbb{K} \times \mathbb{K} \times \cdots \times \mathbb{K} \mapsto \mathbb{K}$ be the KDF performing the combination of key shares to generate a joint transciphering key, related keystreams (RKS) are said to have *known relations* if the difference between the inputs of different instances of Φ are known to the adversary. For example, $\mathsf{Sym}(\Phi(k_i), z)$ and $\mathsf{Sym}(\Phi(k_i, k_j), z)$ are RKS with known relations to anyone knowing k_j. These are the relations that be exploited in an RKA.

In this work, we study the security implications of using transciphering in multi-party protocols in the context of an RKA and propose an efficient (and secure) solution by showing that the Margrethe cipher [HMS23] is resistant against RKA for specific settings that cover most FHE-based MPC protocols. We focus on protocols based on threshold-FHE [AJLA+12] to simplify parameter choices in comparison to Multi-key FHE [CCS19].

Transciphering in FHE-based MPC as an RKA problem. Our first contribution is to define how operations in FHE-based MPC can produce related key streams (RKS), enabling its cryptanalysis at the symmetric cryptography level. For most MPC protocols, and especially those with n fixed parties in the static setting (where parties do not change), only two RKS are produced:

1. Each party $\mathcal{P}_i$ encrypts their input using a key stream $\mathsf{Sym}(\Phi(k_i), z_i)$ produced with their key k_i.

2. All parties jointly generate $\mathsf{Sym}(\Phi(k_1, \dots, k_n), z)$ to store encrypted intermediate results through transciphering operations [Sma23].

This work focus on providing a secure and efficient transciphering solution for this setting, which we refer to as *two-RKS* MPC.

Note that some "trivial" countermeasures could be admissible to prevent the problem altogether. For example, one could simply use different keys at every instance of the KDF, completely avoiding the generation of RKS. However, this complicates key management, increases communication costs, and introduces additional expensive setup phases. Considering this, our goal in this work is to solve vulnerabilities that arise from the use of transciphering in a multi-party protocol within the cipher specification itself, enabling its use as a black-box component instead of relying on protocol-level MPC tools to achieve security.

RKA-security in transciphering and Margrethe The core of our proposal is simply to employ a stream cipher that is resistant against RKA for the specific MPC setting we consider (two-RKS setting). However, RKAs have not been considered in the design of most transciphering solutions, and many popular ciphers (*e.g.* FLIP) are vulnerable to somewhat similar attacks, as we discuss in Section 4.4. These vulnerabilities are not present in the Margrethe cipher [HMS23], making it a strong candidate for the MPC protocol we propose. Additionally, Margrethe's homomorphic implementation leverages state-of-the-art techniques, enabling efficient, low-latency evaluation.

Our second contribution is the security analysis of Margrethe in the two-RKS scenario, demonstrating that it continues to provide 128-bit security. We use a novel approach that links the properties of the first-order derivatives of Boolean functions to the RKA security of the scheme. This allows us to conduct the security analysis with arguments similar to those in [HMS23], providing concrete security estimates. Furthermore, we explore the connection between security in the two-RKS setting and a model of differential fault attacks previously considered in the context of stream ciphers. As a side contribution, we provide an efficient implementation[1] for computing Boolean function properties required for estimating the complexity of the considered attacks against the cipher. We note this could also be generally useful in other contexts (for example, in applications such as leakage profiling [CDSU23, BED+22]).

Efficient evaluation of Margrethe. Margrethe is one of the latest developments in the field of Mixed Filter Permutator (MFP) ciphers, but it features very large lookup tables (LUT), making its homomorphic evaluation challenging for current evaluation approaches adopted for MFP, which are based on bootstrapped evaluation [CHMS22] or leveled evaluation of binary gates [CDPP22]. Our third contribution is the introduction of a generic method for evaluating MFP ciphers in leveled mode, and to present the *first* implementation of Margrethe[2]. Compared with previous literature, our implementation is up to 1985 times faster than FRAST [CCH+24] in terms of latency while offering 62 times higher bit throughput. Following the plaintext-independent transciphering approach of [MPP23], we show our method can be used to generate ciphertexts with plaintext space of up to 26 bits. Compared with [MPP23], we achieve up to a 21.9 times speedup for 8-bit messages, which is the maximum size enabled by their current parameters. We also propose and benchmark a key mixing function for Margrethe to enable the efficient generation of joint secret keys in multi-party protocols. Our key mixing function runs in just 430 ms single-threaded.

Beyond two-RKS MPC. In dynamic MPC protocols in which parties may join and leave at any point, it may be necessary to use many different combinations of keys during computation. This would generate a number of RKS that is exponential in the number

[1] Available at https://github.com/Cirthy/margrethe_rka_scripts
[2] Available at https://github.com/antoniocgj/MARGRETHE

of parties, with the number of known relations (to the adversary) being exponential in the number of possibly malicious (and colluding) parties. We refer to such protocols as *many-RKS* MPC. Margrethe, per se, is not secure in this setting. As a final contribution, we investigate its security and demonstrate it degrades rapidly. Specifically, we show that numerous related keys forming an affine subset of high dimension allow an adversary to derive equations corresponding to high-order derivatives of the filter function. This results in solvable equations involving the key bits. As future work, we present a few possible directions for addressing this setting, noting that, so far, most FHE-based MPC protocols are limited to the static setting.

2 Preliminaries

2.1 Fully Homomorphic Encryption

While the ultimate goal of this work is to provide solutions for multi-party protocols, we need an efficient transciphering implementation as an initial step. Considering that, we implement the Margrethe cipher [HMS23] using techniques from TFHE [CGGI16]. For most of this work, we can treat procedures regarding HE encryption, decryption, and evaluation as black-box functions. We provide details when necessary.

TFHE. Let $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$ be a polynomial ring, and given a set of parameters $(N, p, q, \sigma, \ell, \beta)$, TFHE defines three types of ciphertexts, denoted by:

- $c \in \mathrm{LWE}_s(m) \subset \mathbb{Z}_q^{N+1}$ an LWE sample encrypting some message $m \in \mathbb{Z}_p$ under a secret key $s \in \mathbb{Z}_q^N$.
- $c \in \mathrm{RLWE}_s(m) \subset \mathcal{R}_q^2$ an RLWE sample encrypting some message $m \in \mathcal{R}_p$ under a secret key $s \in \mathcal{R}_q$
- $C \in \mathrm{RGSW}_s(m) \subset \mathcal{R}_q^{2\times 2\ell}$ an RGSW sample encrypting some message $m \in \mathcal{R}_p$ under a secret key $s \in \mathcal{R}_q$. An RGSW C is defined as a vector of 2ℓ RLWE samples such that the i-th RLWE sample c_i encrypts $m\beta^{i-\ell}$ for $i \in [\![\ell, 2\ell)$ and $-sm\beta^i$ for $i \in [\![0, \ell)$.

Arithmetic. When using TFHE, one typically only performs additions between (R)LWE samples and multiplications by cleartext constants. Multiplications between (R)LWE samples, although possible, are avoided. For RGSW samples, on the other hand, it defines two types of products:

- External product: Given a sample $A \in \mathrm{RGSW}_s(m_0)$ and a sample $b \in \mathrm{RLWE}_s(m_1)$, the external product, denoted by $A \boxdot b$, returns a sample $c \in \mathrm{RLWE}_s(m_0 m_1)$, where $m_0 m_1$ is computed in the plaintext space of b.
- Internal product: Given a sample $A \in \mathrm{RGSW}_s(m_0)$ and a sample $B \in \mathrm{RGSW}_s(m_1)$, it produces a sample $C \in \mathrm{RGSW}_s(m_0 m_1)$, where $m_0 m_1$ is computed in the plaintext space of B.

Notice that RGSW products can be performed for RGSW samples with different decomposition parameters (ℓ and β), but the same does not occur in additions.

Lookup table (LUT) evaluation. TFHE defines two main methods for evaluating arbitrary functions represented as lookup tables (LUTs):

- Vertical Packing (VP): Given a k-bit LUT L in $\mathbb{Z}_p^{2^k}$ representing a function f and a vector C of k RGSW samples encrypting bit by bit some message m such that $C_i \in \mathrm{RGSW}_s(m_i)$ for $m = \sum_{i=0}^{k-1} m_i 2^i$, the vertical packing, VP(C, L), computes $c \in \mathrm{LWE}_s(f(m))$.

- Programmable Bootstrapping (PBS): Given a k-bit LUT L in $\mathbb{Z}_p^{2^k}$ representing a function f and an LWE sample $c \in \mathrm{LWE}_s(m)$, the PBS computes $\overline{c} \in \mathrm{LWE}_s(f(m))$.

Notice that while both procedures achieve similar functionality, the PBS operates entirely over LWE samples, whereas the VP receives RGSW samples as input and produces LWE ones as output. Considering this, it would be necessary to convert LWE samples back to RGSW to use the VP in a composable way. This process is a *circuit bootstrapping* [CGGI17] and, despite recent advances [WWL+24, WLW+24], it would still add significant overhead to the computation of the VP.

2.2 FHE-based Multi-Party Computation

Modern FHE schemes such as TFHE have become efficient enough that they can be deployed in real-world applications, as long as there is a way to alleviate the cost of storing and transmitting large FHE ciphertexts. They also support efficient distributed decryption (Threshold-FHE) that can be used to implement *declassify* operations, an analogue of the basic "opening" operation in MPC.

Smart [Sma23] provides an FHE-based MPC protocol that satisfies four properties: (i) *robustness*, in the sense of security in the malicious setting while providing guaranteed output delivery to honest parties; (ii) *composability* with other concurrent protocols, implying security in the UC framework; (iii) *reactive computation*, such that persistent state is maintained across several calls, and parties may provide (private) inputs and receive (private) outputs over multiple rounds; (iv) *communication efficiency* to avoid sending large FHE ciphertexts, employing transciphering as a building block to reduce storage/bandwidth requirements.

Let $\mathbb{P}$ be a set of n parties $\mathcal{P}_i$ which provide inputs, outputs or perform computation ($\mathbb{I} = \mathbb{C} = \mathbb{O}$ in [Sma23]); and let Sym represent an FHE-friendly stream cipher, such as Margrethe. In the *static* setting, all parties are required to commit to the entire duration of the protocol, limiting flexibility. In the *dynamic* setting, parties can go offline and rejoin later, when their resources become available again.

We briefly summarize the overall MPC-FHE protocol in the static setting, and more detailed descriptions of the relevant subprotocols can be found in Section B in the supplementary material:

- *Key generation* is realized with a distributed protocol that secret-shares the FHE secret key among the parties using a (t, n)-linear secret-sharing scheme. For efficiency, the original paper operates in an honest supermajority and tolerates $t < n/3$ corruptions [DDK+23]. We generalize this to a dishonest majority setting where $t < n$, and consider that all parties are needed for decryption. Parties $\mathcal{P}_i$ generate random symmetric keys k_i for Sym, encrypt them with FHE, and broadcast the ciphertexts $\mathfrak{ct}_i$ to all other parties, together with zero-knowledge proofs π_i of correct encryption. If the proofs verify, a joint transciphering key $k_0 = \sum_{i=1}^{n} k_i$ can be obtained by homomorphically combining the ciphertexts as $\mathfrak{ct}_0 = \sum_{i=1}^{n} \mathfrak{ct}_i$. This initialization step is an expensive one-time setup, amortized by the following MPC computation.

- *Inputting* a private value x by party $\mathcal{P}_i$ amounts to encrypting x under Sym and its symmetric key k_i to obtain $\mathbf{c}_x$, which is then broadcast to all parties and transciphered into an encryption $\mathfrak{ct}_x$ under the FHE scheme. This is achieved by each

party homomorphically evaluating decryption with Sym over $\mathbf{c}_x$ with the encryption $\mathfrak{ct}_i$ of k_i, thus obtaining x encrypted under FHE alone.

- *Outputting* a value y to a party $\mathcal{P}_j$ amounts to transciphering the ciphertext under FHE back to Sym using homomorphic evaluation with the FHE-encrypted key k_j and executing distributed decryption to decrypt from FHE to a ciphertext encrypted under Sym alone. The latter can be decrypted to y with key k_j to recover the plaintext.
- The MPC *computation* is realized by evaluating addition and multiplication gates using the native homomorphic operations supported by the FHE scheme.
- *Declassifying* data can be performed by calling the FHE distributed decryption algorithm.
- *Transciphering* operations from FHE to the symmetric cipher to transmit or store intermediate data is performed by homomorphically evaluating Sym under the joint transciphering key k_0, followed by the FHE distributed decryption. The reverse transciphering to recover an encryption under FHE alone is achieved by evaluating decryption with Sym under k_0.

Generating two-RKS. The MPC-FHE protocol produces two-RKS relations in the static setting under dishonest majority with $n-1$ malicious parties, when a particular set of steps is executed in the protocol:

1. Without loss of generality, assume a single honest party $\mathcal{P}_1$ that inputs a value x using symmetric encryption Sym. Assume further that this value is predictable because it was sampled from a non-uniform distribution, or that it is later output and revealed to another malicious participant.
2. Due to the broadcast, this means that an encryption $\mathbf{c}_x$ of (known value) x under a keystream computed from k_1 is now available to all participants. As part of the input protocol, a transciphering $\mathfrak{ct}_x$ under FHE will also be produced.
3. The participants transcipher $\mathfrak{ct}_x$ from FHE to a symmetric encryption by evaluating Sym under encrypted key $\mathfrak{ct}_0$. This ends up revealing an encryption of x under a keystream computed from k_0.
4. Because $k_0 = k_1 + \sum_{i=2}^{n} k_i$, this generates a two-RKS for which the difference $\delta = \sum_{i=2}^{n} k_i$ is the addition of all key shares belonging to malicious parties. In other words, the two-RKS are $\mathsf{Sym}(\Phi(k_1), z)$ and $\mathsf{Sym}(\Phi(k_1, \ldots, k_n), z')$.

Even though the MPC-FHE protocol was presented in the static setting, which forces participants to commit to their symmetric key shares in the setup phase, it can be extended to the dynamic setting and support participant churn by re-sharing keys among the active number of parties. This gives more opportunities to malicious parties to provide their own malicious contributions as key shares, which is a particular concern for transciphering operations involving different sets of active participants across time.

2.3 Boolean functions in cryptography

Analyzing the security of MPF ciphers against related key attacks requires some key concepts on Boolean functions used in cryptography. We recall them in this section. For a deeper introduction and more context on these concepts, we refer to the book [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 (Algebraic Normal Form (ANF) and degree)**.** 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 equals the global degree of its ANF: $\mathsf{deg}(f) = \max_{\{I \mid a_I = 1\}} |I|$ (with the convention that $\mathsf{deg}(0) = -\infty$).

Definition 3 (Algebraic Immunity (AI))**.** The Algebraic Immunity (AI) of a Boolean function $f \in \mathcal{B}_n$, denoted as $\mathsf{AI}(f)$, is defined as:

$$\mathsf{AI}(f) = \min_{g \neq 0} \{\mathsf{deg}(g) \mid fg = 0 \text{ or } (f+1)g = 0\},$$

where $\mathsf{deg}(g)$ is the algebraic degree of g.

Definition 4 (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 between f and all the affine functions in $\mathcal{B}_n$:

$$\mathsf{NL}(f) = \min_{g,\, \mathsf{deg}(g) \leq 1} \{\mathsf{w_H}(f + g)\},$$

where $g(x) = a \cdot x + \varepsilon$, $a \in \mathbb{F}_2^n, \varepsilon \in \mathbb{F}_2$, and $\mathsf{w_H}(f + g)$ is the Hamming distance between the truth table of the functions f and g.

2.3.1 Symmetric functions.

n-variable Boolean symmetric functions are those Boolean functions constant on each slice $\mathsf{E}_{k,n} = \{x \in \mathbb{F}_2^n \mid \mathsf{w_H}(x) = k\}$ for $k \in [0, n]$, where $\mathsf{w_H}(x)$ denotes the Hamming weight of x.

Definition 5 (Elementary symmetric functions)**.** Let $i \in [0, n]$, the elementary symmetric function of degree i in n variables, denoted $\sigma_{i,n}$, is the function which ANF contains all monomials of degree i and no monomials of other degrees.

Property 1 (Properties of elementary symmetric functions)**.** *Let $n \in \mathbb{N}^*$, and $d \in [0, n]$:*

- *The function $\sigma_{d,n}$ takes the value $\binom{k}{d} \mod 2$ on the elements of the slice $\mathsf{E}_{k,n}$.*

Property 2 (Lucas' Theorem)**.** *Let $a, b, p \in \mathbb{N}$ be integers such that $a > b$ and p is a prime. Consider their p-adic expansions $a = \sum_{j=0}^{q} a_j p^j$ and $b = \sum_{j=0}^{q} b_j p^j$ such that $0 \leq a_j < p$ and $0 \leq b_j < p$ for each $j \in [0, q]$ and $a_q \neq 0$. Then*

$$\binom{a}{b} \equiv \prod_{j=0}^{q} \binom{a_j}{b_j} \pmod{p}.$$

Definition 6 (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:

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

Lemma 1 (Direct sum Properties (e.g. [MJSC16] Lemma 3))**.** *Let $h = \mathsf{DS}(f, g)$ be the direct sum of f and g n and m-variable Boolean functions respectively. Then $\mathsf{DS}(f, g)$ has the following cryptographic properties:*

1. *Degree:* $\mathsf{deg}(h) = \max(\mathsf{deg}(f), \mathsf{deg}(g))$.

2. *Algebraic immunity:* $\max(\mathsf{AI}(f), \mathsf{AI}(g)) \leq \mathsf{AI}(h) \leq \mathsf{AI}(f) + \mathsf{AI}(g)$.

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

Lemma 2 ([Méa22] Lemma 5)**.** *Let $n, m \in \mathbb{N}^*$, f and g be Boolean functions in n and m variables. If $\mathsf{AI}(f) < \mathsf{deg}(g)$ then $\mathsf{AI}(\mathsf{DS}(f, g)) > \mathsf{AI}(f)$.*

Lemma 3 ([Méa22] Lemma 6)**.** *Let $t \in \mathbb{N}^*$, and $f_1, \ldots, f_t$ be t Boolean functions, if for $r \in [t]$ there exists r different indexes $i_1, \cdots, i_r$ of $[t]$ such that $\forall j \in [r], \mathsf{deg}(f_{i_j}) \geq j$ then $\mathsf{AI}(\mathsf{DS}(f_1, \ldots, f_t)) \geq r$.*

The three lemmas above are used to bound the Boolean properties of MFP's filter function, from which concrete security estimates are derived. Specifically, the filter function is expressed as the direct sum of t instances of a function computed using a look-up table. This allows us to exhaustively compute the required properties for the (relatively small) functions derived from this block function. Subsequently, we apply Lemma 1 to bound the non linearity of the bigger filter function, and Lemma 3 to bound the algebraic immunity of the full filter function. In order to prove an AI of r using Lemma 3, the idea is to split the direct sum into different terms such that at least one has a degree of (at least) r, at least another one as (at least) degree $r - 1$, etc.

Definition 7 (Derivative)**.** Let $f \in \mathcal{B}_n$ and $a \in \mathbb{F}_2^n$, we call (first-order) derivative in the direction a (or with input difference a) of f the Boolean function:

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

Similarly, let $a^1, \ldots, a^t$ be t different elements of $\mathbb{F}_2^n$, we call t-th order derivative of f in direction $a^1, \ldots, a^t$:

$$\mathsf{D}_{a^1,\ldots,a^t} f(x) = \mathsf{D}_{a^1,\ldots,a^{t-1}} f(x) + \mathsf{D}_{a^1,\ldots,a^{t-1}} f(x + a^t) = \sum_{b \in \mathbb{F}_2^t} f(x + \sum_{i=1}^{t} b_i a^i).$$

Property 3 (Derivatives and properties)**.** *Let $f \in \mathcal{B}_n$ and $a_1, \ldots, a_t \in \mathbb{F}_2^n$, the following hold on the derivative of f:* $\mathsf{deg}(\mathsf{D}_{a_1,\ldots,a_t} f) \leq \mathsf{deg}(f) - t$.

Let h be the direct sum of f and g in n and m variables respectively, for any $a \in \mathbb{F}_2^n$ and $b \in \mathbb{F}_2^m$ the derivative of $h(x, y) = f(x) + g(y)$ in $(a, b) \in \mathbb{F}_2^{n+m}$ is given by:

$$\mathsf{D}_{(a,b)} h = f(x) + f(x + a) + g(y) + g(y + b) = \mathsf{D}_a f + \mathsf{D}_b g.$$

2.4 Mixed Filter Permutator and the Margrethe cipher

The Mixed Filter Permutator (MFP) is a state-of-the-art paradigm for building FHE-friendly stream ciphers that can be evaluated with low latency. It is defined by two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ with operation noted $+_1$ and $+_2$, a forward secure PseudoRandom Number Generator (PRNG), a key size N, a subset size n, and a filtering function f from $\mathbb{G}_1^n$ to $\mathbb{G}_2$. To encrypt m elements of $\mathbb{G}_2$ under a secret key $K \in \mathbb{G}_1^N$, first, the user chooses the public parameters of the PRNG, then the following process is executed for each keystream s_i (for $i \in [m]$):

- The PRNG is updated, its output is used to select a subset, a permutation, and a length-n vector of $\mathbb{G}_1$.
- the n-element subset S_i is chosen over N-element key,
- the n to n permutation P_i is chosen,
- the vector, called whitening and denoted w_i, from $\mathbb{G}_1^n$ is chosen,

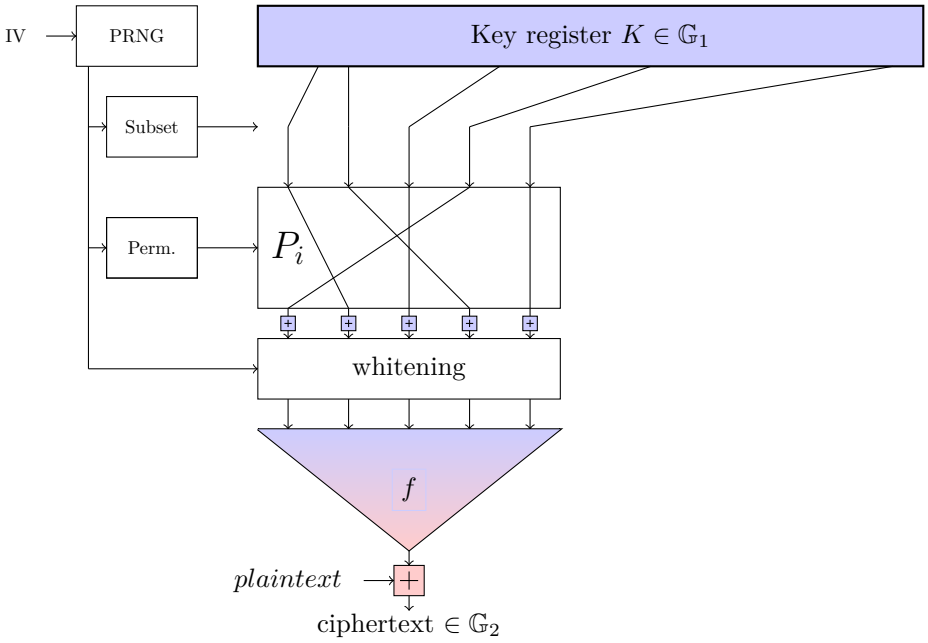

Figure 1: The MFP design

- the key stream element s_i is computed as $s_i = f(P_i(S_i(K)) +_1 w_i)$, where $+_1$ denotes the element-wise addition of $\mathbb{G}_1$.

We denote:

- K the key in $\mathbb{G}_1^N$,
- S_i an n out of N subset,
- P_i a (wire-cross) permutation from $\mathbb{S}_n$,
- ω_i a vector of $\mathbb{G}_1^n$.

2.4.1 Margrethe.

Margrethe is the stream cipher family following the MFP presented in [HMS23]. It is characterized by the use of $\mathbb{F}_2$ for $\mathbb{G}_1$, and the group $\mathbb{Z}_{2^\ell}$ with $\ell \in \mathbb{N}^*$ for $\mathbb{G}_2$. The filter function is obtained by the direct sum over $\mathbb{G}_2$ of t times a function obtained by a look-up table. Margrethe-a-b denotes the instance associated with a Look-Up Table (LUT) of size from a bits to b bits. The instance proposed in [HMS23], we use in this article is Margrethe-18-4, given by the following parameters:

- $\mathbb{G}_1 = \mathbb{F}_2$,
- $\mathbb{G}_2 = \mathbb{Z}_{16}$,
- $N = 2048$,
- the filter function is the function f from $\mathbb{F}_2^n$ to $\mathbb{Z}_{16}$, that uses the inner function g, a function from $\mathbb{F}_2^{18}$ to $\mathbb{Z}_{16}$.

  $g : \mathbb{F}_2^{18} \mapsto \mathbb{Z}_{16}$ is given by a LUT.

  $f : \mathbb{F}_2^{308} \mapsto \mathbb{Z}_{16}$ is defined by:

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

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

The LUT has been generated from the character string "Welcome to Margrethe" and using successive hashing with SHA256. We refer to [HMS23] Section 4 for the details.

3 Efficient homomorphic evaluation of MFP ciphers

The main challenge for homomorphic evaluating MFP ciphers is the filter function f, which is nonlinear and operates over very large inputs, typically with hundreds of bits. Current MFP constructions enable its efficient evaluation either by representing the cipher as a circuit of binary logic gates (*e.g.*, the case of FiLIP [MCJS19]) or by defining f such that all nonlinear sub-procedures only operate over small amounts of data. These relatively small sub-procedures can then be evaluated using techniques such as the programmable bootstrapping (PBS) (*e.g.*, as in the evaluation of Elisabeth [CHMS22] and Gabriel [HMS23] ciphers), which enable low latency implementations.

3.1 Leveled evaluation

It has been shown that the homomorphic evaluation of FiLIP can be significantly accelerated by working in leveled mode [CDPP22, MPP23]. However, these were very specific solutions, as FiLIP is the only MFP to have its filter function represented as a binary logic circuit. Extending their approach to other MFP, that typically work over $\mathbb{Z}_{16}$, although possible, would not present similar improvements, especially as [CDPP22] and [MPP23] rely on NTRU-based evaluation, which is only particularly well suited for binary computation. At first sight, the use of techniques such as the Vertical Packing (VP) is a natural solution for these other cases, as the VP can provide efficient integer evaluation in leveled mode. However, its efficient employment in other MFP ciphers is not straightforward either, since, despite providing flexibility for outputs, the vertical packing still presents strict requirements for its input ciphertexts. Particularly, it requires ciphertexts with low noise encrypting bit-decomposed inputs, which is not directly compatible with some arithmetic requirements of other parts of MFP ciphers, such as whitening and permutation functions. It is also very expensive to compose multiple consecutive evaluations of VP, as it requires inputs to be RGSW ciphertexts while it only outputs LWE ciphertexts. Converting from LWE to RGSW requires circuit bootstrappings, which, as discussed in Section 2.1, are expensive procedures.

3.1.1 Elisabeth

Let us start with the evaluation of Elisabeth 4 [CHMS22], which is one of the most well-known and complex MFP designs. Algorithm 1 shows it. Notice that, while the filter function is defined to work over $\mathbb{Z}_{16}^{60} \mapsto \mathbb{Z}_{16}$, all its non-linear sub-procedures are defined over $\mathbb{Z}_{16} \mapsto \mathbb{Z}_{16}$, requiring only 4 bits of precision, which enables its efficient evaluation using the programmable bootstrapping (PBS). The only straightforward improvement we could make for its evaluation is replacing the first layer of PBSs with Vertical Packings (VP). Replacing both layers would already require circuit bootstrappings, as we would be creating a chained evaluation of VPs. Our main observation at this point is that the entire evaluation can be equivalently represented as in Algorithm 2.

Algorithm 1: Original Elisabeth

Input : Encrypted secret key $k_i \in \mathrm{LWE}_s(sk_i\Delta)$, for $i \in [\![0, 256)$ and $sk \in \mathbb{G}^{256}$
Input : A set of LUTs $L \in \mathbb{G}^{8\times 16}$
Input : Bootstrapping and key switching keys
Output : Encrypted key stream

for $j \leftarrow 0$ **to** 60 **do**
  $r \leftarrow \texttt{rnd}(j, 256)$
  $w \leftarrow \texttt{rnd}(0, 16)$
  $\texttt{swap}(k_j, k_r)$
  $k_j \leftarrow k_j + (0, w\Delta)$
/* Filter */
$acc \leftarrow (0, 0)$
for $b \leftarrow 0$ **to** 11 **do**
  for $j \leftarrow 0$ **to** 4 **do**
    $x_j \leftarrow k'_{5b+j}$
  /* 1st layer */
  for $j \leftarrow 0$ **to** 3 **do**
    $y_j \leftarrow \texttt{PBS}(x_j + x_{(j+1) \mod 4}, L_j)$
  $r \leftarrow 0$
  /* 2nd layer */
  for $j \leftarrow 0$ **to** 3 **do**
    $t \leftarrow \texttt{KS}(y_{(j+1) \mod 4} + y_{(j+2) \mod 4})$
    $r \leftarrow r + \texttt{PBS}(x_j + t, L_{4+j})$
  $t \leftarrow \texttt{KS}(r)$
  $acc \leftarrow acc + t + x_4$
$stream_i \leftarrow acc$
return $stream_i$

Algorithm 2: Leveled Elisabeth

Input : Encrypted (bit by bit) secret key $k_{i,j} \in \mathrm{RGSW}_s(sk_{i,j})$, for $i \in [\![0, 256)$, $j \in [\![0, 4)$, and $sk \in \mathbb{G}^{256}$
Input : A large LUT $L \in \mathbb{G}^{2^{16}}$
Output : Encrypted key stream

for $j \leftarrow 0$ **to** 60 **do**
  $r \leftarrow \texttt{random\_int}(j, 256)$
  $w_j \leftarrow \texttt{random\_int}(0, 16)$
  $\texttt{swap}(k_j, k_r)$
/* Filter */
$acc \leftarrow (0, 0)$
for $b \leftarrow 0$ **to** 11 **do**
  for $j \leftarrow 0$ **to** 4 **do**
    $x_j \leftarrow k_{5b+j}$
    $w'_j \leftarrow w_{5b+j}$
  $L' \leftarrow \texttt{UpdateLUT}(L, w'_0, \ldots, w'_3)$
  $r \leftarrow \texttt{VP}(x_0|x_1|x_2|x_3, L')$
  $acc \leftarrow acc + r + \texttt{RGSWtoLWE}(x_4)$
$stream_i \leftarrow acc$
return $stream_i$

Proc. $\texttt{RGSWtoLWE}(x)$
  $L \leftarrow [0, 1, 2, \ldots, 15]$
  return $\texttt{VP}(x, L)$

Proc. $\texttt{UpdateLUT}(L, w_0, \ldots, w_3)$
  for $i \leftarrow 0$ **to** 2^{16} **do**
    for $j \leftarrow 0$ **to** 3 **do**
      $v_j \leftarrow i/2^{4j} + w_j \mod 16$
    $L'[i] \leftarrow L[v_0|v_1|v_2|v_3]$
  return L'

Essentially, we combine all 8 LUTs of size 16 used in the filter function of Elisabeth into a single larger LUT of size 2^{16}. In other words, we are replacing the evaluation of 8 functions with 4-bit precision with just one 16-bit function, which is possible because all 8 functions only depend on the same 4 4-bit variables (x_0, x_1, x_2, and x_3). By doing this modification, we eliminate the chain of nonlinear functions from the original Elisabeth, enabling its function to be evaluated with a single vertical packing per round. In practice, we replaced 8 4-bit PBSs, which used to take around 5-10 ms each, with a single 16-bit VP, which takes less than a millisecond to be evaluated. Furthermore, Algorithm 2 does not require any bootstrappings or key switching keys, and the noise generated by the evaluation of a VP is much smaller than the one generated by a PBS. It also has no requirement for negacyclic LUTs or padding bits, which, as pointed out by [CCH+24], is a constraint in the original Elisabeth.

Dynamic LUT update. Another problem of employing the VP in the homomorphic evaluation of Elisabeth is the compatibility between the arithmetic requirements of the VP, which works with inputs in $\mathbb{F}_2$, and the whitening, which works in $\mathbb{Z}_{16}$. It is possible to implement a binary logic circuit to emulate (bit by bit) $\mathbb{Z}_{16}$ arithmetic in $\mathbb{F}_2$, but it would

not only be expensive but also introduce noise to ciphertexts before the vertical packing. We solve this problem by dynamically recalculating the lookup table that represents the cipher's filter function to consider each specific whitening input. Let f be the filter function represented by some LUT L and w_i be the whitening values. In the case of Elisabeth, instead of directly evaluating $f(x_0 + w_0, x_1 + w_1, x_2 + w_2, x_3 + w_3)$ using L on each round, which would require performing homomorphic additions in $\mathbb{Z}_{16}$, we produce a new function $f'(x_0, x_1, x_2, x_3) = f(x_0 + w_0, x_1 + w_1, x_2 + w_2, x_3 + w_3)$ represented by a new LUT L'. Producing L' from L also requires additions in $\mathbb{Z}_{16}$, but these are cleartext additions, which are inexpensive and noiseless. The resulting function f' and its LUT representation L' have the same size as the original one, and can be evaluated at the same cost. This process is described in detail in the procedure `UpdateLUT` of Algorithm 2.

RGSW to LWE conversion. A similar problem occurs in the addition between the output of VP, which is an LWE sample, and x_4 (Line 17 of Algorithm 1), which is now a vector of RGSW samples. We could just keep multiple representations of the key, but converting RGSW to LWE is an inexpensive procedure, which we can evaluate with a very small instance of the vertical packing over a LUT encoding the identity function. This is represented by the Procedure `RGSWtoLWE` in Algorithm 2.

3.1.2 Margrethe

From this point, the evaluation of Margrethe is natural, as Algorithm 3 shows. The main difference from Elisabeth's evaluation is the whitening, which is already natively computed in $\mathbb{F}_2$. In this way, instead of dynamically updating the LUT, we perform an XOR with cleartext (`cXOR`), which is inexpensive and does not add any noise. This evaluation is significantly more expensive than Elisabeth's because it requires a larger LUT (18-bit instead of 16-bit LUT) and more rounds.

Algorithm 3: Evaluation of Margrethe

Input : Encrypted secret key $k_i \in \mathrm{RGSW}_s(sk_i)$, for $i \in [\![0, 2048)$, and $sk \in \mathbb{F}_2^{2048}$
Input : A large LUT $L \in \mathbb{Z}_{16}^{2^{18}}$
Output : Encrypted key stream

1 **for** $j \leftarrow 0$ **to** $14f - 1$ **do**
2 $\quad r \leftarrow$ `random_int`$(j, 2048)$
3 $\quad w \leftarrow$ `random_int`$(0, 1)$
4 $\quad$ `swap`(k_j, k_r) ; `// Sampling`
5 $\quad k_j \leftarrow$ `cXOR`(k_j, w); `// Whitening`
6 $acc \leftarrow 0$
7 **for** $b \leftarrow 0$ **to** 13 **do**
8 $\quad x \leftarrow s[b22 : (b+1)22]$
9 $\quad y \leftarrow$ `VP`$(x[0 : 18], L)$
10 $\quad r \leftarrow$ `RSGWtoLWE`$(x[18 : 22])$
11 $\quad acc \leftarrow acc + y + r$
12 $stream_i \leftarrow acc$
13 **return** $stream_i$

1 **Proc.** `cXOR`(C, w)
2 $\quad$ **if** $w = 1$ **then**
3 $\quad\quad$ **return** $\mathrm{RGSW}(1) - C$
4 $\quad$ **return** C

3.1.3 Summary

Table 1 summarizes the main parameters of the homomorphic evaluation of MFP ciphers following our leveled approach. We note that FiLIP, which was exceptionally designed to be evaluated as a binary logic circuit, could only be represented with unrealistic large LUTs in our approach. Conversely, all other MFP ciphers can be evaluated with LUTs of size 2^{16} to 2^{24}, which can all be efficiently evaluated using VPs. Execution time is expected to grow linearly with the number of LUTs and their sizes, which is exponential in the precision of the function they represent (*i.e.*, the number of input bits). Considering that, it is clear that, performance-wise, Elisabeth and Margrethe, in this order, are the two best-performing designs for our solution. However, the design of Elisabeth showed in Algorithm 1 has known vulnerabilities [GHBJR23], and the patch Elisabeth-b4 [HMS23], would require a 24-bit LUTs in our approach. In this way, Margrethe is the most promising solution, and we focus on it for the rest of this work.

Table 1: Summary of MFP cipher evaluation in our approach

Cipher	Key	Whitening	LUTs	
			Number	Size ($\log_2$)
FiLIP	$\mathbb{F}_2$	Native	1	512
Elisabeth	$\mathbb{Z}_{16}$	DynamicLUT	12	16
Elisabeth-b	$\mathbb{Z}_{16}$	DynamicLUT	14	24
Gabriel[a]	$\mathbb{Z}_{16}$	DynamicLUT	8	16
			10	24
Margrethe	$\mathbb{F}_2$	Native	14	18

[a] Gabriel requires **both** 8 16-bit and 10 24-bit LUTs.

We focus on TFHE-based evaluation as it typically presents optimal results for optimizing latency. However, we note that one could also consider a similar evaluation method using other HE schemes that may present different advantages. For example, it has been recently shown that schemes such as CKKS can provide efficient batched evaluation methods for binary logic gates and lookup tables [BCKS24, BKSS25]. This could, in principle, enable one to achieve high-throughput evaluations of MFP ciphers at the cost of increased latency. Assessing whether such solutions would be practical and competitive against other throughput-oriented ciphers (*e.g.* HERA [CHK+21]) is left as future work.

3.2 Multi-party evaluation

From the performance point of view, the only particularities of transchiphering in a multi-party scenario are the procedures for mixing and setting up keys (in the case of ciphers that require a key setup phase). In ciphers such as Elisabeth with keys in $\mathbb{Z}_{16}$, mixing can be done as simple addition. For Margrethe, however, addition needs to be done in $\mathbb{F}_2$ while RGSW samples (although encrypting $\mathbb{F}_2$) do arithmetic in a larger $\mathbb{Z}_p$ plaintext space. In this context, we need to perform the homomorphic evaluation of an `XOR` (Addition in $\mathbb{F}_2$) using RGSW samples. While there are many approaches for implementing it, we note that it is possible to perform XORs between RGSW samples with different parameter sets. This not only gives more flexibility to our construction, but also enables us to better exploit RGSW multiplication noise behavior. Recall that the external product between RGSW and RLWE samples (denoted by $\boxdot$) does not require them to have the same plaintext space, Algorithm 4 presents our solution.

Algorithm 4: RGSW XOR with different parameters

Input : An RGSW sample $A \in \mathrm{RGSW}_{k_A}(m_A)$ with parameters $k_A = (s, \ell_A, \beta_A)$, for $m_A \in \mathbb{F}_2$

Input : An RGSW sample $B \in \mathrm{RGSW}_{k_B}(m_B)$ with parameters $k_B = (s, \ell_B, \beta_B)$, for $m_B \in \mathbb{F}_2$

Output : An RGSW sample $C \in \mathrm{RGSW}_{s,\ell_B,\beta_B}(m_a \oplus m_b)$

$T \leftarrow \mathrm{RGSW}_{k_B}(1) - 2B$
$R \leftarrow \mathrm{RGSW}_{k_B}(0)$
Let r_i and t_i be the i-th RLWE samples of RGSW samples R and T, respectively.
for $i \leftarrow 0$ **to** $2\ell_B - 1$ **do**
  $r_i \leftarrow A \boxdot t_i$
return $R + B$

3.2.1 Compressed keys

The main disadvantage of leveled evaluation in GSW-like schemes (*e.g.*, TFHE and FHEW) is that RGSW samples are significantly larger than LWE samples. Methods for compressing RGSW keys, however, are broadly available in the FHE literature and can be exploited to significantly reduce the size of the keys. In particular, one can use the GLWEtoGGSW adopted in [CCH+24] or new methods for circuit bootstrapping, such as [WWL+24, WLW+24], which are efficient and enable the use of keys at the order of Kilobytes to just a few Megabytes (compared to hundreds of Megabytes that would be required by RSW ciphertexts). We further discuss concrete numbers for key size in Section 5.

We generally consider key compression techniques to be orthogonal to our proposals, as there are many different approaches for it, and most of them are standalone procedures that run independently of the application. Concretely, any of the methods used by [CCH+24], [WWL+24], and [WLW+24] can be used to compress our keys without any modifications to our algorithms or to their methods. When considering the context of a multi-party protocol, however, it is particularly interesting to note that, if the compression methods are based on programmable bootstrappings (which is the case of [CCH+24, WWL+24, WLW+24]), one can perform the key mixing within the decompression algorithm at no additional cost, by using the PBS to evaluate the XOR between keys.

4 Security analysis of Margrethe in the two-RKS setting

As discussed in Section 1, providing a secure solution for transciphering in MPC-FHE protocols that generate up to two Related Key Streams (RKS) can be reduced to the Related-Key Attack (RKA) security of the symmetric encryption scheme. In this section, we provide a method to study the RKA security of MFP stream ciphers using the Boolean properties of the block filter function, its subfunctions and its first-order derivatives. We then use this method to demonstrate that the Margrethe stream cipher is secure against RKA in this context, thereby ensuring the security of transciphering without requiring any specific countermeasures at the MPC protocol level.

In the context of RKA, we will consider that an adversary can get access to the keystream from different (related) keys and aims at recovering the key. More precisely, we assume the adversaries can get access to keystream elements $\mathsf{Sym}(K, z)$ and $\mathsf{Sym}(\Phi_j(K), z)$, for $1 \leq j \leq J$. We consider keystreams under the same IV (z) as a simplification for the analysis, noticing that this only gives an advantage to the adversary. Conversely, we note that simply ensuring the use of different IVs is not a general countermeasure against RKAs.

We focus on the case of a single relation (that is $J = 1$, which is our two-RKS setting) and $\Phi_1(K) = K \oplus \Delta$ where $\oplus$ represents the bitwise XOR and Δ is chosen by the adversary.

Since Margrethe utilizes operations over $\mathbb{F}_2$ and $\mathbb{Z}_{16}$, the equations derived from the related keystreams can be exploited by an adversary in both rings. First, we examine the equations stemming from the interpretation of the keystream over $\mathbb{Z}_{16}$ in Section 4.1, and provide an analysis similar to [HMS23] for this attack path. Next, we analyze the equations derived from the keystream over $\mathbb{F}_2^4$ in Section 4.2. We show that the RKA security of Margrethe with filter function f can be reduced to the regular analysis of Margrethe with filter function f', where f' is a combination of symmetric functions and derivatives of f. We then evaluate the concrete security of Margrethe in the two-RKS setting in Section 4.3. Then, in Section 4.4, we explore the connection between RKA security and a model of differential fault attacks previously considered in the context of stream ciphers. Finally, we demonstrate in Section 4.5 how having more than two related keystreams rapidly degrades the security of Margrethe in the multi-RKS setting.

4.1 Analysis over $\mathbb{Z}_{16}$

For two related keys K and $K \oplus \Delta$, once the message has been subtracted, following the scheme description of Section 2.4.1 the adversary has access to $f(S_i(P_i(K)) \oplus w_i)$ and $f(S_i(P_i(K \oplus \Delta)) \oplus w_i)$ for a keystream element (indexed by i). Since these values are in $\mathbb{Z}_{16}$, the adversary can consider any linear combination over $\mathbb{Z}_{16}$, giving the following equation:

$$a \cdot f(S_i(P_i(K)) \oplus w_i) + b \cdot f(S_i(P_i(K \oplus \Delta) \oplus w_i), \tag{1}$$

where $a, b \in \mathbb{Z}_{16}$, "$\cdot$" and "$+$" denote the multiplication and addition over $\mathbb{Z}_{16}$.

In the next proposition, we show that Equation 1 can be used to generate the keystream of a variant of Margrethe with a different filter function.

Proposition 1. *Let $N, n, K, f, m, P_i, S_i, \omega_i$ for $i \in [1, m]$ be defined for an instance of Margrethe cipher (as in Section 2.4.1). Let $\Delta \in \mathbb{F}_2^N$ and $a, b \in \mathbb{Z}_{16}$, from the keystreams of Margrethe with key K and with key $K \oplus \Delta$, the keystream of a variant of Margrethe with known filtering function f', key K, and same PRNG stream, can be obtained.*

Proof. Since the permutation P_i and the subset selection S_i are linear operations over $\mathbb{F}_2^N$ we can rewrite Equation 1 obtained from the two keystreams as:

$$a \cdot f(K^i \oplus w_i) + b \cdot f(K^i \oplus \Delta^i \oplus w_i),$$

where we denote by Δ^i and K^i the permuted selected part of Δ and K respectively, for each keystream element, that is $K^i = P_i(S_i(K))$. We note that the transformation from K to K^i is linear, and public.

Utilizing the fact that the function f is a direct sum over $\mathbb{Z}_{16}$ of functions with a smaller number of variables, we can rewrite the equation as:

$$\sum_{j=0}^{13} \left(a \cdot g((K^i \oplus w_i)_{[22j+1:22j+18]}) + b \cdot g((K^i \oplus \Delta^i \oplus w_i)_{[22j+1:22j+18]})\right)$$
$$+ \sum_{j=0}^{13} \left(a \cdot (\sum_{k=0}^{3} 2^k (K^i \oplus w_i)_{[22j+19+k]}) + b \cdot (\sum_{k=0}^{3} 2^k (K^i \oplus \Delta^i \oplus w_i)_{[22j+19+k]}) \right),$$

where we denote by $[j]$ the j-th bit of the binary vector and by $[j_1 : j_2]$ the binary vector from the position j_1 to the position j_2 (included).

The last equation we obtained can be considered as the keystream obtained from a variation of Margrethe with the same parameters N and n, with only the filtering function

changed, f' where:

$$\begin{aligned} f'(x_1,\ldots,x_{308}) = a\sum_{j=0}^{13} g(x_{22j+1},\ldots,x_{22j+18}) \\ + b\sum_{j=0}^{13} g(x_{22j+1}\oplus\Delta^i_{22j+1},\ldots,x_{22i+18}\oplus\Delta^i_{22j+18}) \\ + a\sum_{k=0}^{3} 2^k x_{22j+19+k} + b\sum_{k=0}^{3} 2^k(x_{22j+19+k}\oplus\Delta^i_{22j+19+k}). \end{aligned}$$

□

Following Proposition 1, the key recovery attacks considered in the security analysis of Margrethe in the single key setting can also be considered in the present related key setting. Over $\mathbb{Z}_{16}$, the security analysis in the single key setting of [HMS23] emphasizes the fact that g does not admit a polynomial representation, making it difficult to consider algebraic attacks. It is also the take-out for the analysis over $\mathbb{Z}_{16}$ for Elisabeth [CHMS22] and in general for symmetric ciphers over $\mathbb{Z}_q$ in [GAH+23].

Moreover, since the filtering function acts at the bit level, grouping the inputs of f' four by four to study the corresponding function over $\mathbb{Z}_{16}$ has little interest for an adversary. First, the selections and permutations at the bit level force us to consider $4!\binom{N}{4}$ (key) elements of $\mathbb{Z}_{16}$. Then, such embedding of f' gives a function from $\mathbb{Z}_{16}^{77}$ to $\mathbb{Z}_{16}$, or from $\mathbb{Z}_{16}^{63}$ to $\mathbb{Z}_{16}$ if we consider only the parts from g. A random function from $\mathbb{Z}_{16}^4$ to $\mathbb{Z}_{16}$ is already not a polyfunction nor agreeing with a polyfunction on half of the inputs with probability lower than 2^{128} ([CHMS22], Proposition 3). Accordingly, we focus on the analysis over $\mathbb{F}_2$ for the concrete analysis of Margrethe in the related key attack model.

4.2 Analysis over $\mathbb{F}_2$

In the following we consider the equations coming from the interpretation of the keystream over $\mathbb{F}_2$, each keystream element giving four bits. We denote by F the vectorial function $F(x_1,\ldots,x_{308}) \mapsto (z_1,z_2,z_3,z_4)$ defined by applying f and interpreting the output y in $\mathbb{Z}_{16}$ as $z \in \mathbb{F}_2^4$ by identifying $y = z_1 + 2z_2 + 4z_3 + 8z_4$. In this case, similarly to Equation 1, considering two binary vectors a and b an adversary can derive the following linear combinations:

$$a \cdot F(S_i(P_i(K)) + w_i) + b \cdot F(S_i(P_i(K+\Delta)) + w_i), \tag{2}$$

where $\cdot$ denotes the usual inner product over $\mathbb{F}_2^4$ and $+$ denotes the (vector)-addition over $\mathbb{F}_2$. Since the permutation P_i and the subset selection S_i are linear operations over $\mathbb{F}_2^N$ we can rewrite Equation 2 as:

$$\begin{aligned} &a \cdot F(K^i + w_i) + b \cdot F(K^i + \Delta^i + w_i) \\ &= a \cdot F(K^i + w_i) + a \cdot F(K^i + \Delta^i + w_i) + (a+b) \cdot F(K^i + \Delta + w_i) \\ &= a \cdot \mathsf{D}_{\Delta^i} F(K^i + w_i) + (a+b) \cdot F(K^i + \Delta^i + w_i), \end{aligned}$$

where $\mathsf{D}_{\Delta^i} F$ denotes the first order derivative of F in the direction Δ^i.

From these potential equations, the simplest one to use for an adversary is the one such that $a + b = 0$, allowing them to obtain equations depending only on a first order derivative of F (hence of lower degree than F, see Property 3). We delve into the structure of F to see how its derivatives can be used. The expression of F is the following:

$$F(x_1,\ldots,x_{308}) = \boxplus_{i=0}^{13} G(x_{22i+1},\ldots,x_{22i+18}) \boxplus (x_{22i+19}, x_{22i+20}, x_{22i+21}, x_{22i+22}),$$

where G is the vectorial Boolean functions obtained by considering the output of g as an element of $\mathbb{F}_2^4$ (using the same identification as for F) and $\boxplus$ denotes the interpretation of the addition over $\mathbb{Z}_{16}$ as a vectorial Boolean function.

The addition over $\mathbb{Z}_{16}$ is linear for the least significant bit over $\mathbb{F}_2$ (the parity is preserved), but not for the other bits. We denote by H^t the vectorial Boolean function from $\mathbb{F}_2^{4t}$ to $\mathbb{F}_2^4$ giving the binary representation of the addition in $\mathbb{Z}_{16}$. We study it in the following proposition.

Proposition 2. *Let $t \in \mathbb{N}$, $t > 0$, let H^t be the vectorial Boolean function from $\mathbb{F}_2^{4t}$ to $\mathbb{F}_2^4$ giving the binary representation of the addition in $\mathbb{Z}_{16}$. Its coordinate functions h_0, h_1, h_2 and h_3 are given by:*

- $h_0(x_1, \ldots, x_{4t}) = \sigma_{1,t}(x_1, x_5, \ldots, x_{4t-3})$
- $h_1(x_1, \ldots, x_{4t}) = \sigma_{1,t}(x_2, x_6, \ldots, x_{4t-2}) + \sigma_{2,t}(x_1, x_5, \ldots, x_{4t-3})$
- $h_2(x_1, \ldots, x_{4t})$

$$\begin{aligned} &= \sigma_{1,t}(x_3, x_7, \ldots, x_{4t-1}) + \sigma_{2,t}(x_2, x_6, \ldots, x_{4t-2}) + \sigma_{4,t}(x_1, x_5, \ldots, x_{4t-3}) \\ &+ \sigma_{2,2}[\sigma_{1,t}(x_2, x_6, \ldots, x_{4t-2}), \sigma_{2,t}(x_1, x_5, \ldots, x_{4t-3})] \end{aligned}$$

- $h_3(x_1, \ldots, x_{4t})$

$$\begin{aligned} &= \sigma_{1,t}(x_4, x_8, \ldots, x_{4t}) + \sigma_{2,t}(x_3, x_7, \ldots, x_{4t-3}) + \sigma_{4,t}(x_2, x_6, \ldots, x_{4t-2}) \\ &+ \sigma_{8,t}(x_1, x_5, \ldots, x_{4t-3}) + \sigma_{2,4}[\sigma_{1,t}(x_3, x_7, \ldots, x_{4t-1}), \sigma_{2,t}(x_2, x_6, \ldots, x_{4t-2}), \\ &\sigma_{4,t}(x_1, x_5, \ldots, x_{4t-3}, \sigma_{2,2}[\sigma_{1,t}(x_2, x_6, \ldots, x_{4t-2}), \sigma_{2,t}(x_1, x_5, \ldots, x_{4t-3})])], \end{aligned}$$

where $\sigma_{i,t}$ is the t-variable elementary symmetric function of degree i.

Proof. We do the proof in two parts: first, we rewrite the addition in $\mathbb{Z}_{16}$ based on the binary representation of the inputs, and then we show how symmetric functions give the binary representation of the intermediate steps of this computation.

First, we rewrite the sum modulo 16 using the binary representation

$$\begin{aligned} S &\equiv \sum_{i=0}^{t-1} (x_{4i+1} + 2x_{4i+2} + 4x_{4i+3} + 8x_{4i+4}) \\ &\equiv \sum_{i=0}^{t-1} x_{4i+1} + 2\sum_{i=0}^{t-1} x_{4i+2} + 4\sum_{i=0}^{t-1} x_{4i+3} + 8\sum_{i=0}^{t-1} x_{4i+4} \\ &\equiv (\sum_{i=0}^{t-1} x_{4i+1} \mod 16) + 2(\sum_{i=0}^{t-1} x_{4i+2} \mod 8) \\ &\quad + 4(\sum_{i=0}^{t-1} x_{4i+3} \mod 4) + 8(\sum_{i=0}^{t-1} x_{4i+4} \mod 2) \end{aligned}$$

We recall the four obtained terms as

- $a = (\sum_{i=0}^{t-1} x_{4i+1} \mod 16)$
- $b = (\sum_{i=0}^{t-1} x_{4i+2} \mod 8)$
- $c = (\sum_{i=0}^{t-1} x_{4i+3} \mod 4)$
- $d = (\sum_{i=0}^{t-1} x_{4i+4} \mod 2)$

and denote their binary representations as (a_0, a_1, a_2, a_3), (b_0, b_1, b_2), (c_0, c_1) and (d_0). The value of the binary coefficients can be expressed in terms of congruence, taking a as an example we have that:

- $a_0 = a \mod 2$, that is $a_0 = 1 \Leftrightarrow (a \mod 2) \in \{1\}$
- $a_1 = 1 \Leftrightarrow (a \mod 4) \in \{2, 3\}$
- $a_2 = 1 \Leftrightarrow (a \mod 8) \in \{4, 5, 6, 7\}$
- $a_3 = 1 \Leftrightarrow (a \mod 16) \in \{8, 9, 10, 11, 12, 13, 14, 15\}$

We will link these expressions to the results of symmetric functions later on.

S can then be rewritten as:

$$\begin{aligned} S &\equiv a_0 + 2(a_1 + b_0) + 4(a_2 + b_1 + c_0) + 8(a_3 + b_2 + c_1 + d_0) \\ &\equiv z_0 + 2z_1 + 4z_2 + 8z_3, \end{aligned}$$

where $(z_0, z_1, z_2, z_3) \in \mathbb{F}_2^4$.

We have that $z_0 = a_0$ since all the other terms are multiples of 2. z_1 is obtained from the parity of $a_1 + b_0$, that is $z_1 = 1$ if and only if $a_1 + b_0 = 1 \mod 2$. The sum (modulo 16) of $a_1 + b_0$ is between 0 and 2, in the case it equals 2, it will contribute to z_2. Accordingly, $z_2 = a_2 + b_1 + c_0 + e \mod 2$ where $e \in \mathbb{F}_2$ is such that $e = 1$ if and only if $a_1 + b_0 \mod 4 \in \{2, 3\}$. The sum $a_2 + b_1 + c_0 + e \mod 16$ can contribute to z_3, if it is congruent to 2 or 3 modulo 4. Hence, $z_3 = a_3 + b_2 + c_1 + d_0 + f \mod 2$ where $f \in \mathbb{F}_2$ is such that $f = 1$ if and only if $a_2 + b_1 + c_0 + e \mod 4 \in \{2, 3\}$.

Then, we show how the elementary symmetric functions allow us to compute the intermediate values such as e and f from their inputs, through the following lemma. Let $t, n \in \mathbb{N}$ and $2^t \leq n$, the following holds on $\sigma_{2^t,n}$:

$$\sigma_{2^t,n} = \begin{cases} 0 & \text{if } \mathsf{w_H}(x) \mod 2^{t+1} \in [0, 2^t - 1], \\ 1 & \text{if } \mathsf{w_H}(x) \mod 2^{t+1} \in [2^t, 2^{t+1} - 1]. \end{cases}$$

Since $\sigma_{2^t,n}$ is symmetric its value is only determined by the Hamming weight of x. Using Property 1, the value taken by $\sigma_{2^t,n}$ on an element of Hamming weight k is the parity of the binomial coefficient $\binom{k}{2^t}$. From Lucas' theorem (Property 2) we can determine this parity from the binary expansion of 2^t and k. We recall the value of the following binomial coefficients: $\binom{0}{0} = \binom{1}{0} = \binom{1}{1} = 1$ and $\binom{0}{1} = 0$. Since 2^t is a power of 2 it has only one 1 in its binary expansion (in position t), hence the parity of $\binom{k}{2^t}$ is fully determined by the value of the coefficient in position t in the binary expansion of k that we denote by k_t. $\binom{k}{2^t} \equiv 0$ if $k_t = 0$ and $\binom{k}{2^t} \equiv 1$ if $k_t = 1$. Equivalently, $\sigma_{2^t,n}$ takes the value 0 if $\mathsf{w_H}(x) \in [0, 2^t - 1] \mod 2^{t+1}$ and the value 1 if $\mathsf{w_H}(x) \in [2^t, 2^{t+1} - 1] \mod 2^{t+1}$.

Using the expression in terms of σ_1, σ_2, σ_4, and σ_8 for each computation using congruence, we obtain the final expression of the coordinate functions h_i.

□

Using Proposition 2 we can rewrite F as: $F(x_1, \ldots, x_{308}) =$

$$H^{28}(G(x_1, \ldots, x_{18}), x_{19}, x_{20}, x_{21}, x_{22}, \ldots, G(x_{287}, \ldots, x_{304}), x_{305}, x_{306}, x_{307}, x_{308}).$$

Accordingly, we can get $a \cdot F(K^i + w_i)$ derived from Equation 2 by using the derivatives of the coordinate functions of H: h_0, h_1, h_2 and h_3. We denote by f_0, f_1, f_2 and f_3 the coordinate functions of F and by g_0, g_1, g_2 and g_3 the coordinate functions of G. From Proposition 2, we have that f_0 is simply the direct sum of copies of g_0 on different inputs and some key elements. In contrast, the three other coordinate Boolean functions have σ_2, σ_4, or σ_8 in their expression, giving combinations of the g_i of higher degree and more monomials. Hence, we focus our analysis on the equations obtained in the least significant bit, given in the following proposition.

Proposition 3. *Let f_0 and g_0 the Boolean functions in* 308 *and* 18 *variables respectively giving the least significant bit of f and g of Margrethe (as defined in Section 2.4.1), the following holds on the first derivative of f_0:*

$$\forall \Delta \in \mathbb{F}_2^{308}, \quad \mathsf{D}_\Delta f_0(x) = \varepsilon + \sum_{i=0}^{13} \mathsf{D}_{\Delta[1+22i,18+22i]} g_0(x_{1+22i}, \ldots, x_{18+22i}),$$

where $\varepsilon \in \{0,1\}$.

Proof. We rewrite $\mathsf{D}_\Delta f_0(x)$ using its expression as a direct sum and the properties of derivatives (Property 3):

$$\begin{aligned}
&\mathsf{D}_\Delta f_0(x) \\
&= \mathsf{D}_\Delta h_0^{28}(G(x_1, \ldots, x_{18}), x_{19}, x_{20}, x_{21}, x_{22}, \ldots, G(x_{287}, \ldots, x_{304}), x_{305}, x_{306}, x_{307}, x_{308}) \\
&= \mathsf{D}_\Delta \sum_{i=0}^{13} g_0(x_{1+22i}, \ldots, x_{18+22i}) + x_{19+22i} \\
&= \sum_{i=0}^{13} \mathsf{D}_{\Delta[1+22i:18+22i]} g_0(x_{1+22i}, \ldots, +x_{18+22i}) + \mathsf{D}_{\Delta[19+22i]} x_{19+22i} \\
&= \varepsilon + \sum_{i=0}^{13} \mathsf{D}_{\Delta[1+22i,18+22i]} g_0(x_{1+22i}, \ldots, x_{18+22i}),
\end{aligned}$$

where $\varepsilon \in \{0,1\}$. The second equation comes from Proposition 2 (σ_1 is the sum of its inputs). The third and fourth equations come from Property 3, the derivative of a direct sum is the direct sum of their derivatives, and the derivative of a linear function is a constant function. □

Accordingly, with the interpretation over $\mathbb{F}_2$ the attack of the adversary in the related key models boils down to a key recovery attack on a variant of Margrethe where the filter function is obtained from the combination of symmetric functions and derivatives of Margrethe's filter function. We study the complexity of these related key attacks with the security analysis of Margrethe over $\mathbb{F}_2$, based on the cryptographic criteria of the functions used as filters. Since $\mathsf{D}_\Delta f_0(x)$ is a direct sum, we can bound the parameters of the entire function from the parameters of each subpart $\Delta' g_0(y_1, \ldots, y_{18})$. For the concrete analysis (Section 4.3), we determine a lower bound on the complexity of the attacks based on the worst parameters of the derivatives of the 15 components Boolean functions of G. Since the expression of $\mathsf{D}_\Delta f_0(x)$ is the simplest, we assume the complexities we obtain are lower bounds of the ones of strategies considering other component functions since σ_1 appears in each component function (See Proposition 2) and higher degree combination appear for the other components.

4.3 Concrete analysis over $\mathbb{F}_2$

In [HMS23], Section 6.3, Margrethe's security analysis was conducted by computing the values of the cryptographic parameters of the 18-to-4 function corresponding to its LUT, and using the properties of direct sums to derive bounds on the parameter of the filter function. More precisely, Hoffmann *et al.* focused on:

- The degree of the 15 non null component functions fixing up to 2 bits.
- The nonlinearity of the 15 non null component functions fixing up to 2 bits.
- The number of monomials of the 15 non-null component functions fixing up to 2 bits.

The degree allows us to derive a bound on the algebraic immunity using Lemmas 2 and 3. This formula is only tight for the lowest bit here and is used as a lower bound for the other component functions (as explained in Section 4.2). Those characteristics were also studied for all the possible subfunctions obtained by fixing up to two of the 18 inputs.

The algebraic immunity is then used to get a security bound against algebraic attacks and the nonlinearity is used to provide a security bound on correlation-like attacks. Computing the parameters of the subfunctions allows to bound the complexity of variations of the previous attacks using guess and determine strategies. We also verified for a high number of derivatives (checking every possible derivative was out of our computational power) that the number of monomials was close to 2^{17}, as would be the one of a random function. This number of monomials ensures security against linearization attacks on sparse systems such as in [GHBJR23].

Since the concrete security analysis in this work follows the security analysis framework of [HMS23], we do not rewrite the details of the attacks applying and methods to derive the properties. We recall that the attacks on Margrethe and the mixed filter permutator paradigm are based on those considered since the introduction of the filter permutator paradigm and the early version of FLIP cipher. The evolution of these designs and their cryptanalysis, as evidenced by the works [MJSC16, DLR16, CMR17, CT18, MCJS19, CHMS22, GHBJR23, HMS23, GGM24, MW24], highlights the well-founded of the security analysis approach we consider for this type of stream cipher. Yet, we remind the main formulas used to derive the security properties from the characteristics of the Boolean functions, and refer to [HMS23] Section 2.3 and 3 or the eprint version of [MCJS19] for more detailed explanations:

- The Algebraic Attack (AA) [CM03] complexity is $\mathcal{O}(D)^\omega$ where $D = \sum_{i=1}^{\mathsf{AI}(f)} \binom{N}{i}$.
- The Fast Algebraic Attack complexity (FAA) [Cou03] is $\mathcal{O}(D\log^2(D) + N \cdot D\log(D))$ where $D = \sum_{i=1}^{\mathsf{AI}(f)+1} \binom{N}{i}$.
- The correlation-like attack complexity is $\mathcal{O}\left(((2^n)/\mathsf{NL}) \cdot N^\omega\right)$.
- Those attacks can be used on subfunctions resulting from guess and determine strategies e.g. in [DLR16]. In this case, the complexity of the attack comes with an overhead of $2^l\binom{N}{l}$, ℓ being the number of guessed bits.

Considering worst case analysis, to determine the worse value of the cryptographic parameter that can be reached by a derivative of a subfunction of f, we conduct the same analysis for the $2^{18} - 1$ non trivial derivatives of first order. In Table 2, we give the lowest degree and nonlinearity obtained among all the derivatives of the component functions of G. Since building a distinguer does not seem relevant in the case of multiple keystreams, we do not discuss the resiliency order of the functions in the following tables.

Table 2: Minimum cryptographic parameters over the $2^{18} - 1$ first order derivatives of the function G seen as vectorial Boolean functions after fixing up to ℓ binary inputs

ℓ	0	1	2
deg	≥ 13	≥ 13	13
NL	64909	32237	16004

We use Lemma 1 to bound the parameters of the derivatives of (the components of) F from the ones of G. The results on the nonlinearity and correlation-like attack are

displayed in Table 3. For the algebraic immunity, we apply Lemma 3: each one of the term of the sum being at least 13 we can build the chain to ensure that, with no guesses, the lowest AI among all the derivative of F is at least 13, and the same bound applies when each of the 14 parts received at most two guesses [3]. Similarly to the analysis of Margrethe in the standard setting in [HMS23] Section 6.3, for computational reason, we are not able to extend the results of Table 4.3 to more than two guesses. This makes us unable to prove a lower bound on the AI when 3 guesses or more is made on the variables of the same 17-bit function (among the 14). Since this function could be constant (whereas highly unlikely), we cannot rely on those functions in the chain of Lemma 3 since it only gives $\mathsf{AI} \geq 13 - \lfloor \ell/3 \rfloor$. Observing that in the standard setting the same bound in [HMS23] is $\mathsf{AI} \geq 14 - \lfloor \ell/3 \rfloor$, we will rely on the same assumptions. The authors of [HMS23] assume the AI of the components of G is at least 6, which is a low bound for a 18-variable function. If we assume the same bound for the first order derivatives (in 17 variables), if one function did not receive guesses and 7 received less than 3 guesses, applying Lemma 3 already guarantees an AI of at least 13. We expect the bound $\mathsf{AI} = 6$ to be highly conservative since the AI of a random function is close to $n/2$, for further details, we refer to [HMS23]. Following the formulas of AA and FAA, with this assumption, the bound on the algebraic attack and fast algebraic complexities are greater than 2^{128}. We also checked on a high number of derivatives that the minimal number of monomials that appears in any of the component of the derivative is extremely close to 2^{17}, which is the one expected from a random function and sufficiently high to ensure that the attack of [GHBJR23] is inapplicable.

Table 3: Minimal parameter bounds over the first order derivatives of the component functions of F, up to ℓ fixed binary inputs, and corresponding complexity estimations (in bits).

Number of guesses ℓ	0	$[1, 13]$	$[14, 27]$
$\mathsf{NL}/2^{n-\ell}$	$\geq 0.5 - 10^{-4}$	$\geq 0.5 - 10^{-4}$	$\geq 0.5 - 10^{-4}$
Correlation attack complexity (bits)	$>> 128$	$>> 128$	$>> 128$

We provide here a repository on which all the scripts used to compute the Boolean function properties and the complexity of the attack. To efficiently compute the degree, we use the lazy evaluation idea from [HMS23]. We highlight that, to compute the nonlinearity of these functions, we came up with an optimized implementation of the Walsh-Hadamard transform using AVX-512 instructions, which may be of interest to other applications (such as leakage profiling [CDSU23, BED+22]).

4.4 Connection between differential fault attacks and RKA

Related Key Attacks have been so far an unstudied topic in the transciphering literature. Differential Fault Attacks (DFA), on the other hand, have already been demonstrated against several commonly used ciphers, and, in some contexts, they exploit similar vulnerabilities as RKAs.

DFAs are a type of active side-channel attack in which a fault is injected into a device. The principle involves comparing the results of the same algorithm executed on both a normal device and a faulted device (which may also be the same device at different times). The attack is effective when the differences, or the lack thereof, between the two executions provide additional information that allows the recovery of secret information (typically the key for a symmetric cipher). DFA on stream ciphers have been

[3] in detail, the chain technique of Lemma 3 gives a lower bound of 13 while 13 parts received up to two guesses

extensively studied, particularly in [HS04], and on specific ciphers such as those detailed in [HR08,BMS12,BM13,MSS17,SSMC17]. In the following, we focus on DFA where the fault is injected only once into the key register, as demonstrated in [RBM20, RKMR23, MR24].

In the model considered, the adversary has access to elements $\mathsf{Sym}(K, z)$ and $\mathsf{Sym}(K', z)$ which allow them to derive the keystream for the same initialization vector using the original key K and the faulted key K'. The fault model assumed is a bit-flip at unknown positions in K, represented as $K' = K \oplus \Lambda$ where $\Lambda \in \mathbb{F}_2^N$ and N denotes the bit-size of the key. The attacks presented in [RKMR23, MR24] focus on the specific case where Λ has a Hamming weight of 1.

We observe that the same pairs are obtained in this DFA setting as in the related key setting defined in Section 4 when there is only one related key of the form $K \oplus \Delta$. Accordingly, the same analytic attacks can be performed, assuming the vectors Δ and Λ follow the same restrictions. For example, the analysis in Section 4.3 demonstrates that Margrethe remains secure in the related key setting for any Δ, which ensures resistance to DFA with any Λ. Conversely, the DFA attacks on FLIP and FiLIP presented in [MR24] using Λ with Hamming weight 1 do not imply related key attacks for arbitrary Δ. However, these schemes are insecure for any Δ with Hamming weight 1. In conclusion, the resistance of Margrethe to related key attacks implies resistance to this model of DFA. Conversely, no cipher that is vulnerable to DFA can be secure in the related key setting.

4.5 Generalization to more than two keystreams

As defined in Section 2.2, static MPC protocols can generally work by using only up to two related key streams, a setting for which we demonstrated Margrethe's security. Generalizing it to dynamic settings, where *many-RKS* can be produced, would be natural, but, as we show in this section, having more than two related keystreams quickly degrades the security of Margrethe in the related key model. We assume the adversary has access to the keystreams $\mathsf{Sym}(K, z)$, $\mathsf{Sym}(\Phi_j(K), z)$ for $1 \leq j \leq J$ where $\Phi_j(K) = K + \Delta_j$, using the same approach as in Section 4.2, they can derive equations similarly to Equation 2 in the context of two related keys:

$$a \cdot F(S_i(P_i(K)) + w_i) + \sum_{j=1}^{J} b^{(j)} \cdot F(S_i(P_i(K + \Delta_j) + w_i), \tag{3}$$

where the $b^{(j)}$ are elements of $\mathbb{F}_2^4$.

Fixing a to give only the LSB, and $b^{(j)} = a$ for all j, we can use again the linearity over $\mathbb{F}_2$ of the LSB of the sum modulo 16 (see the expression of h_0 in Proposition 2), allowing us to derive the following from Equation 3:

$$f_0(K^i + w_i) + \sum_{j=1}^{J} f_0(K^i + \Delta_j^i + w_i)$$

In particular, if the Δ_j are the elements of a vector space (without the all-0 vector) of dimension t, that is $\{\Delta_j \,|\, j \in [1, J]\} = \{v \cdot (\Delta_{j^1}, \ldots, \Delta_{j^t}), \,|\, v \in \mathbb{F}_2^t \setminus \{0_t\}\}$, it allows the adversary to obtain:

$$\sum_{v \in \mathbb{F}_2^t}^{J} f_0(K^i + w_i + v \cdot (\Delta_{j^1}^i, \ldots, \Delta_{j^t}^i)) = \mathsf{D}_{\Delta_{j^1}^i, \ldots, \Delta_{j^t}^i} f_0(K^i + w_i). \tag{4}$$

Accordingly, the adversary obtains equations from an order-t derivative of f_0, and using Property 3 it gives a function of degree no more than $\mathsf{deg}(f_0) - t$ in the key bits. For Margrethe's filtering function, since f is the direct sum of 18-variable functions, it gives

an algebraic system of degree at most $18 - t$. This system becomes easier to solve as the number of related keystreams becomes higher, up to becoming trivial with access to 2^{18} related keys.

We observed that a key derivation technique could be applied to Margrethe to prevent the related-key relationship from holding for most keystream elements. We explored key derivation methods based on the concepts of Sidon sets and almost perfectly nonlinear functions. While these methods showed potential, they were not conclusive in significantly enhancing the design's security against related-key attacks. Therefore, we have deferred a detailed discussion of this approach to Section A in the supplementary material.

5 Practical Implementation

We select 6 different parameter sets for evaluating Margrethe, with 2 of them specifically for multi-party evaluation. Table 4 presents them. We use binary keys for compatibility with TFHE, but this is not a requirement for our implementation (performance would be similar with, *e.g.*, Gaussian-distributed keys). Input noise is chosen according to the parameters to achieve 128-bit security. Each result is the average of at least 100 executions and produced key streams are validated with a cleartext Python implementation of Margrethe.

Table 4: Parameters for homomorphically evaluating Margrethe. The output noise (std. dev.) of the cipher evaluation is measure experimentally for each parameter.

Parameter	N	q	β	ℓ	Key Size (MB)	Output noise (σ)
Set 1	2048	2^{52}	2^{23}	1	128	$2^{36.7}$
Set 2			2^{17}	2	256	$2^{30.1}$
Set 3			2^{13}	3	384	$2^{26.6}$
Set 4			2^{5}	9	1152	$2^{19.5}$
Set MP1			2^{12}	3	384	$2^{39.9}$
Set MP2			2^{10}	2	256	

5.1 Results

We implement Margrethe using the MOSFHET library [GBA24]. Our main comparison is with other TFHE-based transciphering implementations that also aim at minimizing single-message latency. We compare with the main ciphers following the MFP paradigm as well with the recent work of Cho *et al.* [CCH+24]. Table 5 shows the results. Multithreaded results consider the largest possible parallelism to minimize single-message latency but not throughput. *I.e.*, we consider internal parallelism in the cipher, but not parallel executions for generating multiple messages at once. We measure all our results in a `c6i.metal` instance (Intel Xeon 8375C at 3.5GHz) on AWS. Results for other works are the ones reported by their authors. While some impact due to differences in execution environments is expected, we note most other works run their experiments on similar machines. Our implementation is available at https://github.com/antoniocgj/MARGRETHE.

Remark 1. It is important to notice that our main goal is to minimize latency, and that there are various ciphers that would be faster in terms of throughput. HERA, for example, achieves a throughput of up to 5 kilobytes per second [CHK+21]. On the other hand, it presents 4.1 seconds of latency, which is more than a thousand times slower than our implementation. Ultimately, it boils down to the specific applications to define which aspect should be prioritized. [CHMS22] further discusses these aspects and compares with other ciphers.

Table 5: Margrethe evaluation and comparison with other ciphers. Time is provided in milliseconds and throughput is measured in the number of bits per second. Multi-threading is used only to improve latency (see Remark 2).

	Implement.	Single-thread		Multi-thread	
		Latency	Throughput	Latency	Throughput
FiLIP [MCJS19]	[CDPP22] I	2.62	381.68	-	-
FiLIP [MCJS19]	[MPP23] I	25.2	158.73	-	-
FiLIP [MCJS19]	[MPP23] II	71	56.50	-	-
Elisabeth [CHMS22]	[CCH+24]	2049	1.96	-	-
Elisabeth-b [HMS23]	[CCH+24]	5538	0.75	-	-
Gabriel [HMS23]	[CCH+24]	4662	0.86	-	-
Kreyvium [CCF+18]	[CCH+24]	-	-	134.00	7.47
Kreyvium [CCF+18]	[BOS23]	-	-	150.00	427.47[a]
Trivium [CP08]	[BOS23]	-	-	121.00	529.47[a]
FRAST [CCH+24]	[CCH+24]	6194.00	-	-	20.66[a]
Margrethe [HMS23]	**Set 1**	**27.2**	**147.06**	**3.12**	**1282.05**
Margrethe [HMS23]	**Set 2**	**54.2**	**73.8**	**4.21**	**950.11**

[a] These implementations employ trivial parallelization to improve throughput.

5.1.1 Probability of failure and plaintext independent transciphering

Let erf be the Gaussian error function, given a ciphertext modulus q and a noise standard deviation σ, a k-bit message can be successfully decrypted with probability $\mathsf{erf}\left(\frac{q/2^{k+1}}{\sigma\sqrt{2}}\right)$. As Table 4 shows, the ratio between ciphertext modulus and output noise in our evaluation approach (q/σ) varies between 2^{12} to 2^{32}, which would be enough to guarantee successful decryption with a low probability of failure ($< 2^{-64}$) for messages of up to 7 and 26 bits, respectively. Margrethe only produces 4-bit messages at a time, but, given the very low output noise, we can combine multiple streams to produce ciphertexts for larger plaintext spaces. This idea, called *plaintext independent transciphering*, was recently introduced by Méaux *et al.* [MPP23] for FiLIP (which produces only 1 bit at a time) to generate key streams for messages of size up to 2^8.

Our VP-based evaluation method for MFP ciphers introduces noise that is much smaller than previously used bootstrapped methods, including the one from [MPP23]. In this way, we can not only reproduce their technique for message composition but also extend it to messages of up to 26 bits. Table 6 compares our results with them and Figure 2 shows our results for larger messages. We note that our evaluation of Margrethe is not only up to 20 times faster than their evaluation but also enables a much smaller probability of failure (FR) and keys. We use parameter set 1, which requires 128 MB of keys, whereas [MPP23] requires 215 MB for their Set I and 1 GB for their Set II. Additionally, our methods do not require any setup phase at the server while [MPP23] takes up to 6.5s to set up.

Remark 2. We employ threading parallelization only to improve latency but not throughput in our implementation. More specifically, our benchmark uses up to 14 threads to produce a single message (improving latency), but it does not produce multiple messages in parallel (which would improve throughput, but not latency). This is done for two main reasons. First, we want to ensure a fair comparison with [MPP23], which also does not exploit multi-threading to improve throughput. The use of parallelism for improving latency is well established in the MPF literature [CHMS22], and the only reason [MPP23] does not feature internal parallelism is that their construction does not allow it. Second, improving throughput by evaluating multiple instances of the cipher simultaneously would be a trivial result that relates more to available computational resources than to the cipher

Table 6: Margrethe with parameter Set 1 in the plaintext-independent approach [MPP23]. FR is the logarithm of the probability of failure based on the noise introduced by each evaluation, computed with 1000 bits of precision. Time in milliseconds.

Message Size ($\log_2$)	FINAL-FiLIP [MPP23]				**Margrethe**		
	Set I		Set II		Multi thread	Single thread	FR
	Time	FR	Time	FR			
2	13	-150	36	< -1000	3.12	25.2	< -1000
3	18.8	-30	54	< -1000	3.12	25.2	< -1000
4	25.2	-8	71	< -1000	3.12	25.2	< -1000
5			84.6	< -1000	6.24	50.4	< -1000
6			101	-831	6.24	50.4	< -1000
7			117	-209	6.24	50.4	< -1000
8			137	-54	6.24	50.4	< -1000

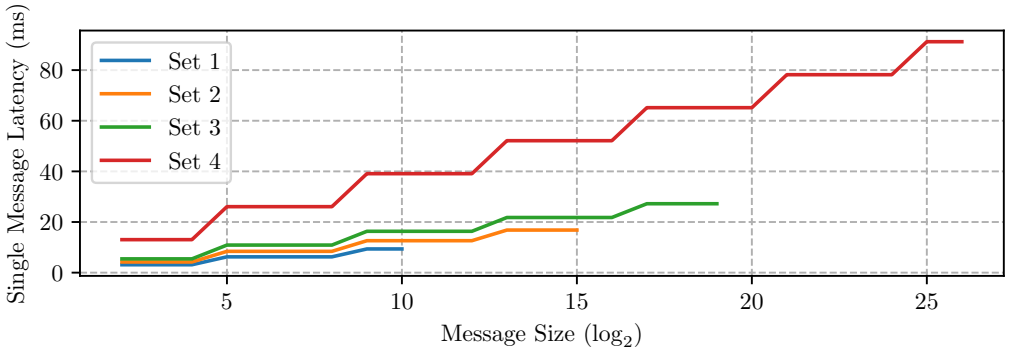

Figure 2: Results for the plaintext-independent approach for messages of up to 26 bits with probability of failure of at most 2^{-64} (computed based on measured output noise).

implementation itself. It would also make it more difficult to compare with other works.

Key compression As we previously discussed, the main disadvantage of the method we present is the requirement for RGSW ciphertexts, which can be significantly larger than the LWE ones used by previous construction. Even with this disadvantage, we still enable the use of keys up to 8 times smaller than [MPP23] (128MB compared with their 1GB keys), for example. Considering numbers from [WWL+24], a setup phase to expand the 2048 RGSW ciphertexts (from small packed (R)LWE ciphertexts) needed by Margrethe would take $2048 \times 0.14 = 287.4$ seconds single-threaded. Their method provides RGSW samples with $\ell = 2$ and noise $\sigma = 2^{19}$ (considering $q = 2^{52}$), which is enough for us to use parameter Set MP2 for messages of up to 8 bits with failure probability of at most 2^{-22}. The execution time of the cipher would be the same as the executions with parameter Set 2.

5.2 Transciphering in multi-party protocols

We consider two main aspects of using our solution in a multiparty protocol. The first is the concrete performance of performing key mixing, which we benchmark and present results in Section 5.2.1. The second is the impact, at the protocol level, of providing RKA security for the cipher.

5.2.1 Key Mixing

We implement key mixing as described in Section 3.2 and benchmark it for the full key of Margrethe. We run the entire key combination (2048 samples) for two keys with parameters Set MP1 and Set MP2 in 430 ms, single-threaded. Notice that each combination is given by 2048 independent execution of Algorithm 4 and, could, therefore, be evaluated in parallel. The evaluation of the cipher would then have the same performance as parameter Set 2, which is only 1.35 times slower than Set 1. We note one could perform the mixing using larger parameter sets to enable the evaluation of the cipher with Set 1, which we don't provide due to implementation limits of the MOSFHET [GBA24] library.

5.2.2 Comparison against protocol-level countermeasures

As an alternative to our approach, one could avoid addressing RKA security by adopting protocol-level countermeasures. This would enable a more flexible choice of cipher (not requiring RKA security) but would require the protocol to avoid generating related key streams. The most straightforward way for achieving it is to simply use different keys at every transchipering instantiation. In the two-RKS setting, this would require the protocol to use twice as many keys, which introduces issues:

- **Key size:** The immediate impact of doubling the number of keys is to increase the key size by a factor of two. Concretely, considering the parameters of Table 4, this would represent an overhead from 256MB up to 2GB in communication for a two-party protocol, and it would increase with the number of parties.

- **Setup phase:** Different ciphers may require different setup phases for expanding keys and performing other types of pre-computation. Key compression algorithms are an example of this. As with the key size, increasing the number of keys would also increase the execution time required for setting up different keys, with the concrete overhead being also up to 2 times the execution time of the setup phase. To exemplify, the setup phase in [CCH+24] takes 25 seconds per key while [MPP23] takes between 5.4 and 17.5 seconds per key. Our method also requires setup phases when compression algorithms are used (as discussed in Section 5.1.1).

One could also avoid providing RKA security by working on a weaker security model, under the assumption that a minimum number of honest parties will always be part of the key generation protocol for transciphering. This solution would prevent related key streams from becoming known to malicious parties, which could suffice for security without requiring additional guarantees from the cipher itself. However, relying on such assumption greatly limits the protocol applicability, as it restricts the overlying application to only work on specific contexts where it is possible to guarantee a minimum number of honest parties and limit corruption, which is not realistic in many use cases.

It is also important to notice that our results demonstrate that the leveled evaluation of Margrethe is the best-performing solution even among ciphers that do not provide RKA security, outperforming the previous state of the art by a factor of up to 21.9 times. Therefore, even if one considers alternative solutions for avoiding RKA security, the use of our evaluation approach can still be the optimal choice.

In summary, our approach provides superior performance in comparison to generic protocol-level countermeasures, but at the cost of analyzing the specific choice of cipher with respect to RKA. Fortunately, we have shown that Margrethe provides suitable performance and security properties within the two-RKS case, which is applicable to the static MPC setting.

Acknowledgments

Pierrick Méaux was supported by the ERC Advanced Grant no. 787390. This work has been funded in part by the ERC Advanced Grant number 101096871. This work is also supported by the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme in the scope of the CONFIDENTIAL6G project under Grant Agreement 101096435. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

References

[AJLA+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, Lecture Notes in Computer Science, pages 483–501, Berlin, Heidelberg, 2012. Springer.

[BCKS24] Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, and Damien Stehlé. Bootstrapping Bits with CKKS. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 94–123, Cham, 2024. Springer Nature Switzerland.

[BCL09] Lilya Budaghyan, Claude Carlet, and Gregor Leander. Constructing new apn functions from known ones. *Finite Fields and Their Applications*, 15(2):150–159, 2009.

[BD94] Thomas. Beth and Cunsheng. Ding. On almost perfect nonlinear permutations. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, pages 65–76, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[BED+22] Javad Bahrami, Mohammad Ebrahimabadi, Jean-Luc Danger, Sylvain Guilley, and Naghmeh Karimi. Leakage power analysis in different s-box masking protection schemes. In Cristiana Bolchini, Ingrid Verbauwhede, and Ioana Vatajelu, editors, *2022 Design, Automation & Test in Europe Conference & Exhibition, DATE 2022, Antwerp, Belgium, March 14-23, 2022*, pages 1263–1268. IEEE, 2022.

[BKSS25] Youngjin Bae, Jaehyung Kim, Damien Stehlé, and Elias Suvanto. Bootstrapping Small Integers With CKKS. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024*, pages 330–360, Singapore, 2025. Springer Nature.

[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.

[BMS12] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A differential fault attack on the grain family of stream ciphers. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 122–139, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[BOS23] Thibault Balenbois, Jean-Baptiste Orfila, and Nigel P. Smart. Trivial transciphering with trivium and TFHE. In Michael Brenner, Anamaria Costache, and Kurt Rohloff, editors, *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Copenhagen, Denmark, 26 November 2023*, pages 69–78. ACM, 2023.

[BS85] László Babai and Vera T. Sós. Sidon sets in groups and induced subgraphs of cayley graphs. *European Journal of Combinatorics*, 6(2):101–114, 1985.

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

[Car22] Claude Carlet. On apn functions whose graphs are maximal sidon sets. In *LATIN 2022: Theoretical Informatics: 15th Latin American Symposium, Guanajuato, Mexico, November 7–11, 2022, Proceedings*, page 243–254, Berlin, Heidelberg, 2022. Springer-Verlag.

[CCF+18] 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. *Journal of Cryptology*, 31(3):885–916, July 2018.

[CCH+24] Mingyu Cho, Woohyuk Chung, Jincheol Ha, Jooyoung Lee, Eun-Gyeol Oh, and Mincheol Son. Frast: TFHE-friendly Cipher Based on Random S-boxes, 2024. Publication info: Preprint.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Multi-key homomorphic encryption from TFHE. In *ASIACRYPT (2)*, volume 11922 of *Lecture Notes in Computer Science*, pages 446–472. Springer, 2019.

[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.

[CDSU23] Gaëtan Cassiers, Henri Devillez, François-Xavier Standaert, and Balazs Udvarhelyi. Efficient regression-based linear discriminant analysis for side-channel security evaluations: Towards analytical attacks against 32-bit implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023, Issue 3:270–293, 2023.

[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, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33, 2016.

[CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 377–408, Cham, 2017. Springer International Publishing.

[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 Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 640–669. Springer, 2021.

[CHMS22] Orel Cosseron, Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. Towards case-optimized hybrid homomorphic encryption - featuring the elisabeth stream cipher. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part III*, volume 13793 of *Lecture Notes in Computer Science*, pages 32–67. Springer, 2022.

[CM03] Nicolas T Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22*, pages 345–359. Springer, 2003.

[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.

[Cou03] Nicolas T Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23*, pages 176–194. Springer, 2003.

[CP08] Christophe De Cannière and Bart Preneel. Trivium. *LNCS, New Stream Cipher Designs - The eSTREAM Finalists*, page 244–266, 2008.

[CT18] Benoît Cogliati and Titouan Tanguy. Multi-user security bound for filter permutators in the random oracle model. *Designs, Codes and Cryptography*, 09 2018.

[DDK+23] Morten Dahl, Daniel Demmler, Sarah El Kazdadi, Arthur Meyre, Jean-Baptiste Orfila, Dragos Rotaru, Nigel P. Smart, Samuel Tap, and Michael Walter. Noah's ark: Efficient threshold-fhe using noise flooding. In *WAHC@CCS*, pages 35–46. ACM, 2023.

[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, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 457–475. Springer, 2016.

[GAH+23] Lorenzo Grassi, Irati Manterola Ayala, Martha Norberg Hovd, Morten Øygarden, Håvard Raddum, and Qingju Wang. Cryptanalysis of symmetric primitives over rings and a key recovery attack on rubato. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III*, volume 14083 of *Lecture Notes in Computer Science*, pages 305–339. Springer, 2023.

[GBA24] Antonio Guimarães, Edson Borin, and Diego F. Aranha. MOSFHET: Optimized Software for FHE over the Torus. *Journal of Cryptographic Engineering*, July 2024.

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

[GHBJR23] Henri Gilbert, Rachelle Heim Boissier, Jérémy Jean, and Jean-René Reinhard. Cryptanalysis of elisabeth-4. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 256–284. Springer, 2023.

[HMS23] Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. The patching landscape of elisabeth-4 and the mixed filter permutator paradigm. In Anupam Chattopadhyay, Shivam Bhasin, Stjepan Picek, and Chester Rebeiro, editors, *Progress in Cryptology - INDOCRYPT 2023 - 24th International Conference on Cryptology in India, Goa, India, December 10-13, 2023, Proceedings, Part I*, volume 14459 of *Lecture Notes in Computer Science*, pages 134–156. Springer, 2023.

[HR08] Michal Hojsík and Bohuslav Rudolf. Differential fault analysis of trivium. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2008.

[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.

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

[Méa22] Pierrick Méaux. On the algebraic immunity of direct sum constructions. *Discret. Appl. Math.*, 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, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory*

and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I, volume 9665 of *Lecture Notes in Computer Science*, pages 311–343. Springer, 2016.

[MPP23] Pierrick Méaux, Jeongeun Park, and Hilder V. L. Pereira. Towards Practical Transciphering for FHE with Setup Independent of the Plaintext Space, 2023. Publication info: Published elsewhere. Communications in Cryptology.

[MR24] Pierrick Méaux and Dibyendu Roy. Theoretical differential fault attacks on flip and filip. *Cryptography and Communications*, pages 1936–2455, 2024.

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

[MW24] Pierrick Méaux and Qingju Wang. Extreme algebraic attacks. *IACR Cryptol. ePrint Arch.*, page 64, 2024.

[NK93] Kaisa Nyberg and Lars Ramkilde Knudsen. Provable security against differential cryptanalysis. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, pages 566–574, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

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

[Nyb94] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, pages 55–64, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[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*, 72(8):2418–2425, 2023.

[Sma23] Nigel P. Smart. Practical and efficient fhe-based MPC. In *IMACC*, volume 14421 of *Lecture Notes in Computer Science*, pages 263–283. Springer, 2023.

[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.

[WLW+24] Benqiang Wei, Xianhui Lu, Ruida Wang, Kun Liu, Zhihao Li, and Kunpeng Wang. Thunderbird: Efficient Homomorphic Evaluation of Symmetric Ciphers in 3GPP by combining two modes of TFHE. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(3):530–573, July 2024. Number: 3.

[WWL+24] Ruida Wang, Yundi Wen, Zhihao Li, Xianhui Lu, Benqiang Wei, Kun Liu, and Kunpeng Wang. Circuit Bootstrapping: Faster and Smaller. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 342–372, Cham, 2024. Springer Nature Switzerland.

Supplementary Material

A Key extension functions and security beyond two-RKS

As seen in Section 4.5, the security of Margrethe degrades quickly if an adversary has access to more than two related keystreams. In the following, we investigate key derivation methods to strengthen the security of the design against related key attacks.

The goal of a Key Extension Function (KEF) in our context is to extend keys with sufficient entropy (e.g. t bits) into binary vectors of size N that can be used as keys for Margrethe for the many-RKS setting. The extension should be efficient to compute and verify. To determine the most suitable KEF, we rely on the concept of a Sidon set. If the derived keys form a Sidon set, no derivative of order greater than one can be applied to the entire keystream. This allows us to utilize the results from Section 4.3. While avoiding derivatives of order greater than one might be excessive (as other low-order derivatives may not compromise security), we find the idea of a KEF generating Sidon sets both interesting and challenging, noting that it has not been explored to our knowledge.

The primary challenge for a KEF compatible with Margrethe's design is that, in addition to deriving keys that form a Sidon set, we want most of the n-out-of-N subsets of these 2^t keys to also form Sidon sets. This is to avoid finding specific derivatives for each element of the keystream indexed by i.

We propose an initial approach using Almost Perfectly Nonlinear (APN) functions, which are extensively used in the study and design of block ciphers. However, so far this approach does not guarantee resistance beyond two related keystreams for Margrethe, so we only present it as an interesting direction.

In the following we explain the approach of using Sidon sets to study the security of RKA with multiple keystreams. We introduce a candidate solution of KEF for Margrethe using almost perfectly nonlinear functions and study its properties.

First, we connect the maximum order of derivative the adversary can use to a property on the set of related keys. We denote by $S_K = \{K, K+\Delta_1, \ldots, K+\Delta_J\}$ the set of related keys, and by $\mathsf{AD}(S_K)$ the maximum over the dimension of the affine spaces contained in S_k:

$$\mathsf{AD}(S_k) = \max\{t \,|\, \exists s_1, \ldots, s_t \in S_K \,|\, \forall v \in \mathbb{F}_2^t \; K + v \cdot (s_1 + K, \ldots, s_t + K) \in S_k\}.$$

Then, $\mathsf{AD}(S_k)$ is the maximum order of derivative that can be used on all keystream bits (as in Equation 4). We show how the notion of Sidon sets can be used to get $\mathsf{AD}(S_k)$ as low as possible, which prevents from using high order derivatives for all keystream bits.

Definition 8 (Sidon Set [BS85] Definition 1)**.** We call a subset S of an Abelian group G a Sidon set if for any x, y, z, and $w \in S$ of which at least three are different: $x + y \neq z + w$

In our context, by a Sidon set of $\mathbb{F}_2^n$ we refer to a Sidon set of the group of binary strings of fixed length n with operation the component-wise XOR.

Remark 3. If S is a Sidon set of $\mathbb{F}_2^n$, using a permutation on the bits of the elements of S, or appending the same vector to all the elements of S, still gives a Sidon set.

Proposition 4. *Let n a positive integer and S such that $|S| > 1$ a subset of $\mathbb{F}_2^n$. If S is a Sidon set then $\mathsf{AD}(S) = 1$ and*

Proof. We proceed by contradiction. If $\mathsf{AD}(S) > 1$ then there exist a, b, c and $d \in S$ such that $a + b + c + d = 0$, that is $a + b = c + d$ with a, b, c, d different elements of S, hence S is not a Sidon set. It allows us to conclude $\mathsf{AD}(S) \leq 1$. Furthermore, since $|S| > 1$, $\mathsf{AD}(S) = 1$ since any pair of elements correspond to an affine space of dimension 1. □

In the case of Margrethe, each keystream element depends only on a subset of size n of the key. Accordingly, the adversary obtains equations depending on (permuted) substrings of the elements of S_K, then for each i we are interested in the probability of S_{K^i} to be a Sidon set. Based on Remark 3, we can study cases where the Sidon set property is preserved.

We propose a key derivation function using Almost Perfect Nonlinear (APN) functions. APN functions are well-studied vectorial Boolean functions, known for providing optimal resistance against differential cryptanalysis when used as substitution boxes in block ciphers. They have been extensively studied due to their implications in cryptography and coding theory, e.g. [NK93,Nyb94,BD94,BCL09,Car21]. A function F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^n$ is called APN if for every nonzero $a \in \mathbb{F}_2^n$ and every $b \in \mathbb{F}_2^n$, the equation $\mathsf{D}_a F(x) = F(x) + F(x+a) = b$ has at most two solutions. In the following, we use the link between Sidon set and APN as highlighted in [Car22]: F is APN if and only if $\{(x, F(x)) \,|\, x \in \mathbb{F}_{2^n}\}$ is a Sidon set in $\mathbb{F}_2^{2n}$.

Definition 9. Key Derivation Function based on APN functions Let $t, u, v, N \in \mathbb{N}$ such that $t(u+v)$ divides N. We denote by $\mathsf{KDF}_{t,u,v}$ a (vectorial Boolean) function from $\mathbb{F}_2^t$ to $\mathbb{F}_2^{t(u+v)}$ that associate to $x \in \mathbb{F}_2^t$ the vector consisting in u copies of x and $F_i(x)$ for $i \in [1, v]$ where the F_i are distinct t-variable APN functions, that is $x \mapsto (x, \ldots, x, F_1(x), \ldots, F_v(x))$.

We denote by $\mathsf{KDF}_{t,u,v,N}$ the function from $\mathbb{F}_2^{tN/(u+v)}$ to $\mathbb{F}_2^N$ which output is obtained by the concatenation of $N/(t(u+v))$ functions $\mathsf{KDF}_{t,u,v}$.

From the general KEF construction from Definition 9 we focus on three particular cases:

1. $\mathsf{KDF}_{t,1,N/t-1,N}$ which outputs vectors $(x, F_1(x), \ldots, F_{N/t-1}(x))$ where the F_i are chosen such that for all $1 \leq i < j < N/t$ the set $\{(F_i(x), F_j(x)) \,|\, x \in \mathbb{F}_2^n\}$ is a Sidon set of $\mathbb{F}_2^{2t}$.

2. $\mathsf{KDF}_{t,1,1,N}$ which outputs vectors $(x, F_1(x), y, F_1(y), \ldots, z, F_1(z))$.

3. $\mathsf{KDF}_{t,N/2t,N/2t,N}$ which outputs vectors $(x, \ldots, x, F_1(x), \ldots, F_{N/2t}(x))$.

For these cases, considering the KEF is used to extend the key for Margrethe, we study the probability that the subsets selected for each keystream belong to a Sidon set. We denote by $S_{n,t,r}$ the number of solutions to $x_1 + \ldots + x_r = n$ where $x_i \in \mathbb{N}$, $x_i \leq t$ for $i \in [1, r]$. It corresponds to the number of ways to put n balls into r bins with maximum occupancy of t (a variant of Stirling numbers of the second kind) and can be computed as:

$$S_{n,t,r} = \sum_{0 \leq k \leq n/t \leq r} (-1)^k \binom{r}{k} \binom{n + r - 1 - k(t+1)}{n - k(t+1)}.$$

Proposition 5. *Let $t, u, v, N \in \mathbb{N}$ such that $u + v$ divides N, and $F = \mathsf{KDF}_{t,u,v,N}$ be a KEF as defined in Definition 9, then the probability of the set $\{F(x) \,|\, x \in \mathbb{F}_2^{tN/(u+v)}\}$ to be a Sidon set of $\mathbb{F}_2^N$ is at least:*

1. $p \geq \frac{S_{n-2t,t,N/t-2}}{S_{n,t,N/t}}$ *for $\mathsf{KDF}_{t,1,N/t-1,N}$ where each pair of APN functions gives a Sidon set.*

2. $p \geq \frac{S_{n-2t,2t,N/2t-1}}{S_{n,2t,N/2t}}$ *for $\mathsf{KDF}_{t,1,1,N}$.*

3. $p \geq \sum_{k=0}^{n} \frac{\binom{N/2}{k}\binom{N/2}{n-k}}{\binom{N}{n}} \frac{S_{k-t,t,N/2t-1}}{S_{k,t,N/2t}} \frac{S_{n-k-t,t,N/2t-1}}{S_{n-k,t,N/2t}}$ *for $\mathsf{KDF}_{t,N/2t,N/2t,N}$.*

Proof. For the first case, the KEF outputs vectors of shape $(x, F_1(x), \ldots, F_{N/t-1}(x))$ where any pair of blocs of t elements of shape x or $F_i(x)$ gives a Sidon set. Since appending a vector a permuting the bits does not alter the property of being a Sidon set (see Remark 3), all permuted subset of the output containing 2 entire blocs is a Sidon set. Then, we can count the number of ways to select n bits from the N bit key such that all bits of at least two of the N/t blocs are selected using the balls into bin approach, with bins with maximal occupancy t. The numbers of ways the n selected bits for each equation are distributed over the N/t blocs is given by $S_{n,t,N/t}$. Then, the positive outcome in our context is when any 2 bins are totally filled, it corresponds to the number of ways the remaining $n-2t$ key bits can be distributed over $N/t-2$ bins of maximum occupancy t, that is $S_{n-2t,t,N/t-2}$. It allows to derive the lower bound for this case.

For the second case, the KEF outputs a vector composed of $N/2t$ blocs of size $2t$ of shape $x, F_1(x)$. Any of this bloc is a Sidon set of $\mathbb{F}_2^{2t}$, then we bound the property of one bloc to be selected to ensure the Sidon property of the KEF output of $\mathbb{F}_2^N$. Using the same combinatorial approach, there are $S_{n,2t,N/2t}$ ways to fill the $N/2t$ blocs of size $2t$ with n key bits, and $S_{n-2t,2t,N/2t-1}$ ways such that at least one bloc is filled. It allows to conclude for this case $p \geq S_{n-2t,2t,N/2t-1}/S_{n,2t,N/2t}$.

For the third case, the KEF outputs a vector where the first half are copies of x and the second ones distinct APN functions applied on x. Then, any subset containing one bloc of x and one bloc $F_i(x)$ is a Sidon set. We bound the probability of this event by considering the probability that an entire bloc of x is obtained in the first half and the probability that an entire bloc $F_i(x)$ is obtained in the second half, considering the partition of n into the two halves. The probability of k of the n bits to be selected in the first half (and $n-k$ in the second) is given by: $\binom{N/2}{k}\binom{N/2}{n-k}/\binom{N}{n}$. Then, using the balls into bins approach, with k bits selected the probability of having a full bloc selected is given by $S_{k-t,t,N/2t-1}/S_{k,t,N/2t}$. It allows to derive the final bound:

$$p \geq \sum_{k=0}^{n} \frac{\binom{N/2}{k}\binom{N/2}{n-k}}{\binom{N}{n}} \frac{S_{k-t,t,N/2t-1}}{S_{k,t,N/2t}} \frac{S_{n-k-t,t,N/2t-1}}{S_{n-k,t,N/2t}}.$$

□

The advantages of an APN-based KEF compared to approaches using 2^t random keys or keys extended with hash functions would be the following:

- The APN-based KEF is the only one that reliably produces a Sidon set, ensuring that the same relation is not used for all keystream elements.
- The computation of APN functions, such as Gold power functions over $\mathbb{F}_{2^t}$, can be efficiently performed and verified.
- The probability that n-out-of-N subsets are (or are not) Sidon sets can be bounded.

B MPC-FHE protocol

We summarize the subprotocols within MPC-FHE by [Sma23]. We preserve most of the original notation, but simplify the descriptions to only include the relevant steps and to write the symmetric encryption portions in terms of a stream cipher Sym. In particular, we omit counters, nonces and session identifiers as encryption arguments for simplicity; and denote by $\mathcal{F}_{\mathsf{KeyGenDec}}$ the ideal functionality for distributed key generation and decryption.

Protocol Π_{init}

1. Parties call the functionality $\mathcal{F}_{\mathsf{KeyGenDec}}$ to obtain the public key $\mathfrak{pk}$.
2. All parties $\mathcal{P}_i \in \mathbb{P}$ generate a random symmetric key $k_i \in \mathcal{R}^\rho$, encrypt it using $\mathsf{Enc}(k_i, \mathfrak{pk}; r_i)$ to obtain $\mathfrak{ct}_i \in \mathcal{E}^\rho$. The ciphertext $\mathfrak{ct}_i$ is broadcast to all parties.
3. All parties $\mathcal{P}_i \in \mathbb{P}$ call the prover $\pi \leftarrow \mathsf{Prv}(\mathfrak{ct}, (m, r))$ with input the respective ciphertexts $\mathfrak{ct}_i$, the respective plaintexts k_i and randomness r_i in order to obtain proofs π_i.
4. The proofs are broadcast to the parties in $\mathbb{P}$, which verify them using $\mathsf{Ver}(\mathfrak{ct}, \pi)$. If any proof fails then the parties in $\mathbb{P}$ replace the associated ciphertext with a default encryption of zero.
5. Set $\mathfrak{ct}_0 = \sum_{i=1}^{n} \mathfrak{ct}_i$ as an encryption of $k_0 = \sum_{i=1}^{n} k_i$.

Protocol Π_{input}

$\mathsf{Input}(\mathsf{varid}, x)$:

1. If $\mathsf{type}(\mathsf{varid}) = \mathsf{Enc}$ then
   (a) Player $\mathcal{P}_i$ encrypts x using $\mathfrak{ct}_x \leftarrow \mathsf{Enc}(x, \mathfrak{pk}, r_x)$ with randomness r_x.
   (b) Player $\mathcal{P}_i$ broadcasts $\mathfrak{ct}_x$ to all parties in $\mathbb{P}$.
   (c) Player $\mathcal{P}_i$ invokes the prover $\pi_x \leftarrow \mathsf{Prv}(\mathfrak{ct}_x, (x, r_x))$ with input the ciphertext $\mathfrak{ct}_x$, the message x and the associated randomness r_x, and broadcasts the resulting proof π_x to all parties in $\mathbb{P}$.
   (d) If the proof verifies when calling $\mathsf{Ver}(\mathbf{c}_x, \pi_x)$, then all parties in $\mathbb{P}$ store $\mathbf{c}_x$ under a suitable label varid, otherwise they store a default encryption of zero.
2. If $\mathsf{type}(\mathsf{varid}) = \mathsf{Sym}$ then
   (a) Player $\mathcal{P}_i$ encrypts x using $\mathbf{c}_x \leftarrow x \oplus \mathsf{Sym}(k_i)$.
   (b) Player $\mathcal{P}_i$ broadcasts $\mathbf{c}_x$ to all parties in $\mathbb{P}$
   (c) The parties in $\mathbb{P}$ transcipher $\mathbf{c}_x$ from an encryption under Sym to an encryption under Enc by homomorphically computing $\mathfrak{ct}_x \leftarrow \mathbf{c}_x \oplus \mathsf{Sym}(\mathfrak{ct}_i)$.

Protocol Π_{output}

$\mathsf{Output}(\mathsf{varid}, j)$:

1. If $\mathsf{type}(\mathsf{varid}) = \mathsf{Enc}$ then
   (a) All parties in $\mathbb{P}$ invoke $\mathsf{DistDecrypt}(\mathfrak{ct}_y, \{\mathcal{P}_j\})$, on functionality $\mathcal{F}_{\mathsf{KeyGenDec}}$, where $\mathfrak{ct}_y$ is the contents of the variable.
   (b) Party $\mathcal{P}_j$ receives the output plaintext value y and takes this as the output value.
2. If $\mathsf{type}(\mathsf{varid}) = \mathsf{Sym}$ then
   (a) All parties in $\mathbb{P}$ take the contents $\mathfrak{ct}_y$ of the variable and compute homomorphically $\mathfrak{ct}'_y \leftarrow \mathfrak{ct}_y \oplus \mathsf{Sym}(\mathfrak{ct}_j)$
   (b) All parties in $\mathbb{P}$ invoke $\mathbf{c}_y \leftarrow \mathsf{DistDecrypt}(\mathfrak{ct}'_y, \mathbb{P})$, with output player $\mathcal{P}_j$.
   (c) Player $\mathcal{P}_j$ decrypts $\mathbf{c}_y$ to obtain y by computing $y \leftarrow \mathbf{c}_y \oplus \mathsf{Sym}(k_j)$.

Protocol $\Pi_{declassify/transcipher}$

$\mathsf{Declassify}(\mathsf{varid}_y, \mathsf{varid}_x)$:

1. The parties in $\mathbb{P}$ retrieve $\mathfrak{ct}_x$ from varid_x and execute $\mathsf{DistDecrypt}(\mathfrak{ct}_x, \mathbb{P})$ on $\mathcal{F}_{\mathsf{KeyGenDec}}$.
2. The output y they receive is assigned to the register with the corresponding label varid_y.

$\mathsf{Transcipher}^{s \to t}(\mathsf{varid}_y, \mathsf{varid}_x)$:

1. The parties in $\mathbb{P}$ retrieve $\mathfrak{ct}_x$ from varid_x (an FHE-encrypted ciphertext) and homomorphically compute $\mathfrak{ct}'_x \leftarrow \mathfrak{ct}_x \oplus \mathsf{Sym}(\mathfrak{ct}_0)$.
2. The parties in $\mathbb{P}$ execute $\mathbf{c} \leftarrow \mathsf{DistDecrypt}(\mathfrak{ct}'_x, \mathbb{P})$ on $\mathcal{F}_{\mathsf{KeyGenDec}}$.
3. The output $\mathbf{c}$ they receive is assigned to the register with the corresponding label varid_y.

$\mathsf{Transcipher}^{t \to s}(\mathsf{varid}_y, \mathsf{varid}_x)$:

1. The parties in $\mathbb{P}$ retrieve $\mathbf{c}_x$ from varid_x (a symmetric-encrypted ciphertext) and homomorphically compute $\mathfrak{ct}_y \leftarrow \mathbf{c}_x \oplus \mathsf{Sym}(\mathfrak{ct}_0)$.
2. The output $\mathfrak{ct}_y$ is assigned to the register varid_y.