Towards Practical Transciphering for FHE with Setup Independent of the Plaintext Space

Pierrick Méaux[1], Jeongeun Park[2], and Hilder V. L. Pereira[3]

[1] University of Luxembourg, Luxembourg
[2] imec-COSIC, KU Leuven, Leuven, Belgium
[3] Universidade Estadual de Campinas, Campinas, Brazil

pierrick.meaux@uni.lu,
jeongeun.park@kuleuven.be,
hilder@unicamp.br

Abstract. Fully Homomorphic Encryption (FHE) is a powerful tool to achieve non-interactive privacy preserving protocols with optimal computation/communication complexity. However, the main disadvantage is that the actual communication cost (bandwidth) is high due to the large size of FHE ciphertexts. As a solution, a technique called transciphering (also known as Hybrid Homomorphic Encryption) was introduced to achieve almost optimal bandwidth for such protocols. However, all of existing works require clients to fix a precision for the messages or a mathematical structure for the message space beforehand. It results in unwanted constraints on the plaintext size or underlying structure of FHE based applications.
In this article, we introduce a new approach for transciphering which does not require fixed message precision decided by the client, for the first time. In more detail, a client uses any kind of FHE-friendly symmetric cipher for $\{0,1\}$ to send its input data encrypted bit-by-bit, then the server can choose a precision p depending on the application and homomorphically transforms the encrypted bits into FHE ciphertexts encrypting integers in $\mathbb{Z}_p$. To illustrate our new technique, we evaluate a transciphering using FiLIP cipher and adapt the most practical homomorphic evaluation technique [CCS'22] to keep the practical latency. As a result, our proof-of-concept implementation for p from 2^2 to 2^8 takes only from 13 ms to 137 ms.

1 Introduction

As fully homomorphic encryption (FHE) allows any computation over encrypted data, it can be applied to *secure outsourced computation* where a server which has strong computational resources do (requested) computation over a client's data keeping the client's privacy. It unlocked many real-world applications recently such as privacy preserving machine learning [10,36,37,17,5,30,35] and secure outsourced storage [9,16].

Although the latency of such protocol has been considered practical enough, the network cost is still not desirable due to the large size of FHE ciphertexts. In fact, FHE suffers from a large ciphertext expansion, the amount of data that a client has to upload to the server is typically huge. For example, in the worst case, an FHE ciphertext encrypting one single bit costs 2.5KB, if we use a small TFHE [13] ciphertext achieving 128 bits of security.

To solve this problem, the transciphering (a.k.a. Hybrid Homomorphic Encryption (HHE)) approach has been proposed [33]: a client encrypts the data using some block or stream cipher Π, and sends the ciphertexts to the server. Because these ciphers have ciphertext expansion close to one or exactly one, now the upload size is almost the same as the size of the data itself. The server then runs the decryption of Π homomorphically, using an FHE scheme, obtains $\mathsf{FHE.Enc}(m)$, and can then proceed with the usual homomorphic computation.

One way of implementing transciphering is by simply asking the client to encrypt the data using some well-known cipher, like AES. However, it is generally very expensive to evaluate the decryption of such ciphers using FHE. Hence, multiple works have proposed transciphering strategies by first constructing FHE-friendly ciphers [2,7,32,19,27,20,31,24,15,25,3,18], whose decryption function can be evaluated homomorphically using little memory and time. However, all of these FHE-friendly ciphers *fix a plaintext space* (denoted by $\mathcal{M}$) to obtain efficiency gains during the FHE evaluation or to fit with the constraints of the security analysis. For example, most of them [2,19,31,27] are designed for $\mathcal{M} = \mathbb{F}_2$. For larger space, PASTA [20] requires $\mathcal{M} = \mathbb{F}_p^k$, for a positive integer k and a prime p such that $p-1$ is not divisible by 3, and HERA [15] requires $\mathcal{M} = \mathbb{Z}_t$ where $t \geq 2^{16}$.

These natural strategies incur some inconveniences for the versatility of the computations to evaluate, since each application has its ideal plaintext space, that can also evolve based on the client requests. For example, if we want to evaluate binary circuits, we set $\mathcal{M} = \mathbb{F}_2$, if we want to work with bytes, we set $\mathcal{M} = \mathbb{Z}_{2^8}$, etc. Thus, to use transciphering without relying on a sole model of computation, the client would have to be able to implement several different ciphers, depending on each application. For illustration we consider the following scenario: medical doctors/researchers handling patients' private data want to study a relation between certain diseases and a specific human genome sequence via secure machine learning algorithm, keeping individual patient's privacy. Patients' data is already stored up to certain number of bits (let's say 32 bits). Depending on which algorithm the computing party uses and accuracy rate they want to achieve, the input precision differs. For example, the recent secure neural network instantiation [35] uses 8-bit integer for their inputs which is enough to achieve over 90% accuracy, and [17,36] uses 11-16 bits for private decision tree evaluation. We argue why it would be a pain for both the server and client to use different ciphers for the different computations.

First, it means that the client would have to generate and manage several keys and run the setup of the transciphering for each plaintext space appropriate for the desired computations. The setup part of a transciphering is expensive (it is the bottleneck in bandwidth for the HHE protocol e.g. [20]) and this cost is amortized because the setup is run only once in the full protocol. However, this is no longer true if the client has to run several different setups. Then, working with multiple different ciphers is far from optimal from the security point of view, because the security of the full protocol relies on the security of all these ciphers. The full protocol is secure only if all of them are secure, alone and combined, since the same data is encrypted with the different schemes. Assuming the security of multiple ciphers alone and combined is a stronger assumption than relying on the security of a sole cipher.

Finishing on the downside of fixed plaintext space for transciphering, the aforementioned message spaces of existing FHE-friendly ciphers do not always match the most common plaintext spaces and structures used by applications. Namely, since general purpose CPUs use bytes, 32-bit or 64-bit integers, we expect to use these data types most of the times in our applications. Thus, the FHE schemes would have to use the rings $\mathbb{Z}_{2^8}$, $\mathbb{Z}_{2^{32}}$, and $\mathbb{Z}_{2^{64}}$ as message spaces. Moreover, FHE-friendly symmetric ciphers are designed to fit the particularities of one FHE scheme, and the best performances in terms of latency and throughput of one transciphering are obtained for the transciphering standing alone. One downside is that getting these best transciphering performances impacts the choice of parameters of the chosen FHE scheme, which focuses the optimization on the symmetric cipher rather than on the evaluation of the functions that the server will have to compute later on.

Therefore, it would be ideal if there were an efficient transciphering technique which does not require FHE scheme's message space as a parameter on the client's side, so that the server can

transform client's given data into any FHE ciphertext of which message space fits into various applications on the fly.

1.1 Our Contributions

In this article, we introduce a possible solution to achieve the aforementioned ideal case for the first time. Our solution is to "compose" bits into an integer homomorphically. In more detail, a client encrypts each bit of its input data separately using an FHE-friendly cipher for $\mathbb{Z}_2$, then upload those ciphertexts. The server homomorphically transforms them into FHE ciphertexts encrypting bits (where by bits we refer to 0 and 1 integer values, without the structure of $\mathbb{F}_2$) and homomorphically composes them into an integer, by taking some of given encrypted bits, depending on the precision that a target application requires.

Since a client only sends the bit representation of its data, regardless of its original data size, message space for applications based on FHE is not specified beforehand. Let's assume that the client's input data size is initially set to t bits. If an application which the client targets requires $\mathcal{M} = \mathbb{Z}_p$, where $\log p \leq t$, the server grabs the upper $\log p$ bits from given client's data stored as FHE-friendly cipher, and transforms them into an FHE ciphertext encrypting message in $\mathcal{M}$ via our transciphering technique.

In our instantiation, we use FiLIP cipher [31] for the client's side since its homomorphic decryption is already optimized by [17] for a practical application, which only takes 2.6 milliseconds per bit. Therefore, their little computational overhead is still preserved in our case. Moreover, we tweak their approach to directly produce an FHE ciphertext encrypting a bit scaled with a corresponding power of 2, instead of computing $2^j \cdot \mathsf{FHE.Enc}(b_j) = \mathsf{FHE.Enc}(2^j \cdot b_j)$, where $b_j \in \{0, 1\}$, after transciphering, to minimize additional computation overhead.

We implement our result as a proof of concept by using FINAL [4] for underlying FHE scheme and choose message precision $\log p$ in the range from 2 to 8 bits. The corresponding running time of server is in the range from 6.5ms to 18ms per bit by only using single thread. Since we cannot directly use [17]'s optimization for efficient compositing technique, we have more computational overhead than their result. Moreover, the choice of parameters differs for different $\log p$ to manage the noise growth, which affects on computation time. Compared to the most recent transciphering result (Elisabeth-4 [18] which is designed for 4-bit integer only) of which best performance is 371 ms per bit without parallel computation, our method is faster and easier to adapt to different use-cases.

1.2 Technical Overview

We start with a bit representation of integer data elements. In the client side, an integer $\mu \in \mathbb{Z}^+_{2^t}$ is decomposed into t bits $b_0, \ldots, b_{t-1}$ such that $\sum_{j=0}^{t-1} b_j \cdot 2^j = \mu$. Then, each b_j is encrypted with FiLIP cipher, generating a ciphertext c_j, which is sent to the server. On the server side, depending on the target application, a precision $\log p$ is chosen, and the $\log p$ most significant bits of μ are considered. The goal is to generate an FHE ciphertext encrypting $\bar{\mu} := \mu - (\mu \mod 2^{t-\log p})$. To do so, we proceed in two steps: Firstly, we homomorphically evaluate a modified version of FiLIP's decryption so that instead of just generating $\mathsf{FHE.Enc}(b_j)$ for the desired bits, we generate $\mathsf{FHE.Enc}(2^j \cdot b_j)$, where the message space is $\mathbb{Z}_p$. This is the main step. Second, it just remains to homomorphically add all those ciphertexts, so that we obtain

$$\sum_{j=t-\log p}^{t-1} \mathsf{FHE.Enc}(2^j \cdot b_j) = \mathsf{FHE.Enc}(\bar{\mu}).$$

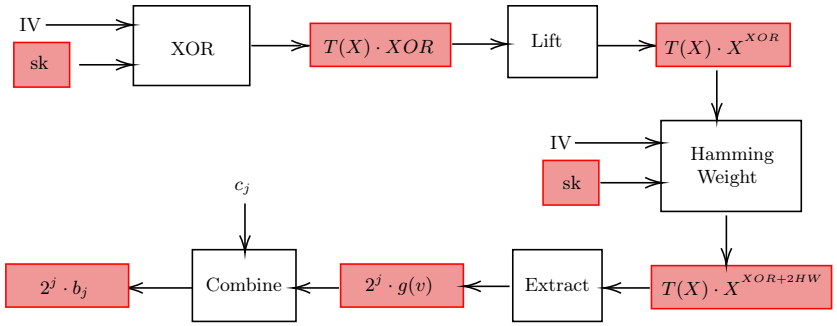

Fig. 1: *Main pieces of the first step of our homomorphic decryption, where we transform a FiLIP ciphertext c_j encrypting a bit b_j with initialization vector IV and secret key sk into an FHE encryption of $2^j \cdot b_j$. The red boxes represent values encrypted with FHE.*

First of all, notice that FiLIP's decryption requires evaluating a Boolean function g with a secret vector v derived[4] from the secret key sk. Thus, it would be natural to work with an FHE scheme whose message space is binary. However, we have to adapt the decryption to work modulo p. More specifically for the instances of FiLIP we consider $g(v)$ consists of two main sub-functions: one that computes the XOR operation of the k bits of v, denoted by $x := \mathsf{XOR}(x_1, \ldots, x_k)$, and the other is a Boolean threshold function, denoted by $y := \mathbf{T}_{d,s}(\mathbf{y})$, which outputs 1 if the Hamming weight of $\mathbf{y}$, the last s bits of v, is equal to or greater than d. The results of these two subfunctions are them xored.

Since the threshold function involves a comparison, it is hard to evaluate it homomorphically. Thus, our strategy is to use homomorphic look-up-tables to evaluate it. In more detail, we use the standard technique of mapping an encryption of X^m to an encryption of $f(m)$ by multiplying by a so-called "test polynomial" $T(X)$ that depends on f. Thus, we define $T(X)$ with respect to the function g of FiLIP's decryption and develop an arithmetic gate that computes the XORs already multiplied by $T(X)$, so that applying it k times, we have $\mathsf{FHE.Enc}(T(X) \cdot \mathsf{XOR}(x_1, \ldots, x_k))$, which we can lift to the exponent of X, obtaining $\mathsf{FHE.Enc}(T(X) \cdot X^{\mathsf{XOR}(x_1,\ldots,x_k)})$. Then, we multiply this ciphertext by encryptions of X^{2y_i}, for each of the last s bits of v, denoted y_i. With this, we obtain $\mathsf{FHE.Enc}(T(X) \cdot X^{\mathsf{XOR}(x_1,\ldots,x_k)+2\cdot \mathsf{W_H}(\mathbf{y})})$. But, due to the way $T(X)$ is defined, this is basically the same as $\mathsf{FHE.Enc}(2^j \cdot g(v))$, thus, we just have to combine this result with the FiLIP ciphertext c_j encrypting the bit b_j, to finally produce $\mathsf{FHE.Enc}(2^j \cdot b_j)$. We illustrate this process in Figure 1.

2 Preliminaries

2.1 Vectors, matrices, distributions

Notation: We use lower-case bold letters for vectors and upper-case bold letters for matrices. A zero vector is denoted by $\mathbf{0}$. The inner product of two vectors $\mathbf{a}$ and $\mathbf{b}$ is denoted by $\mathbf{a} \cdot \mathbf{b}$ (or $\langle \mathbf{a}, \mathbf{b} \rangle$). For any vector $\mathbf{u}$, $\|\mathbf{u}\|$ denotes the infinity norm. We denote the dot product of two vectors $\mathbf{v}, \mathbf{w}$ by $\langle \mathbf{v}, \mathbf{w} \rangle$. For a vector $\mathbf{x}$, $\mathbf{x}[i]$ or x_i denotes the i-th component scalar of $\mathbf{x}$. We use the Euclidean norm as a default norm for a vector $\mathbf{x}$.

[4] The transformation from sk to v is simple homomorphically, with no impact on the error, therefore skipped in the overview

Subgaussian distribution. For the analysis of noise of each homomorphic operation, we need subgaussian random variables over $\mathbb{R}$.

Definition 1. *A random variable V over $\mathbb{R}$ is σ-subgaussian if its moment generating function satisfies*

$$\mathbb{E}[\exp(t \cdot V)] \leq \frac{1}{2}\exp(\sigma^2 \cdot t^2)$$

for all $t \in \mathbb{R}$.

From the definition, we can prove that the variance of V, denoted by $\mathsf{Var}(V)$ is bounded by σ^2, i.e. $\mathsf{Var}(V) \leq \sigma^2$. Informally, the tails of V are dominated by a Gaussian function with standard deviation σ. We use the property that a vector with subgaussian coordinates is also subgaussian for our noise analysis, which is proved in [29]. Subgaussian random variables have an important property called Pythagorean additivity. Given two random variables, α-subgaussian X and β-subgaussian Y, and $a, b \in \mathbb{Z}$, the random variable $a \cdot X + b \cdot Y$ is $\sqrt{a^2 \cdot \alpha^2 + b^2 \cdot \beta^2}$-subgaussian. It implies that

$$\mathsf{Var}(a \cdot X) + \mathsf{Var}(b \cdot Y) \leq a^2 \cdot \mathsf{Var}(X) + b^2 \cdot \mathsf{Var}(Y) \leq a^2 \cdot \alpha^2 + b^2 \cdot \beta^2.$$

For $a \in R$ (resp. $\mathbf{x} \in \mathbb{Z}^n$), we denote by $\mathsf{Var}(a)$ (resp. $\mathsf{Var}(\mathbf{x})$) the maximum variance of each coefficient (resp. component) of a (resp. $\mathbf{x}$). The variance of the product of two polynomials $a, b \in R$ is $\mathsf{Var}(a \cdot b) = n \cdot \mathsf{Var}(a) \cdot \mathsf{Var}(b)$. Similarly, $\mathsf{Var}(\mathbf{X})$ denotes the maximum variance of each column of $\mathbf{X}$.

2.2 Fully homomorphic encryption

Roughly speaking, we can divide fully homomorphic encryption (FHE) schemes in two classes: one with large ciphertexts, packing and slow bootstrapping and the other with small ciphertexts, no packing, but fast and programmable bootstrapping. The first family contains schemes such as BGV [6], FV [23], and CKKS [11], while the second one is represented by schemes like FHEW [21], TFHE [12], FHE over the integers [34], and FINAL [4]. In this work, we are only interested in the second type of FHE, thus, in this section we present a general and abstract definition of an FHE scheme that can be instantiated with any of those schemes.

There are three types of ciphertexts:

- *Integer ciphertext*, which is defined over the set $\mathbb{Z}_q$ for some $q \in \mathbb{N}$. We denote by $\mathsf{IntCtxt}_{\mathbf{z}}(\lfloor q/p \rceil \cdot m, E)$ the set of integer ciphertexts encrypting $m \in \mathbb{Z}_p$, under key $\mathbf{z}$, and with E-subgaussian noise. They are the output format of our transciphering and the input format of the subsequent homomorphic computation.
- *Ring ciphertext*, which is defined as a single element or a pair of elements of $\mathcal{R}_Q := \mathbb{Z}_Q[X]/\langle X^N + 1\rangle$ for some $Q, N \in \mathbb{N}$, with N as a power of two. We denote by $\mathsf{RingCtxt}_s(\lfloor Q/p \rceil \cdot m, E)$ the set of ring ciphertexts encrypting $m \in \mathcal{R}_p$, under key $s \in \mathcal{R}$, and with E-subgaussian noise.
- *Gadget ciphertext*, which is defined as a vector or a matrix with entries in $\mathcal{R}_Q$. We denote by $\mathsf{GadgetCtxt}_s^{Q,\ell}(m, E)$ the set of gadget ciphertexts encrypting $m \in \mathcal{R}$, under key $s \in \mathcal{R}$, and with E-subgaussian noise.

Note that we omit the noise parameter when we define any ciphertext if it is not necessary in the context.

Given the security parameter λ, we typically have $q, Q \in \tilde{O}(\lambda^{1.5})$ and $N \in O(\lambda)$. We assume that all ciphertexts carry an estimation of their current noise, which increases as we operate homomorphically with them.

This abstract scheme can then be defined by the following algorithms:

- $\mathsf{FHE.ParamGen}(1^\lambda, p)$: generate parameters params that achieve λ bits of security and allow us to work with plaintext space $\mathbb{Z}_p$. The parameters also include the ring $\mathcal{R}$ an integer $\mathtt{B}_g$, called the decomposition base, and $\ell := \lceil \log Q \rceil$, which defines the dimension of the gadget ciphertexts.
- $\mathsf{FHE.KeyGen}(\mathsf{params})$: generate the secret key $\mathsf{sk} := (\mathbf{z}, s)$, a key-switching key ksk from $\mathbf{s}$ to $\mathbf{z}$, where $\mathbf{s}$ is the vector of coefficients of s, and the bootstrapping key bk.
- $\mathsf{FHE.EncInt}(\mathbf{z}, m)$: using params, output $c \in \mathsf{IntCtxt}_{\mathbf{z}}(\lfloor q/p \rceil \cdot m, E_{in})$ for some $E_{in} = O(q/(2p))$.
- $\mathsf{FHE.DecInt}(\mathbf{z}, \mathbf{c})$: output the message $m \in \mathbb{Z}_p$ encrypted by $\mathbf{c}$ under the secret key $\mathbf{z}$.
- $\mathsf{FHE.EncRing}(s, m)$: using params, output $\mathbf{c} \in \mathsf{RingCtxt}_s(\lfloor Q/p \rceil \cdot m, E_{in})$ for some $E_{in} = O(q/(2p))$.
- $\mathsf{FHE.EncGadget}(s, m)$: using params, output $\mathbf{C} \in \mathsf{GadgetCtxt}_s^{Q,\ell}(m, E_{in})$ for some $E_{in} = O(q/(2p))$.
- Trivial-noiseless ciphertext: any FHE ciphertext defined above where all randomness and the noise are set to 0. We call it a trivial-noiseless ciphertext in this paper.
- $\mathsf{FHE.Add}$: homomorphically add two ciphertexts of the same type, e.g., maps $\mathsf{RingCtxt}_s(\lfloor Q/p \rceil \cdot m_0, E_0) \times \mathsf{RingCtxt}_s(\lfloor Q/p \rceil \cdot m_1, E_1)$ to $\mathsf{RingCtxt}_s\left(\lfloor Q/p \rceil \cdot (m_0 + m_1), \sqrt{E_0^2 + E_1^2}\right)$.
- $\mathsf{FHE.AddPlaintext}$: given a ciphertext of any type, encrypting some message m_0, and a plaintext m_1, this operation outputs a ciphertext of the same type encrypting $m_0 + m_1$. The noise is unchanged, i.e., both input and output have the same noise.
- $\mathsf{FHE.MultPtxt}$: given a message $m_0 \in \mathcal{R}_p$ and a ciphertext $\mathbf{c}_1 \in \mathsf{RingCtxt}_s(\lfloor Q/p \rceil \cdot m_1, E_1)$, outputs $\mathbf{c} \in \mathsf{RingCtxt}_s(\lfloor Q/p \rceil \cdot m_0 \cdot m_1, E)$. If instead of a ring ciphertext, we have $\mathbf{C}_1 \in \mathsf{GadgetCtxt}_s^{Q,\ell}(m_1, E_1)$, it outputs $\mathbf{C} \in \mathsf{GadgetCtxt}_s^{Q,\ell}(m_0 \cdot m_1, E)$. In both cases, $E = \|m_0\|_2 \cdot E_1$.
- $\mathsf{FHE.ExtProd}$: given ciphertexts $\mathbf{c}_0 \in \mathsf{RingCtxt}_s(\lfloor Q/p \rceil \cdot m_0, E_0)$ and $\mathbf{C}_1 \in \mathsf{GadgetCtxt}_s^{Q,\ell}(m_1, E_1)$, it outputs $\mathbf{c} \in \mathsf{RingCtxt}_s(\lfloor Q/p \rceil \cdot m_0 \cdot m_1, E)$ where $E \leq \sqrt{\ell N \cdot \mathtt{B}_g^2 \cdot E_1^2 + \|m_1\|_2^2 \cdot E_0^2}$, where $\mathtt{B}_g$ is the decomposition base. For succinctness, we can write $\mathbf{c}_0 \boxdot_{i=1}^{k} \mathbf{C}_i$ to denote $\mathsf{FHE.ExtProd}(...(\mathsf{FHE.ExtProd}(\mathbf{c}_0, \mathbf{C}_1), \mathbf{C}_2), ..., \mathbf{C}_k)$. In this case, assuming that $\|m_i\|_2 = 1$ for $1 \leq i \leq k$, the resulting ciphertext has E-gaussian noise with $E \leq \sqrt{\sum_{i=1}^{k} \ell \cdot N \cdot \mathtt{B}_g^2 \cdot E_i^2 + E_0^2}$,
- $\mathsf{FHE.Extract}$: Given $\mathbf{c} \in \mathsf{RingCtxt}_s(\lfloor Q/p \rceil \cdot m, E)$ and $i \in [\![0, N-1]\!]$, output $c \in \mathsf{IntCtxt}_{\mathbf{s}}(\lfloor Q/p \rceil \cdot m_i, E)$, where m_i is the i-th coefficient of m. We note that this algorithm is defined as $\mathsf{SampleExtract}$ in [13] and it does not add any noise to the ciphertext. Moreover, it is almost for free in practice since it only rearranges the order of components of input vector/polynomial, which is by far much cheaper than the other operations.
- $\mathsf{FHE.ModSwt}$: Given $\hat{c} \in \mathsf{IntCtxt}_{\mathbf{s}}(\lfloor Q/p \rceil \cdot m, \hat{E})$ and $q \in \mathbb{N}$, output $c \in \mathsf{IntCtxt}_{\mathbf{s}}(\lfloor q/p \rceil \cdot m, E)$, with $E \leq \sqrt{(\hat{E} \cdot (q/Q))^2 + (\|\mathbf{s}\|_2 / 2)^2}$.
- $\mathsf{FHE.KeySwt}$: Given $\hat{c} \in \mathsf{IntCtxt}_{\mathbf{s}}(\lfloor q/p \rceil \cdot m, \hat{E})$, and a key-switching key ksk from $\mathbf{s} \in \mathbb{Z}^N$ to $\mathbf{z} \in \mathbb{Z}^n$, output $c \in \mathsf{IntCtxt}_{\mathbf{z}}(\lfloor q/p \rceil \cdot m, E)$, with $E \leq \sqrt{\hat{E}^2 + N \cdot \log_{\mathtt{B}_{\mathsf{ksk}}} q \cdot \mathtt{B}_{\mathsf{ksk}}^2 \cdot E_k^2}$, where $\mathtt{B}_{\mathsf{ksk}}$ is the decomposition base used during the key-switching.
- $\mathsf{FHE.bootstrap}$: Given $c' \in \mathsf{IntCtxt}_{\mathbf{z}}(\lfloor q/p \rceil \cdot m, E')$, and a function $f : \mathbb{Z}_p \to \mathbb{Z}_{\hat{p}}$, output, $c \in \mathsf{IntCtxt}_{\mathbf{z}}(\lfloor \hat{q}/\hat{p} \rceil \cdot f(m), E_{in})$ where $E_{in} < E'$. Notice that the bootstrapping allow us to change

the ciphertext modulus from q to $\hat{q}$ and the plaintext modulus from p to $\hat{p}$, but in most of the cases, one chooses $q = \hat{q}$ and $p = \hat{p}$.

To analyze the noise growth of a sequence of homomorphic operations, we can iteratively apply the noise bounds of each operations. For example, to homomorphically compute $3 \cdot (m_0 + m_1)$, we could have $\mathbf{c}' = \mathsf{FHE.Add}(\mathbf{c}_0, \mathbf{c}_1)$ and the final ciphertext as $\mathbf{c} = \mathsf{FHE.MultPtxt}(\mathbf{c}_2, \mathbf{C}_1)$. Then, assuming that $\mathbf{c}_i$ has noise with parameter E_i, the noise of $\mathbf{c}'$ would have parameter $\bar{E} = \sqrt{E_0^2 + E_1^2}$, and the final noise would be E-subgaussian where $E = \|3\|_2 \cdot \bar{E} = 3 \cdot \sqrt{E_0^2 + E_1^2}$. Most of the time, deriving the noise like the above is good enough, however, there is a special case, that will be used to construct our homomorphic XOR gate modulo p presented in Section 3.1, where we can have better bounds by analyzing the final noise more carefully (See Lemma 1). Notice that the final noise in the lemma just has E_0 itself, while a naive computation would give us noise including $2 \cdot E_0$. This would be problematic because applying this homomorphic computation iteratively k times would introduce a factor of 2^k to the noise, thus, the estimation would be far from the actual noise.

Lemma 1. *Let $\Delta = \lfloor Q/p \rceil$, $\mathbf{c}_0 \in \mathsf{RingCtxt}_s(\Delta \cdot m_0, E_0)$, and $\mathbf{C}_1 \in \mathsf{GadgetCtxt}_s^{Q,\ell}(m_1, E_1)$, with $m_1 \in \{0, 1\}$. Now, consider the following homomorphic computation:*

1. $\mathbf{c}' = \mathsf{FHE.MultPtxt}(\mathbf{c}_0, -2)$
2. $\hat{\mathbf{c}} = \mathsf{FHE.ExtProd}(\mathbf{c}', \mathbf{C}_1)$
3. $\tilde{\mathbf{c}} = \mathsf{FHE.Add}(\mathbf{c}_0, \hat{\mathbf{c}})$.

Then, it holds that $\tilde{\mathbf{c}} \in \mathsf{RingCtxt}_s(\Delta \cdot (m_0 - 2 \cdot m_0 \cdot m_1), E)$ and

$$E \leq \sqrt{\ell N \cdot \mathtt{B}_g^2 \cdot E_1^2 + E_0^2}$$

Proof. For any ciphertext c, denote by $\mathsf{Err}(c)$ the noise term included in c. Let $\mathbf{y}$ be the vector with the decomposition in base $\mathtt{B}_g$ of $\mathbf{c}'$. Then, we have

$$\begin{aligned}
\mathsf{Err}(\tilde{\mathbf{c}}) &= \mathsf{Err}(\mathbf{c}_0) + \mathsf{Err}(\hat{\mathbf{c}}) \\
&= \mathsf{Err}(\mathbf{c}_0) + \mathbf{y} \cdot \mathsf{Err}(\mathbf{C}_1) + m_1 \cdot \mathsf{Err}(\mathbf{c}') \\
&= \mathsf{Err}(\mathbf{c}_0) + \mathbf{y} \cdot \mathsf{Err}(\mathbf{C}_1) - 2 \cdot m_1 \cdot \mathsf{Err}(\mathbf{c}_0) \\
&= \mathbf{y} \cdot \mathsf{Err}(\mathbf{C}_1) \pm \mathsf{Err}(\mathbf{c}_0)
\end{aligned}$$

Thus, assuming that $\mathsf{Err}(\mathbf{C}_1)$ and $\mathsf{Err}(\mathbf{c}_0)$ are independent, we have that $\mathbf{y} \cdot \mathsf{Err}(\mathbf{C}_1)$ is $(\sqrt{\ell N} \cdot \mathtt{B}_g \cdot E_1)$-subgaussian, then, by Pythagorean inequality for subgaussians, it gives us $E \leq \sqrt{\ell N \cdot \mathtt{B}_g^2 \cdot E_1^2 + E_0^2}$.

The correctness of the encrypted message follows directly from the definition of the homomorphic operations. □

Corollary 1. *Instead of the homomorphic computation presented in Lemma 1, if we compute*

$$\tilde{\mathbf{c}} = \mathbf{c}_0 + (m_2 - 2 \cdot \mathbf{c}_0) \boxdot \mathbf{C}_1$$

for any plaintext m_2, then E is still bounded as $E \leq \sqrt{\ell N \cdot \mathtt{B}_g^2 \cdot E_1^2 + E_0^2}$.

Proof. This follows directly from the fact that $\mathsf{FHE.AddPlaintext}$ does not add any noise to the ciphertexts. □

2.3 FiLIP cipher

FiLIP is a binary stream cipher based on the improved filter permutator paradigm [31]. The encryption and decryption algorithms work as follows: Let $K \in \{0,1\}^Z$ be the secret key; for each bit m_i of the message, we use a forward secure PRNG to sample

- S_i: a subset of z out of Z,
- P_i: a z to z permutation,
- $\mathbf{w}_i$: an z-dimensional binary vector called *whitening*.

Then, for a filter function $f : \{0,1\}^z \to \{0,1\}$ fixed beforehand, we compute $c_i := m_i \oplus f(P_i(S_i(K)) \oplus \mathbf{w}_i) \in \{0,1\}$. The paradigm of FiLIP is recalled in Figure 2.

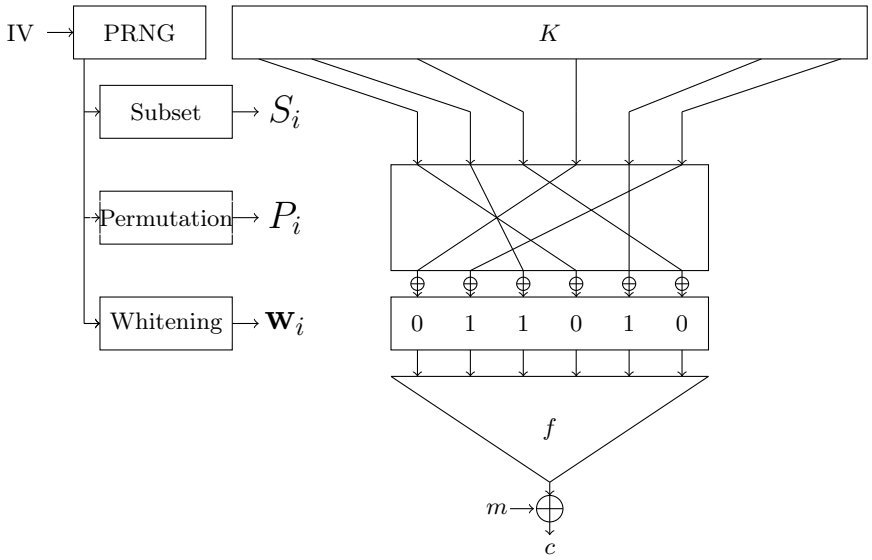

Fig. 2: *FiLIP's paradigm.*

We implemented the variant called FiLIP-144 in [28], which consists in setting $Z = 2^{14}, z = 144$ and f as the XOR-THR function $\mathsf{XTHR}_{[81,32,63]}$ described in Definition 2. We note that those parameters of FiLIP-144 yield 128 bit security, following the analysis in [31]. The cryptographic parameters of XOR-THR functions are studied in details in [8].

Definition 2 (Threshold Function). *Let $s \in \mathbb{N}^*$. For any positive integer $d \leq s+1$, the Boolean function $\mathbf{T}_{d,s}$ is defined as:*

$$\forall x = (x_1, \ldots, x_s) \in \mathbb{F}_2^s, \mathbf{T}_{d,s}(x) = \begin{cases} 1 & \text{if } \mathsf{W_H}(x) \geq d, \\ 0, & \text{otherwise} \end{cases}$$

where $\mathsf{W_H}(x)$ is the Hamming weight of a binary vector x.

Definition 3 (XOR-THR Function (e.g. [28], Definition 11)). *For any positive integers k, d, and s such that $d \leq s+1$, and for all $z = (x_1, \ldots, x_k, y_1, \ldots, y_s) \in \mathbb{F}_2^{k+s}$, $\mathsf{XTHR}_{[k,d,s]}$ is defined as:*

$$\mathsf{XTHR}_{[k,d,s]}(z) = \mathsf{XOR}_k(x) + \mathbf{T}_{d,s}(y) \in \mathbb{F}_2,$$

where $\mathsf{XOR}_k(x) = x_1 + \cdots + x_k$.

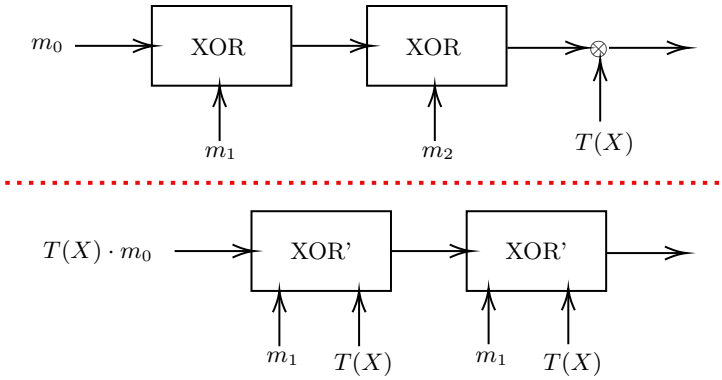

Fig. 3: *Two strategies to compute a sequence of XOR gates multiplied by a polynomial. In both cases, the output is* $T(X) \cdot \mathsf{XOR}(m_2, \mathsf{XOR}(m_1, m_0))$*, but the second computation inserts* $T(X)$ *right in the beginning and carries it until the end, reducing thus the final noise when evaluated homomorphically.*

3 Ad hoc homomorphic building blocks

In this section we define new homomorphic operations that will be used in our transciphering. They are constructed using the operations defined in Section 2.2, thus, they can also be instantiated with any FHEW-like scheme.

3.1 Homomorphic XOR modulo p

We start with the simplest scenario where the ciphertexts only encrypt bits, but using $\mathbb{Z}_p$ as the message space. Given $m_0, m_1 \in \{0, 1\}$, we can see that

$$\mathsf{XOR}(m_0, m_1) = m_0 + m_1 - 2 \cdot m_0 \cdot m_1 \pmod{p}$$

Thus, we can easily compute an encryption of $\mathsf{XOR}(m_0, m_1)$ given encryptions of m_0 and m_1, as we show in Appendix A. And, in fact, one could implement FiLIP modulo p using such simple homomorphic XOR, however, at the very end of the main loop, after all the external products, one would obtain an encryption of a power of X and would have to multiply it by the test polynomial $T(X)$[5] to extract $\mathsf{XTHR}_{[k,d,s]}$. But, multiplying by $T(X)$ introduces an extra factor of $\sqrt{N}$ in the final noise. Thus, as it was done originally in the bootstrapping of TFHE [12], we would like to start the loop with $T(X)$ already, so that it is always multiplied on the left and does not impact the noise.

For this, we introduce another homomorphic XOR gate modulo p that outputs an encryption of $u \cdot \mathsf{XOR}(m_0, m_1)$ for any polynomial u, so that we can carry the test polynomial $T(X)$ from the beginning of the computation and we do not need to multiply it at the end, thus, reducing the final noise. This is illustrated in Figure 3.

In more detail, let $m_0, m_1 \in \{0, 1\}$ and u be a polynomial, let $\mathbf{c}_0 \in \mathsf{RingCtxt}_s(\Delta \cdot u \cdot m_0, E_0)$, $\mathbf{C}_1 \in \mathsf{GadgetCtxt}_s^{Q,\ell}(m_1, E_1)$, and $\mathbf{c}_2 \in \mathsf{RingCtxt}_s(\Delta \cdot u \cdot m_1, E_2)$. We define this gate as follows

$$\mathsf{FHE.XOR}(\mathbf{c}_0, \mathbf{C}_1, \mathbf{c}_2) := \mathbf{c}_0 + \mathbf{c}_2 - (2 \cdot \mathbf{c}_0) \boxdot \mathbf{C}_1.$$

[5] We use two terms, a test polynomial and a test vector, to refer to $T(x)$ interchangeably.

Algorithm 1: FHE.XOR

Input: $\mathbf{c}_0 \in \mathsf{RingCtxt}_s(\Delta \cdot u \cdot m_0, E_0)$, $\mathbf{C}_1 \in \mathsf{GadgetCtxt}_s^{Q,\ell}(m_1, E_1)$, and $\mathbf{c}_2 \in \mathsf{RingCtxt}_s(\Delta \cdot u \cdot m_1, E_2)$, where $u \in \mathcal{R}_t$, $m_0, m_1 \in \{0,1\}$, and $\Delta = \lfloor Q/p \rceil$.

Output: $\mathbf{c} \in \mathsf{RingCtxt}_s(\Delta \cdot u \cdot \mathsf{XOR}(m_0, m_1), E_{out})$

Noise growth: $E_{out} \leq \sqrt{\ell N \cdot \mathtt{B}_g^2 \cdot E_1^2 + E_0^2 + E_2^2}$

$\mathbf{c}' = \mathsf{FHE.MultPtxt}(-2, \mathbf{c}_0)$; ▷ $\mathsf{RingCtxt}_s(-\Delta \cdot 2 \cdot u \cdot m_0)$

$\mathbf{c}'' = \mathsf{FHE.ExtProd}(\mathbf{c}', \mathbf{C}_1)$; ▷ $\mathsf{RingCtxt}_s(-\Delta \cdot 2 \cdot u \cdot m_0 \cdot m_1)$

$\hat{\mathbf{c}} = \mathsf{FHE.Add}(\mathbf{c}'', \mathbf{c}_0)$; ▷ $\mathsf{RingCtxt}_s(\Delta u \cdot m_0(1 - 2m_1), \hat{E})$

$\mathbf{c} = \mathsf{FHE.Add}(\mathbf{c}_2, \hat{\mathbf{c}})$; ▷ $\mathsf{RingCtxt}_s(\Delta \cdot u \cdot \mathsf{XOR}(m_0, m_1)), E_{out})$

return $\mathbf{c}$

We show it in thoroughly in Algorithm 1. From Lemma 1, it holds that $\hat{E} \leq \sqrt{\ell N \cdot \mathtt{B}_g^2 \cdot E_1^2 + E_0^2}$. Then, by the properties of $\mathsf{FHE.Add}$, we have $E_{out} \leq \sqrt{\hat{E}^2 + E_2^2} \leq \sqrt{\ell N \cdot \mathtt{B}_g^2 \cdot E_1^2 + E_0^2 + E_2^2}$.

Moreover, one can see that the output of $\mathsf{FHE.XOR}$ is composable as

$$\mathsf{FHE.XOR}(\mathsf{FHE.XOR}(\mathbf{c}_0, \mathbf{C}_1, \mathbf{c}_1), \mathbf{C}_2, \mathbf{c}_2)).$$

Thus, if we execute k consecutive compositions of this gate with $\mathbf{c}_i \in \mathsf{RingCtxt}_s(\Delta \cdot u \cdot m_i, E_i)$ for $0 \leq i \leq k$ and $\mathbf{C}_j \in \mathsf{GadgetCtxt}_s^{Q,\ell}(m_j, \hat{E}_j)$ for $1 \leq j \leq k$, we obtain an encryption $\mathsf{XOR}_k(m_0, m_1, \ldots, m_k)$. Additionally, we can verify that the final noise is E-subgaussian with

$$E \leq \sqrt{\sum_{i=1}^{k} \ell N \cdot \mathtt{B}_g^2 \cdot \hat{E}_i^2 + \sum_{i=0}^{k} E_i^2}. \tag{1}$$

Homomorphically lifting a bit to the exponent Let u be a polynomial and $b \in \{0,1\}$. This homomorphic operation takes an encryption of $u \cdot b$ and outputs an encryption of $u \cdot X^b$. Suppose we have $\mathbf{c} \in \mathsf{RingCtxt}_s(\Delta \cdot u \cdot b, E)$, we just compute the following

$$\mathsf{FHE.LiftExp}(\mathbf{c}, u) := (X - 1) \cdot \mathbf{c} + u.$$

The correctness and noise growth follow directly from the properties of the plaintext-ciphertext addition and multiplication. We show it in detail in Algorithm 2.

Algorithm 2: FHE.LiftExp

Input: $\mathbf{c} \in \mathsf{RingCtxt}_s(\Delta \cdot u \cdot b, E)$, where $u \in \mathcal{R}_p$, $b \in \{0,1\}$, and $\Delta = \lfloor Q/p \rceil$.

Output: $\hat{\mathbf{c}} \in \mathsf{RingCtxt}_s(\Delta \cdot u \cdot X^b, E_{out})$

Noise growth: $E_{out} = 2 \cdot E$

$\mathbf{c}' = \mathsf{FHE.MultPtxt}(X - 1, \mathbf{c})$; ▷ $\mathsf{RingCtxt}_s(\Delta \cdot (ubX - ub), 2 \cdot E)$

$\hat{\mathbf{c}} = \mathsf{FHE.AddPlaintext}(u, \mathbf{c}')$; ▷ $\mathsf{RingCtxt}_s(\Delta \cdot (ubX + u(1 - b)), 2 \cdot E)$

return $\hat{\mathbf{c}}$

4 Transciphering for $\mathbb{Z}_p$ from transciphering for $\{0,1\}$

4.1 Setup for homomorphic FiLIP

This phase starts with the client generating the secret keys for FiLIP and for the FHE scheme, then encrypting FiLIP's key under the FHE key and sending it to the server. This is called client's setup and it is shown in Algorithm 3, where we assume that the noise of fresh ciphertexts is sampled from a σ-subgaussian distribution.

Algorithm 3: ClientSetup

Input: FHE's secret key s, FiLIP's secret key $\mathbf{k} = (k_0, ..., k_{Z-1}) \in \{0,1\}^Z$
Output: $\mathbf{C}_i \in \mathsf{GadgetCtxt}_s^{Q,\ell}(k_i, \sigma)$

for $0 \leq i < Z$ **do**
  $\mathbf{C}_i = \mathsf{FHE.EncGadget}(s, k_i)$
return $(\mathbf{C}_0, ..., \mathbf{C}_{Z-1})$

Then, the server expands the FHE encryptions by running a global setup which is independent of the FHE plaintext space p. This is shown thoroughly in Algorithm 4.

Algorithm 4: GlobalSetup

Input: For $0 \leq i < Z$, $\mathbf{C}_i \in \mathsf{GadgetCtxt}_s^{Q,\ell}(k_i, \sigma)$
Output: Z triples of gadget ciphertexts $(\bar{\mathbf{C}}_i, \hat{\mathbf{C}}_i, \tilde{\mathbf{C}}_i)$

for $0 \leq i < Z$ **do**
  $\bar{\mathbf{C}}_i = \mathsf{FHE.Add}(1, -\mathbf{C}_i)$; ▷ $\mathsf{GadgetCtxt}_s^{Q,\ell}(\mathsf{NOT}(k_i), \sigma)$
  $\hat{\mathbf{C}}_i = \mathsf{FHE.Add}(1, (X^2 - 1) \cdot \mathbf{C}_i)$; ▷ $\mathsf{GadgetCtxt}_s^{Q,\ell}(X^{2 \cdot k_i}, 2 \cdot \sigma)$
  $\tilde{\mathbf{C}}_i = \mathsf{FHE.Add}(1, (X^2 - 1) \cdot \bar{\mathbf{C}}_i)$; ▷ $\mathsf{GadgetCtxt}_s^{Q,\ell}(X^{2 \cdot \mathsf{NOT}(k_i)}, 2 \cdot \sigma)$
return $(\bar{\mathbf{C}}_i, \hat{\mathbf{C}}_i, \tilde{\mathbf{C}}_i)_{i=0}^{Z-1}$

Moreover, for any given p, the server also has to run, only once, a setup step. We call this $\mathsf{p{-}Setup}$ and show it in detail in Algorithm 5. It depends on the following function F_d, which is used to map a value of the form $b + 2 \cdot w$ to $b + y \bmod 2$, where $y = 1$ if $w \geq d$ and $y = 0$ otherwise. That is, we define

$$F_d(u) := \begin{cases} u + 1 \bmod 2 & \text{if } \lfloor u/2 \rfloor \geq d \\ u \bmod 2 & \text{otherwise} \end{cases} \tag{2}$$

In the online phase, we will compute b as the XOR of some bits of FiLIP's secret key and w as the Hamming weight of some other bits, then $F_d(b + 2 \cdot w)$ is applied to those bits. After that, the server is ready to apply the transciphering as many times as needed to transform FiLIP's ciphertexts into FHE ciphertexts with $\mathbb{Z}_p$ as the plaintext space.

4.2 Online phase

In this step, the server transforms groups of $L := \lceil \log p \rceil$ FiLIP's ciphertexts into integer ciphertexts (e.g., LWE ciphertexts) encrypting integers modulo p. Thus, consider FiLIP's ciphertexts

Algorithm 5: p−Setup

Input: $p \in \mathbb{N}$, the function F_d defined in Equation (5), and for $0 \leq i < Z$, $\mathbf{C}_i \in \mathsf{GadgetCtxt}_s^{Q,\ell}(k_i, \sigma)$ and $\bar{\mathbf{C}}_i \in \mathsf{GadgetCtxt}_s^{Q,\ell}(\mathsf{NOT}(k_i), \sigma)$

Output: $(\mathbf{c}_{i,j}, \bar{\mathbf{c}}_{i,j})$ for $0 \leq i < Z$ and $0 \leq j < \lceil \log_2(p) \rceil$.

Noise growth: $E_{out} = \sqrt{\ell N} \cdot \mathtt{B}_g \cdot \sigma$

$\Delta := \lfloor Q/p \rceil$

$T(X) := \sum_{i=0}^{N-1} F_d(i) \cdot X^{2N-i} \bmod X^N + 1$

for $0 \leq j < \lceil \log_2(p) \rceil$ **do**

  $u_j := 2^j \cdot T(X)$

  $\mathbf{c}^{(j)} :=$ trivial-noiseless ring encryption of $\Delta \cdot u_j$

  for $0 \leq i < Z$ **do**

    $\mathbf{c}_{i,j} = \mathsf{FHE.ExtProd}(\mathbf{c}^{(j)}, \mathbf{C}_i)$; ▷ $\mathsf{RingCtxt}_s(\Delta \cdot u_j \cdot k_i, E_{out})$

    $\bar{\mathbf{c}}_{i,j} = \mathsf{FHE.ExtProd}(\mathbf{c}^{(j)}, \bar{\mathbf{C}}_i)$; ▷ $\mathsf{RingCtxt}_s(\Delta \cdot u_j \cdot \mathsf{NOT}(k_i), E_{out})$

return $(\mathbf{c}_{i,j}, \bar{\mathbf{c}}_{i,j})$

$c_0, ..., c_{L-1} \in \{0,1\}$ encrypting bits $b_0, ..., b_{L-1}$, respectively. It holds that $c_j = b_j + \mathbf{F}(\mathbf{k}, \mathsf{IV}_j) \bmod 2$, where $\mathbf{k}$ is FiLIP's secret key and $\mathbf{F}$ is FiLIP encryption function, as explained in Section 2.3. Our goal is to describe a method to output $c \in \mathsf{IntCtxt}_{\mathbf{z}}(\lfloor q/p \rceil \cdot m, E)$ where $m = \sum_{j=0}^{L-1} b_j \cdot 2^j$ and $E = O(q/(2p))$, allowing thus any subsequent homomorphic computation via programmable bootstrapping.

To do so, we proceed by generating ciphertexts $\mathbf{c}^{(j)} \in \mathsf{RingCtxt}_s(\lfloor Q/p \rceil \cdot \mu_j, E/\sqrt{p})$, such that the first coefficient of μ_j is equal to $2^j \cdot \mathbf{F}(\mathbf{k}, \mathsf{IV}_j)$.

Then, by using FiLIP's ciphertexts c_j's, we can generate encryptions of $2^j \cdot b_j$ and add them together to obtain an encryption of m, as desired. Notice that each $\mathbf{c}^{(j)}$ can be computed in parallel. This first step is described in Algorithm 6 and it is similar to the homomorphic evaluation of FiLIP presented in [17], but the XOR is not longer computed with homomorphic additions and the whole computation carries the power of two and the test vector $T(X)$. In Lemma 2, we prove the correctness of Algorithm 6 and analyze the noise of its output.

Lemma 2. *[Correctness and noise analysis of* BinaryTranscipher*] Let* $\mathbf{k} = (k_0, ..., k_{Z-1}) \in \{0,1\}^Z$ *be the secret key of FiLIP. Fix integers* j *and* IV*. Let* $\mathbf{F}$ *be FiLIP encryption function, i.e., FiLIP's ciphertexts are of the form* $c = b + \mathbf{F}(\mathbf{k}, \mathsf{IV}) \bmod 2$*. Let* $u_j := 2^j \cdot T(X)$*, where* $T(X)$ *is the test vector defined in* p−Setup*. Let* $\Delta := \lfloor Q/p \rceil$*. For* $i \in [\![0, Z-1]\!]$*, consider the following input ciphertexts:*

- *Generated by* ClientSetup
  - $\mathbf{C}_i \in \mathsf{GadgetCtxt}_s^{Q,\ell}(k_i, \sigma)$
- *Generated by* GlobalSetup
  - $\bar{\mathbf{C}}_i \in \mathsf{GadgetCtxt}_s^{Q,\ell}(\mathsf{NOT}(k_i), \sigma)$
  - $\hat{\mathbf{C}}_i \in \mathsf{GadgetCtxt}_s^{Q,\ell}(X^{2 \cdot k_i}, 2 \cdot \sigma)$
  - $\tilde{\mathbf{C}}_i = \mathsf{GadgetCtxt}_s^{Q,\ell}(X^{2 \cdot \mathsf{NOT}(k_i)}, 2 \cdot \sigma)$
- *Generated by* p−Setup
  - $\mathbf{c}_{i,j} \in \mathsf{RingCtxt}_s(\Delta \cdot u_j \cdot k_i, \sqrt{\ell N} \cdot \mathtt{B}_g \cdot \sigma)$
  - $\bar{\mathbf{c}}_{i,j} \in \mathsf{RingCtxt}_s(\Delta \cdot u_j \cdot \mathsf{NOT}(k_i), \sqrt{\ell N} \cdot \mathtt{B}_g \cdot \sigma)$

Algorithm 6: BinaryTranscipher

Input: An integer IV, an integer j, and, for $i \in [\![0, Z-1]\!]$, the ciphertexts $\mathbf{c}_{i,j}$ and $\bar{\mathbf{c}}_{i,j}$ computed in p−Setup, $\bar{\mathbf{C}}_i, \hat{\mathbf{C}}_i$, and $\tilde{\mathbf{C}}_i$ computed in GlobalSetup, and $\mathbf{C}_i$ generated by ClientSetup.

Output: $\mathbf{c} \in \mathsf{RingCtxt}_s(\lfloor Q/p \rceil \cdot \mu, E_{out})$ where $\mu \in \mathcal{R}_t$ with $\mu_0 = 2^j \cdot F(\mathbf{k}, \mathsf{IV})$, $\mathbf{k}$ is FiLIP's secret key, and F FiLIP's encryption.

Noise growth: $E_{out} \leq 15\sqrt{\ell N} \cdot \mathtt{B}_g \cdot \sigma$

Sample the subset $S := \{s_1, ..., s_z\} \subseteq \{1, ..., Z\}$

Sample the permutation $P : S \to S$.

Sample thewhitening vector $\mathbf{w} \in \{0,1\}^z$

for $0 \leq i < 144$ **do**

  $r \leftarrow P[s_i]$ **if** $w_i = 0$ **then**

    ▷ `Select encryptions of` $2^j \cdot T(X) \cdot k_r$, k_r, `and` $X^{2 \cdot k_r}$

    $\mathbf{c}^{(i)} := \mathbf{c}_{r,j}$, $\mathbf{C}^{(i)} := \mathbf{C}_r$, $\hat{\mathbf{C}}^{(i)} := \hat{\mathbf{C}}_r$,

  else

    ▷ `Select` $2^j \cdot T(X) \cdot \mathsf{NOT}(k_r)$, $\mathsf{NOT}(k_r)$, `and` $X^{2 \cdot NOT(k_r)}$

    $\mathbf{c}^{(i)} := \bar{\mathbf{c}}_{r,j}$, $\mathbf{C}^{(i)} := \bar{\mathbf{C}}_r$, $\hat{\mathbf{C}}^{(i)} := \tilde{\mathbf{C}}_r$,

▷ `Now compute` $\mathsf{XOR}(k'_1, \ldots, k'_{80})$ `where` k'_i `are the permuted and whitened bits of FiLIP's secret key`

$\mathbf{c}_{\mathsf{XOR}} = \mathbf{c}^{(0)}$

for $1 \leq i < 81$ **do**

  $\mathbf{c}_{\mathsf{XOR}} = \mathsf{FHE.XOR}(\mathbf{c}_{\mathsf{XOR}}, \mathbf{C}^{(i)}, \mathbf{c}^{(i)})$

$u_j := 2^j \cdot T(X)$; ▷ `Scaled test vector as in p−Setup`

$\mathbf{c} = \mathsf{FHE.LiftExp}(\mathbf{c}_{\mathsf{XOR}}, u_j)$; ▷ $\mathsf{RingCtxt}_s(\Delta \cdot u_j \cdot X^{\mathsf{XOR}(k'_0, \ldots, k'_{80})}, E_{\mathsf{XOR}})$

▷ `Now accumulate` $2 \cdot \mathsf{HW}(k'_{81}, \ldots, k'_{143})$ `in the exponent`

for $81 \leq i < 144$ **do**

  $\mathbf{c} = \mathsf{FHE.ExtProd}(\mathbf{c}, \hat{\mathbf{C}}^{(i)})$

return $\mathbf{c}$; ▷ $\mathsf{RingCtxt}_s(\Delta \cdot u_j \cdot X^{\mathsf{XOR}(k'_0, \ldots, k'_{80}) + 2 \cdot \mathsf{HW}(k'_{81}, \ldots, k'_{143})}, E_{out})$

Then, if $\mathbf{c}$ *is the output of* BinaryTranscipher*, it holds that* $\mathbf{c} \in \mathsf{RingCtxt}_s(\lfloor Q/p \rceil \cdot \mu, E_{out})$ *where* $\mu \in \mathcal{R}_t$ *with* $\mu_0 = 2^j \cdot \mathbf{F}(\mathbf{k}, \mathsf{IV})$*, and*

$$E_{out} \leq 15\sqrt{\ell N} \cdot \mathtt{B}_g \cdot \sigma.$$

Proof. Let $k'_0, ..., k'_{143}$ be the bits of FiLIP's secret key after taking the subset and applying the permutation and the whitening. Notice that $\mathbf{F}(\mathbf{k}, \mathsf{IV}) = \mathsf{XOR}(k'_0, ..., k'_{80}) + \mathbf{T}_{32,63}(k'_{81}, ..., k'_{143}) \bmod 2$, as in Definition 3.

Since the whitening corresponds to negating the bit when $w_i = 1$, it holds that at the end of the first loop of BinaryTranscipher, the ciphertexts $\mathbf{c}_{i,j}$, $\mathbf{C}^{(i)}$, and $\hat{\mathbf{C}}^{(i)}$ encrypt $u_j \cdot k'_i$, k'_i, and $X^{2 \cdot k'_i}$, respectively.

Thus, from the correctness of FHE.XOR, at the end of the second for loop, we have $\mathbf{c}_{\mathsf{XOR}} \in \mathsf{RingCtxt}_s(\Delta \cdot u_j \mathsf{XOR}(k'_0, ..., k'_{80}), \hat{E})$, for some $\hat{E}$.

Then, from the correctness of FHE.LiftExp, it holds that $\mathbf{c} \in \mathsf{RingCtxt}_s(\Delta \cdot u_j \cdot X^{\mathsf{XOR}(k'_0, ..., k'_{80})}, E_{\mathsf{XOR}})$, for some E_{XOR}.

Finally, each iteration of the last loop adds $2 \cdot k'_i$ to the exponent of X encrypted in $\mathbf{c}$. But notice that since $k'_i \in \{0,1\}$, it holds that the Hamming weight is equal to the sum, thus, at the end, we have $\mathbf{c} \in \mathsf{RingCtxt}_s(\Delta \cdot u_j \cdot X^{\mathsf{XOR}(k'_0, ..., k'_{80}) + 2 \cdot \mathsf{HW}(k'_{81}, ..., k'_{143})}, E_{out})$, for some E_{out}. Now, recall that the test vector $T(X)$ encodes the function F_d from Equation 5, thus, $T(X) \cdot X^k$ results in a polynomial

μ whose constant term is $m_0 = F_d(k)$. Hence, $\mathbf{c}$ encrypts μ such that

$$\mu_0 = 2^j \cdot F_d(\mathsf{XOR}(k'_0, ..., k'_{80}) + 2 \cdot \mathsf{HW}(k'_{81}, \ldots, k'_{143})) = 2^j \cdot \mathbf{F}(\mathbf{k}, \mathsf{IV})$$

as desired.

Now it remains to analyze the noise. By Inequality 5, it holds that

$$E_{\mathsf{XOR}} \leq \sqrt{80 \cdot \ell N \cdot \mathtt{B}_g^2 \cdot \sigma^2 + 81 \cdot \ell N \cdot \mathtt{B}_g^2 \cdot \sigma^2} = \sqrt{161 \cdot \ell N} \cdot \mathtt{B}_g \cdot \sigma.$$

Finally, the 63 consecutive external products in the last loop give us

$$\begin{aligned} E_{out} &\leq \sqrt{63\ell N \cdot \mathtt{B}_g^2 \cdot \sigma^2 + E_{\mathsf{XOR}}^2} \\ &\leq \sqrt{63\ell N \cdot \mathtt{B}_g^2 \cdot \sigma^2 + (\sqrt{161 \cdot \ell N} \cdot \mathtt{B}_g \cdot \sigma)^2} \\ &\leq \sqrt{63\ell N \cdot \mathtt{B}_g^2 \cdot \sigma^2 + 161 \cdot \ell N \cdot (\mathtt{B}_g \cdot \sigma)^2} \\ &\leq \sqrt{224\ell N \cdot \mathtt{B}_g^2 \cdot \sigma^2} \\ &\leq 15\sqrt{\ell N} \cdot \mathtt{B}_g \cdot \sigma \end{aligned}$$

□

The full transciphering procedure is shown in Algorithm 7 and it works by calling L times $\mathsf{BinaryTranscipher}$, then combing the ciphertexts and finally using key- and modulus-switching procedures to output an integer ciphertext with the right format.

Lemma 3. *[Correctness and noise analysis of $\mathbb{Z}_p\mathsf{Transcipher}$] Consider the same notation and inputs used in Lemma 2. Let $L := \lceil \log p \rceil$. Assume that the key-switching key ksk has σ_{ksk}-subgaussian noise for some σ_{ksk}. For $j \in [\![0, L-1]\!]$, let $c_j = b_j + \mathbf{F}(\mathbf{k}, \mathsf{IV}_j) \bmod 2$ be a FiLIP ciphertext.*

Then, if c is the output of $\mathbb{Z}_p\mathsf{Transcipher}$, it holds that $c \in \mathsf{IntCtxt}_{\mathbf{z}}(\lfloor q/p \rceil \cdot m, E_{out})$ where $m = \sum_{j=0}^{L-1} 2^j \cdot b_j$ and

$$E_{out} \leq \sqrt{N \cdot (15^2 \cdot \log p \cdot \ell \cdot (\mathtt{B}_g \cdot \sigma \cdot q/Q)^2 + \ell_{\mathsf{ksk}} \cdot (\mathtt{B}_{\mathsf{ksk}} \cdot \sigma_{\mathsf{ksk}})^2) + \|\mathbf{s}\|_2^2/4}$$

where $\ell_{\mathsf{ksk}} = \log_{\mathtt{B}_{\mathsf{ksk}}} q$.

Proof. From Lemma 2, we know that the constant term of the message encrypted by $\mathbf{c}^{(j)}$ is equal to $2^j \cdot \mathbf{F}(\mathbf{k}, \mathsf{IV})$. Notice that if $c_j = 0$, then $b_j = \mathbf{F}(\mathbf{k}, \mathsf{IV}) \in \{0, 1\}$, thus, this constant term is already equal to $2^j \cdot b_j$. If $c_j = 1$, then $b_j = 1 - \mathbf{F}(\mathbf{k}, \mathsf{IV}) \in \{0, 1\}$, and line 4 turns the constant term into $2^j - 2^j \cdot \mathbf{F}(\mathbf{k}, \mathsf{IV}) = 2^j \cdot (1 - \mathbf{F}(\mathbf{k}, \mathsf{IV}))$. Therefore, at the end of the for loop, each $\mathbf{c}^{(j)}$ encrypts $2^j \cdot b_j$ in the constant term. It follows that $\mathbf{c}$ encrypts m in the constant term.

Hence, from the correctness of $\mathsf{FHE.Extract}$, $\mathsf{FHE.ModSwt}$, and $\mathsf{FHE.KeySwt}$, $c \in \mathsf{IntCtxt}_{\mathbf{z}}(\lfloor q/p \rceil \cdot m, E_{out})$, for some E_{out}, as desired.

Now it remains to prove the noise bound. Again from Lemma 2 and using the fact that $\mathsf{FHE.AddPlaintext}$ does not change the noise, at the end of the for loop, each $\mathbf{c}^{(j)}$ has E-subgaussian noise with $E \leq 15\sqrt{\ell N} \cdot \mathtt{B}_g \cdot \sigma$.

In line 5, we apply $\log p$ times $\mathsf{FHE.Add}$, thus, the we obtain $(\sqrt{\log p} \cdot E)$-subgaussian noise.

Then, FHE.Extract does not change the noise distribution and FHE.ModSwt gives us $\left(\sqrt{\log p \cdot E^2 \cdot (q/Q)^2 + (\|\mathbf{s}\|_2/2)^2}\right)$-subgaussian noise.

Finally, the after FHE.KeySwt, we have

$$\begin{aligned} E_{out} &\leq \sqrt{\log p \cdot E^2 \cdot (q/Q)^2 + \|\mathbf{s}\|_2^2/4 + N \cdot \log_{\mathsf{B}_{\mathsf{ksk}}} q \cdot (\mathsf{B}_{\mathsf{ksk}} \cdot \sigma_{\mathsf{ksk}})^2} \\ &\leq \sqrt{\log p \cdot (15 \cdot \sqrt{\ell \cdot N} \cdot \mathsf{B}_g \cdot \sigma)^2 \cdot (q/Q)^2 + \|\mathbf{s}\|_2^2/4 + N \cdot \log_{\mathsf{B}_{\mathsf{ksk}}} q \cdot (\mathsf{B}_{\mathsf{ksk}} \cdot \sigma_{\mathsf{ksk}})^2} \\ &\leq \sqrt{15^2 \cdot \log p \cdot \ell \cdot N \cdot (\mathsf{B}_g \cdot \sigma)^2 \cdot (q/Q)^2 + \|\mathbf{s}\|_2^2/4 + N \cdot \log_{\mathsf{B}_{\mathsf{ksk}}} q \cdot (\mathsf{B}_{\mathsf{ksk}} \cdot \sigma_{\mathsf{ksk}})^2} \end{aligned}$$

□

Algorithm 7: $\mathbb{Z}_p$Transcipher

Input: Key-switching key ksk. All the ciphertexts generated by ClientSetup, GlobalSetup, and p−Setup. For $0 \leq j < L := \lceil \log p \rceil$, FiLIP ciphertext $c_j = b_j + \mathbf{F}(\mathbf{k}, \mathsf{IV}_j) \bmod 2$ and the initialization vector IV_j.

Output: $c \in \mathsf{IntCtxt}_{\mathbf{z}}(\lfloor q/p \rceil \cdot m, E_{out})$ where $m = \sum_{j=0}^{L-1} 2^j \cdot b_j$.

Noise growth: $E_{out} \leq \sqrt{N(15^2 \cdot \log p \cdot \ell \cdot (\mathsf{B}_g \sigma \cdot q/Q)^2 + \ell_{\mathsf{ksk}}(\mathsf{B}_{\mathsf{ksk}} \cdot \sigma_{\mathsf{ksk}})^2) + \|\mathbf{s}\|_2^2/4}$

for $0 \leq j < L$ **do**
  $\mathbf{c}^{(j)} = \mathsf{BinaryTranscipher}(\mathsf{IV}_j)$; ▷ `Constant term:` $2^j \cdot \mathbf{F}(\mathbf{k}, \mathsf{IV}_j)$
  if $c_j = 1$ **then**
    $\mathbf{c}^{(j)} = \mathsf{FHE.AddPlaintext}(2^j, -\mathbf{c}^{(j)})$; ▷ `Constant term:` $2^j \cdot b_j$
▷ `Now combine the` L `ciphertexts`
$\mathbf{c} = \sum_{j=0}^{L-1} \mathbf{c}^{(j)}$
$c' = \mathsf{FHE.Extract}(\mathbf{c}, 0)$; ▷ $\mathsf{IntCtxt}_{\mathbf{s}}(\lfloor Q/p \rceil \cdot m)$
$\hat{c} = \mathsf{FHE.ModSwt}(c', q)$; ▷ $\mathsf{IntCtxt}_{\mathbf{s}}(\lfloor q/p \rceil \cdot m)$
$c = \mathsf{FHE.KeySwt}(\hat{c}, \mathsf{ksk})$; ▷ $\mathsf{IntCtxt}_{\mathbf{z}}(\lfloor q/p \rceil \cdot m)$
return c

5 Experimental Results and Comparisons

5.1 Instantiation and implementation

We show implementation results of our approach as a proof of concept. One can use different third-generation FHE schemes to instantiate our transciphering. To obtain our practical results, we used FINAL [4], as it allows us to represent gadget ciphertexts with a vector of $\ell := \lceil \log_{\mathsf{B}_g} Q \rceil$ elements of $\mathcal{R}_Q$, which tends to be smaller than the GSW ciphertexs used by TFHE, that are $2 \times 2\ell'$ matrices (although usually $\ell' < \ell$). Thus, the three ciphertexts types presented in Section 2.2 are shown in Table 1. The extraction procedure generates a vector that is still encrypted under NTRU, thus, FINAL offers a NTRU-to-LWE key switching that we use to obtain the output of the transciphering. When we compare with other algorithms instantiated with TFHE in the later section, the term GSW ciphertext corresponds to the output of FHE.EncGadget, and RLWE ciphertext corresponds to the output of FHE.EncRing.

Table 1: *Actual ciphertext types and parameters when our transciphering is instantiated with FINAL.*

Ciphertext type	Hard problem	Parameters	Ciphertext space
Integer	LWE	$n, q, \sigma_{\mathsf{LWE}}$	$\mathbb{Z}_q^{n+1}$
Ring	NTRU	$N, Q, \sigma_{\mathsf{NTRU}}$	$\mathcal{R}_Q$
Gadget	NTRU	$N, Q, \sigma_{\mathsf{NTRU}}, \ell, \mathtt{B}_g$	$\mathcal{R}_Q^\ell$

We extended the implementation of homomorphic FiLIP provided in [17] to obtain our proof of concept. Our source code is publicly available[6]. For this, we used two sets of parameters, presented in Table 2, both offering 128 bits of security (LWE from [1], NTRU from [22]).

Table 2: *The parameters of NTRU gadget ciphertexts, the decomposition base of the NTRU-to-LWE key-switching, and the parameters of the LWE ciphertexts. An upper bound to σ_{LWE} is presented in Lemma 3.*

	N	Q	σ_{NTRU}	$\mathtt{B}_g$	ℓ	$\mathtt{B}_{\mathsf{ksk}}$	σ_{ksk}	n	q
Set-I	2^{10}	$912829 \approx 2^{19.8}$	$1/\sqrt{2}$	2^4	5	2^3	$1/\sqrt{2}$	610	$92683 \approx 2^{16.5}$
Set-II	2^{11}	$1073741827 \approx 2^{30}$	$1/\sqrt{2}$	2^4	8	2^5	$1/\sqrt{2}$	768	$9209716 \approx 2^{21}$

We ran all our experiments on a single core of an Intel Xeon Gold 6248R CPU at 3.00GHz, in a machine with 500 GB of RAM memory. We summarize all the practical results in Table 3. Since the client has to encrypt each bit of the FiLIP's secret key $\mathbf{k} \in \{0,1\}^{2^{14}}$ into one gadget ciphertext, the total upload, in bits, is $2^{14} \cdot \ell \cdot N \cdot \log Q$ plus the size of the key-switching key. "On-line phase" shows the running time of executing $\mathbb{Z}_p$Transcipher, and the next column shows this time divided by the number of bits, i.e., $\log p$. We note that the running times are already very low, although we have a non-optimized proof of concept (for example, one could speed it up by using a dedicated FFT library for the cyclotomic rings used in FHE instead of the general library FFTW that we used). We stress that the first loop of our transciphering is composed by $\log p$ independent calls to BinaryTranscipher, therefore, it can be easily parallelized, which should divide the total time, and thus, also the amortized time per bit, by almost $\log p$, since the step where the outputs of BinaryTranscipher are combined is very cheap compared to the running time of BinaryTranscipher itself.

Failure probabilities As it is done in virtually all FHE schemes that use subgaussian noise analysis [21,12,4], we use the central limit heuristic to model as a Gaussian the final error in the LWE ciphertexts output by $\mathbb{Z}_p$Transcipher. Moreover, based on Lemma 3, we assume the following variance:

$$\sigma_{LWE}^2 := 15^2 \cdot \log p \cdot \hat{\ell} \cdot N \cdot (\mathtt{B}'_g \cdot \sigma)^2 \cdot (q/Q)^2 + \|\mathbf{s}\|_2^2 /4 + N \cdot \hat{\ell}_{\mathsf{ksk}} \cdot (\mathtt{B}'_{\mathsf{ksk}} \cdot \sigma_{\mathsf{ksk}})^2$$

where $\hat{\ell} := \log_{\mathtt{B}_g}(Q/2)$ and $\ell_{\mathsf{ksk}} := \log_{\mathtt{B}_{\mathsf{ksk}}}(q/2)$, since in practice before decomposing the values, we can put them in the centered representation, e.g., in $[\![-q/2, ..., q/2]\!]$, instead of in $[\![0, ..., q-1]\!]$. Also,

[6] The Github repository with our source code will be made public in the final version of the paper, after acceptance.

Table 3: *Running times and upload depending on different parameter sets*

	Client's upload	Client's setup	Global setup	p	p−Setup	On-line phase	Per bit
Set-I	215 MB	3.4 s	2 s	2^2	2.6 s	13 ms	6.5 ms
				2^3	3.74 s	18.8 ms	6.3 ms
				2^4	5.26 s	25.2 ms	6.3 ms
Set-II	1 GB	11 s	6.5 s	2^2	7.4 s	36 ms	18 ms
				2^3	11 s	54 ms	18 ms
				2^4	14.7 s	71 ms	17.7 ms
				2^5	17.7 s	84.6 ms	17 ms
				2^6	21.2 s	101 ms	16.8 ms
				2^7	24.8 s	117 ms	16.7 ms
				2^8	29.7 s	137 ms	17.1 ms

Table 4: *Failure probability of output of* $\mathbb{Z}_p$Transcipher.

	p	$\log(\sigma_{\mathsf{LWE}})$	Upper bound on failure probability
Set-I	2^2	≈ 10	2^{-150}
	2^3	≈ 10.5	2^{-30}
	2^4	≈ 11	2^{-8}
Set-II	2^2	≈ 11.03	$2^{-215347}$
	2^3	≈ 11.04	2^{-53842}
	2^4	≈ 11.04	2^{-13382}
	2^5	≈ 11.05	2^{-3329}
	2^6	≈ 11.05	2^{-831}
	2^7	≈ 11.06	2^{-209}
	2^8	≈ 11.05	2^{-54}

$\mathsf{B}'_g := \mathsf{B}_g - 1$ and $\mathsf{B}'_{\mathsf{ksk}} := \mathsf{B}_{\mathsf{ksk}} - 1$, since when we decompose integers in some base B, we actually obtain values less than or equal to $B-1$. And since we used ternary keys for the NTRU secret, the value $\|\mathbf{s}\|_2$ was replaced by $\sqrt{N}$.

Notice that σ_{LWE} grows very slowly as we increase p (assuming other parameters fixed), as it is just proportional to $\sqrt{\log p}$. However, the failure probability is computed as $1-\mathsf{erf}(q/(2\cdot p\cdot\sigma_{\mathsf{LWE}}\cdot\sqrt{2}))$, thus, increasing p increases the probability exponentially (this is the case for any FHE scheme). In Table 4, we present all the values σ_{LWE} and the corresponding probabilities. We stress that this is the probability that an LWE ciphertext output by $\mathbb{Z}_p$Transcipher does not encrypt the correct value, but the failure probability of the programmable bootstrapping executed afterwards is independent of this and can be chosen by setting accordingly the parameters of the FHE scheme used to the computation — which is not necessarily the same scheme and parameters used for the transciphering.

5.2 General comparisons

Since our transciphering is adapted for so-called third generation schemes, we do comparisons with the other transcipherings performed with this type of schemes. Namely, we compare our performances to the transcipherings with FiLIP performed with TFHE in [28,18] and FINAL in [17], and Elisabeth with TFHE in [18]. For a further extension to other types of schemes which require batched ciphertexts, we discuss the possibility and the limitation in Section 6.1.

Table 5: *Comparison of running time of transcipherings with FiLIP.*

Work	Cipher	Scheme	Latency(ms)	Time per bit(ms)
[28]	FiLIP-1280	TFHE	2200	2200
	FiLIP-1216	TFHE	1900	1900
	FiLIP-144	TFHE	2500	2500
[18]	FiLIP-1280	TFHE	627	627
	FiLIP-1216	TFHE	586	586
	FiLIP-144	TFHE	134	134
[17]	FiLIP-144	FINAL	2.62	2.62
This work, Set-I, $p = 2^3$	FiLIP-144	FINAL	18.8	6.3
This work, Set-II, $p = 2^7$	FiLIP-144	FINAL	117	16.7

In Table 5, we compare the timings of the different methods proposed with FiLIP in the literature. The latency is extracted form each paper, corresponding to the time of on-line computation required to obtain an homomorphic ciphertext by the sever. All the timings correspond to monothreaded computations. We defer the comparison with Elisabeth to the next subsection, since each ciphertext contains 4 bits of information, which makes it comparable with ours for p with 4 bits.

We can observe that ours is highly competitive with the other evaluations with FiLIP. Even for larger message precision (running $\log p$ times of BinaryTranscipher), we have better computation time than the other transciphering producing an LWE ciphertext of a bit. More precisely, the latency is always lower than the one obtained in former works except [17], and the time per bit is greatly improved, of 1 or 2 orders of magnitude.

Compared to [17], the time per bit is slower but in the same order (2.4 and 6.4) whereas the latency is 7 and 44 times slower for these parameters. It is because each BinaryTranscipher requires 144 external products per bit, whereas [17] runs less than half of it. This difference comes from the instantiation of homomorphic XOR part (over $\mathbb{Z}_p$ in our case, over $\mathbb{Z}_2$ in [17]). Therefore, we use different parameters for FINAL to manage noise as well. All of these factors incur the overhead. Nevertheless, the slightly slower time per bit is acceptable since we aim for larger message precision. In other words, our transciphering will be preferable than the one of [17] when non-binary circuits are necessarily evaluated.

5.3 Comparison with Elisabeth-4

In [18], an HHE scheme is presented combining the symmetric cipher Elisabeth-4 and TFHE as FHE scheme. Elisabeth-4 works with plaintext over $\mathbb{Z}_{16}$, therefore in the transciphering presented

Table 6: *TFHE parameters for Elisabeth-4.*

Mode	n	$\log(\sigma_{LWE})$	k	N	$\log(\sigma_{GLWE})$	PBS $\log(B)$	PBS ℓ
2 KS	784	13.3342	3	512	25.5003	19	1
Single KS	863	11.2506	3	512	25.5003	19	1

in the same paper the homomorphic ciphertexts obtained contain plaintext with values modulo 16, which allows a direct comparison with our method when we fix $p = 2^4$.

Their algorithm uses homomorphic additions modulo 16 and evaluations of Negacyclic Look up Tables (NLUT) from $\mathbb{Z}_{16}$ to $\mathbb{Z}_{16}$ using the Programmable Bootstrapping (PBS). To produce each ciphertext, their evaluation requires 96 PBS corresponding to $96 \cdot n$ external products where n is the size of the LWE key and 203 LWE ciphertext additions. The error constraints to enter in the PBS require to use key switching during the evaluations, therefore the authors present two evaluations with different sets of parameters, with one or two key switchings. The parameters for these two modes are shown in Table 6, where their evaluation corresponds to 75264 external products for the mode with 2 key switchings and 82848 for the mode with a single key switching.

Table 7: *Timings comparison between the evaluation of Elisabeth-4 with TFHE from [18], and FiLIP recombining 4 bits with our transciphering. Multithreaded versions of Elisabeth-4 were probably executed on 12, or 48, or 64 threads, but this information is not explicitly written in [18].*

Evaluation	Mode	Latency (ms)	Time per bit (ms)
Elisabeth-4	2 KS, multithreaded	91.143	22.786
Elisabeth-4	Single KS, multithreaded	103.810	25.953
Elisabeth-4	2 KS, monothreaded	1485.0	371.25
Elisabeth-4	Single KS, monothreaded	1648.6	412.15
Ours	Set-I, monothreaded	25.2	6.3
Ours	Set-II, monothreaded	71	17.7

The comparison of the running time of Elisabeth-4 in [18] and ours when setting $p = 2^4$ is given in Table 7, from which we can conclude that our transciphering (for 4 bits) is much faster than the one with Elisabeth-4. Comparing to monothreaded computations, our implementations is more than 20 times faster, for the different sets of parameters. The latency we obtain is smaller but of the same order of the timings of the multithreaded evaluation of Elisabeth-4[7]), since most of the operations in our transciphering are performed independently on the 4 bits we could expect a latency close to the current time per bit with 4 threads. The main reason for such efficiency for our transciphering is that we have far smaller number of external products, namely, executing a single PBS requires more external products than transciphering one bit with our method.

On the downside, in our method, each bit of the FiLIP's secret key is encrypted into one gadget ciphertext, while in Elisabeth-4, the client sends LWE ciphertexts to the server, which can be compressed with standard techniques. Namely, each LWE ciphertext is composed by $n+1$ elements

[7] Elisabeth-4 has been designed to take advantage of the multiple PBS evaluable in parallel with Concrete, ideally running on 48 threads.

of $\mathbb{Z}_q$, but n of them are uniformly distributed and only one of them depends on the secret key. Thus, instead of sending those $n+1$ elements, the client can send the seed used to generate the n random elements together with one single element of $\mathbb{Z}_q$, hence, drastically reducing the upload. In [18], it is reported that the client just has to upload 8 KB or 20 KB, depending on the mode, to send the symmetric key encrypted with compressed LWE ciphertexts. However, to evaluate Elisabeth-4, the server also needs the bootstrapping keys, which corresponds to more than 12 MB. Thus, depending on whether one considers that the bootstrapping keys are part of the setup step of Elisabeth-4 or not, the client's upload is estimated as a few kilobytes or a few megabytes. While in our case, the client's upload ranges from megabytes to one gigabyte. We stress that if the client wants to use applications with different values of plaintext modulus p, then extra costly conversions of homomorphic ciphertexts and more uploads are needed, since Elisabeth-4 is bent to use $p = 2^4$ only.

5.4 Comparison for neural networks evaluation

Our method can shine in neural network evaluation. The versatility of our non binary transciphering allows to adapt the precision on the plaintexts, which fits well with Convolutional Neural Networks (CNN) working on quantized data. For example, the transciphering of [18] is followed by a CNN evaluating the classification of Fashion MNIST pictures homomorphically. The Fashion MNIST picture database consists of images of 784 gray pixels, each one of 8 bits of information. For a faster evaluation (taking advantage of the 4-bits PBS implemented in Concrete), the evaluation of [18] restrict the gray-scale to only 3 bits of information and homomorphically evaluate the quantized CNN. The advantage of ours is that from the encrypted data of the client, the server could choose rather to evaluate a cheap CNN with data quantized over t bits with potentially relatively low accuracy or a more costly CNN with data quantized over $t' > t$ bits with high accuracy, depending on computing environment. The CNN choice does not requires the client to re-encrypt data, and choosing the precision after client's query allows the server to adapt the cost and the precision for each functionality asked by the client.

For the particular CNN used in [18], the transciphering is considered with parameters already compatible with the PBS used for the CNN, rather than the optimal ones we recalled in Table 6. Moreover, only 3 of the 4 bits of each plaintext of Elisabeth-4 are used in the ciphertext with plaintext space $\mathbb{Z}_{16}$, since the CNN takes bootstrapped LWE ciphertexts, that allow PBS on 3 bits of data, one bit being used for padding. In [18] the transciphering with Elisabeth-4 and homomorphic inference takes 427 seconds, compared to 6 seconds without the transciphering. Using the optimal parameters for Elisabeth-4 evaluation recalled in Table 6, it reduces the total time to 77 seconds, without considering a keyswitching before entering the CNN. If we use our technique which outputs LWE ciphertexts encrypting 3 bits of integer, in the same setting, we would expect the total running time to be reduced to 15 seconds[8] (with parameter Set-I), and 43 seconds with parameter Set-II).

6 Discussion and Conclusion

6.1 Discussion

Our idea which homomorphically composes $\{0,1\}$ elements into an integer can be naturally extended to any FHE scheme which uses batching methods [6,23,11]. The naive approach for server is to

[8] With Set-I it leads to a transciphering with homomorphic inference in $28 * 28 * 18.8 * 10^{-3} + 6 = 15$ seconds

run our transciphering per coefficient, and homomorphically moves the coefficient message into the corresponding slot by computing a linear transformation [26]. However, the complexity of this process is $\mathcal{O}(N)$, where N is the number of slots, which would require more optimizations for practical uses.

Additionally, one might argue that the same property for general message precision can be achieved by functional bootstrapping [14]. However, our approach is much cheaper than running one bootstrapping since our transciphering only requires 144 times of external products (using $\log p$-threads), whereas one bootstrapping requires at least 630 up to around 900 external products depending on the desired precision.

6.2 Conclusion

In this article, we have presented a new transciphering method which can be used for any message precision for the first time. In other words, the client does not need to set message precision before sending its data to the server. Therefore, the server can reuse the given data for several application algorithms by taking only necessary upper bits of data, depending on the target application, without running different setups with the client. This approach gives more freedom to clients in cloud-based service, in terms of parameter setting and communications with the server. Hence, a service provider can offer a more user-friendly environment to the clients.

References

1. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015). https://doi.org/doi:10.1515/jmc-2015-0016, https://doi.org/10.1515/jmc-2015-0016
2. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 430–454. Springer, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015). https://doi.org/10.1007/978-3-662-46800-5_17
3. Ashur, T., Mahzoun, M., Toprakhisar, D.: Chaghri - a fhe-friendly block cipher. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 139–150. CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3548606.3559364, https://doi.org/10.1145/3548606.3559364
4. Bonte, C., Iliashenko, I., Park, J., Pereira, H.V.L., Smart, N.P.: Final: Faster fhe instantiated with ntru and lwe. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology – ASIACRYPT 2022. pp. 188–215. Springer Nature Switzerland, Cham (2022)
5. Brabant, M., Pereira, O., Méaux, P.: Homomorphic encryption for privacy-friendly augmented democracy. In: 2022 IEEE 21st Mediterranean Electrotechnical Conference (MELECON). pp. 18–23 (2022). https://doi.org/10.1109/MELECON53508.2022.9843009
6. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012: 3rd Innovations in Theoretical Computer Science. pp. 309–325. Association for Computing Machinery, Cambridge, MA, USA (Jan 8–10, 2012). https://doi.org/10.1145/2090236.2090262
7. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In: Peyrin, T. (ed.) Fast Software Encryption – FSE 2016. Lecture Notes in Computer Science, vol. 9783, pp. 313–333. Springer, Heidelberg, Germany, Bochum, Germany (Mar 20–23, 2016). https://doi.org/10.1007/978-3-662-52993-5_16
8. Carlet, C., Méaux, P.: A complete study of two classes of boolean functions: Direct sums of monomials and threshold functions. IEEE Trans. Inf. Theory **68**(5), 3404–3425 (2022). https://doi.org/10.1109/TIT.2021.3139804, https://doi.org/10.1109/TIT.2021.3139804
9. Chen, H., Chillotti, I., Ren, L.: Onion ring ORAM: Efficient constant bandwidth oblivious RAM from (leveled) TFHE. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019: 26th Conference on Computer and Communications Security. pp. 345–360. ACM Press, London, UK (Nov 11–15, 2019). https://doi.org/10.1145/3319535.3354226

10. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019: 26th Conference on Computer and Communications Security. pp. 395–412. ACM Press, London, UK (Nov 11–15, 2019). https://doi.org/10.1145/3319535.3363207
11. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017, Part I. Lecture Notes in Computer Science, vol. 10624, pp. 409–437. Springer, Heidelberg, Germany, Hong Kong, China (Dec 3–7, 2017). https://doi.org/10.1007/978-3-319-70694-8_15
12. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 3–33. Springer, Heidelberg, Germany, Hanoi, Vietnam (Dec 4–8, 2016). https://doi.org/10.1007/978-3-662-53887-6_1
13. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (Jan 2020). https://doi.org/10.1007/s00145-019-09319-x
14. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: Dolev, S., Margalit, O., Pinkas, B., Schwarzmann, A. (eds.) Cyber Security Cryptography and Machine Learning. pp. 1–19. Springer International Publishing, Cham (2021)
15. Cho, J., Ha, J., Kim, S., Lee, B., Lee, J., Lee, J., Moon, D., Yoon, H.: Transciphering framework for approximate homomorphic encryption. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021, Part III. Lecture Notes in Computer Science, vol. 13092, pp. 640–669. Springer, Heidelberg, Germany, Singapore (Dec 6–10, 2021). https://doi.org/10.1007/978-3-030-92078-4_22
16. Cong, K., Das, D., Nicolas, G., Park, J.: Panacea: Non-interactive and stateless oblivious ram. Cryptology ePrint Archive, Paper 2023/274 (2023), https://eprint.iacr.org/2023/274, https://eprint.iacr.org/2023/274
17. Cong, K., Das, D., Park, J., Pereira, H.V.: Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 563–577. CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3548606.3560702, https://doi.org/10.1145/3548606.3560702
18. Cosseron, O., Hoffmann, C., Méaux, P., Standaert, F.X.: Towards case-optimized hybrid homomorphic encryption - featuring the elisabeth stream cipher. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology – ASIACRYPT 2022, Part III. Lecture Notes in Computer Science, vol. 13793, pp. 32–67. Springer, Heidelberg, Germany, Taipei, Taiwan (Dec 5–9, 2022). https://doi.org/10.1007/978-3-031-22969-5_2
19. Dobraunig, C., Eichlseder, M., Grassi, L., Lallemand, V., Leander, G., List, E., Mendel, F., Rechberger, C.: Rasta: A cipher with low ANDdepth and few ANDs per bit. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part I. Lecture Notes in Computer Science, vol. 10991, pp. 662–692. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018). https://doi.org/10.1007/978-3-319-96884-1_22
20. Dobraunig, C., Grassi, L., Helminger, L., Rechberger, C., Schofnegger, M., Walch, R.: Pasta: A case for hybrid homomorphic encryption. Cryptology ePrint Archive, Report 2021/731 (2021), https://eprint.iacr.org/2021/731
21. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 617–640. Springer, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015). https://doi.org/10.1007/978-3-662-46800-5_24
22. Ducas, L., van Woerden, W.: Ntru fatigue: How stretched is overstretched? In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021. pp. 3–32. Springer International Publishing, Cham (2021)
23. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), https://eprint.iacr.org/2012/144
24. Ha, J., Kim, S., Choi, W., Lee, J., Moon, D., Yoon, H., Cho, J.: Masta: An he-friendly cipher using modular arithmetic. IEEE Access **8**, 194741–194751 (2020). https://doi.org/10.1109/ACCESS.2020.3033564
25. Ha, J., Kim, S., Lee, B., Lee, J., Son, M.: Rubato: Noisy ciphers for approximate homomorphic encryption. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022, Part I. Lecture Notes in Computer Science, vol. 13275, pp. 581–610. Springer, Heidelberg, Germany, Trondheim, Norway (May 30 – Jun 3, 2022). https://doi.org/10.1007/978-3-031-06944-4_20
26. Halevi, S., Shoup, V.: Bootstrapping for HElib. Journal of Cryptology **34**(1), 7 (Jan 2021). https://doi.org/10.1007/s00145-020-09368-7
27. Hebborn, P., Leander, G.: Dasta – alternative linear layer for rasta. IACR Transactions on Symmetric Cryptology **2020**(3), 46–86 (Sep 2020). https://doi.org/10.13154/tosc.v2020.i3.46-86, https://tosc.iacr.org/index.php/ToSC/article/view/8696

28. Hoffmann, C., Méaux, P., Ricosset, T.: Transciphering, using FiLIP and TFHE for an efficient delegation of computation. In: Bhargavan, K., Oswald, E., Prabhakaran, M. (eds.) Progress in Cryptology - INDOCRYPT 2020: 21st International Conference in Cryptology in India. Lecture Notes in Computer Science, vol. 12578, pp. 39–61. Springer, Heidelberg, Germany, Bangalore, India (Dec 13–16, 2020). https://doi.org/10.1007/978-3-030-65277-7_3
29. Jeon, S., Lee, H.S., Park, J.: Practical randomized lattice gadget decomposition with application to fhe. Cryptology ePrint Archive, Paper 2023/535 (2023), https://eprint.iacr.org/2023/535, https://eprint.iacr.org/2023/535
30. Kim, M., Jiang, X., Lauter, K., Ismayilzada, E., Shams, S.: Secure human action recognition by encrypted neural network inference. Nature Communications **13** (08 2022). https://doi.org/10.1038/s41467-022-32168-5
31. Méaux, P., Carlet, C., Journault, A., Standaert, F.X.: Improved filter permutators for efficient FHE: Better instances and implementations. In: Hao, F., Ruj, S., Sen Gupta, S. (eds.) Progress in Cryptology - INDOCRYPT 2019: 20th International Conference in Cryptology in India. Lecture Notes in Computer Science, vol. 11898, pp. 68–91. Springer, Heidelberg, Germany, Hyderabad, India (Dec 15–18, 2019). https://doi.org/10.1007/978-3-030-35423-7_4
32. Méaux, P., Journault, A., Standaert, F.X., Carlet, C.: Towards stream ciphers for efficient FHE with low-noise ciphertexts. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part I. Lecture Notes in Computer Science, vol. 9665, pp. 311–343. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016). https://doi.org/10.1007/978-3-662-49890-3_13
33. Naehrig, M., Lauter, K.E., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Cachin, C., Ristenpart, T. (eds.) Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011. pp. 113–124. ACM (2011)
34. Pereira, H.V.L.: Bootstrapping fully homomorphic encryption over the integers in less than one second. In: Garay, J. (ed.) PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, vol. 12710, pp. 331–359. Springer, Heidelberg, Germany, Virtual Event (May 10–13, 2021). https://doi.org/10.1007/978-3-030-75245-3_13
35. Stoian, A., Frery, J., Bredehoft, R., Montero, L., Kherfallah, C., Chevallier-Mames, B.: Deep neural networks for encrypted inference with tfhe. Cryptology ePrint Archive, Paper 2023/257 (2023), https://eprint.iacr.org/2023/257, https://eprint.iacr.org/2023/257
36. Tueno, A., Boev, Y., Kerschbaum, F.: Non-interactive private decision tree evaluation. In: IFIP Annual Conference on Data and Applications Security and Privacy. pp. 174–194. Springer (2020)
37. Zuber, M., Sirdey, R.: Efficient homomorphic evaluation of k-nn classifiers. Proceedings on Privacy Enhancing Technologies **2021**, 111 – 129 (2021)

A Homomorphic XOR modulo p

We start with the simplest scenario where the ciphertexts only encrypt bits, but using $\mathbb{Z}_p$ as the message space. Given $m_0, m_1 \in \{0,1\}$, we can see that

$$\mathsf{XOR}(m_0, m_1) = m_0 + m_1 - 2 \cdot m_0 \cdot m_1 \pmod p$$

Thus, let $\mathbf{c}_0 \in \mathsf{RingCtxt}_s(\lfloor Q/p \rceil \cdot m_0, E_0)$ and $\mathbf{C}_1 \in \mathsf{GadgetCtxt}_s^{Q,\ell}(m_1, E_1)$. Then, as shown in Algorithm 8, on can compute the homomorphic XOR as

$$\mathsf{FHE.XOR}(\mathbf{c}_0, \mathbf{C}_1) := \mathbf{c}_0 + (1 - 2 \cdot \mathbf{c}_0) \boxdot \mathbf{C}_1$$

By Corollary 1, it follows that $\mathsf{FHE.XOR}(\mathbf{c}_0, \mathbf{C}_1) \in \mathsf{RingCtxt}_s(\Delta \cdot \mathsf{XOR}(m_0, m_1), E)$ where $E \leq \sqrt{\ell N} \cdot \mathsf{B}_g \cdot E_1 + E_0$.

Algorithm 8: FHE.XORSimple

Input: $\mathbf{c}_0 \in \mathsf{RingCtxt}_s(\Delta \cdot m_0, E_0)$ and $\mathbf{C}_1 \in \mathsf{GadgetCtxt}_s^{Q,\ell}(m_1, E_1)$ where $m_0, m_1 \in \{0,1\}$ and $\Delta = \lfloor Q/p \rceil$.
Output: $\mathbf{c} \in \mathsf{RingCtxt}_s(\Delta \cdot \mathsf{XOR}(m_0, m_1), E_{out})$
Noise growth: $E_{out} \leq \sqrt{\ell N} \cdot \mathtt{B}_g \cdot E_1 + E_0$

1 $\mathbf{c}' = \mathsf{FHE.MultPtxt}(-2, \mathbf{c}_0)$; ▷ $\mathsf{RingCtxt}_s(-2 \cdot \Delta \cdot m_0)$
2 $\mathbf{c}'' = \mathsf{FHE.AddPlaintext}(1, \mathbf{c}')$; ▷ $\mathsf{RingCtxt}_s(\Delta(1 - 2 \cdot m_0))$
3 $\mathbf{c} = \mathsf{FHE.ExtProd}(\mathbf{c}'', \mathbf{C}_1)$; ▷ $\mathsf{RingCtxt}_s(\Delta(m_1 - 2 \cdot m_0 \cdot m_1))$
4 **return c**