Optimized stream-cipher-based transciphering by means of functional-bootstrapping[⋆]

Adda-Akram Bendoukha[1⋆⋆], Pierre-Emmanuel Clet[2], Aymen Boudguiga[2], and Renaud Sirdey[2]

[1]Samovar, Télécom SudParis, Institut Polytechnique de Paris, 91120 Palaiseau, France adda-akram.bendoukha@telecom-sudparis.eu
[2]Université Paris-Saclay, CEA-List, Gif-sur-Yvette, France.
firstname.lastname@cea.fr

Keywords: FHE · stream-ciphers · transciphering · functional bootstrapping

Abstract. Fully homomorphic encryption suffers from a large expansion in the size of encrypted data, which makes FHE impractical for low-bandwidth networks. Fortunately, transciphering allows to circumvent this issue by involving a symmetric cryptosystem which does not carry the disadvantage of a large expansion factor, and maintains the ability to recover an FHE ciphertext with the cost of extra homomorphic computations on the receiver side. Recent works have started to investigate the efficiency of TFHE as the FHE layer in transciphering, combined with various symmetric schemes including a NIST finalist for lightweight cryptography, namely Grain128-AEAD. Yet, this has so far been done without taking advantage of TFHE functional bootstrapping abilities, that is, evaluating any discrete function "for free" within the bootstrapping operation. In this work, we thus investigate the use of TFHE functional bootstrapping for implementing Grain128-AEAD in a more efficient base ($B > 2$) representation, rather than a binary one. This significantly reduces the overall number of necessary bootstrappings in a homomorphic run of the stream-cipher, for example reducing the number of bootstrappings required in the warm-up phase by a factor of ≈ 3 when $B = 16$.

1 Introduction

Despite its privacy advantages in cloud services, Fully Homomorphic Encryption (FHE) has a notable drawback of expanding the size of encrypted data to a significant extent, which limits its practicality for low-bandwidth networks. However, transciphering provides a solution by using a symmetric cryptosystem

⋆ This work was supported by the France ANR project ANR-22-CE39-0002 EQUIHID (first author) and France 2030 ANR project ANR-22-PECY-003 SecureCompute (third and fourth authors).

⋆⋆ This author's contribution to this work was done while at CEA List.

that does not result in a large expansion factor and still allows the recovery of an FHE ciphertext with extra computation on the receiver's end.

Previous research [BBS22] has shown the effectiveness of TFHE [CGGI18] as the FHE layer in transciphering, combined with various symmetric schemes such as Grain128-AEAD [HJMM06], which is a finalist for NIST call for lightweight cryptography. The functional bootstrapping feature of TFHE allows the evaluation of any function with the bootstrapping operation, leading to a significant improvement in transciphering performance.

This work explores the utilization of TFHE's functional bootstrapping to implement the boolean operators which are required for updating the internal state of a stream-cipher. The approach involves representing the internal state of the stream-cipher using decomposition in a base B ($B > 2$) instead of a binary one. This results in reducing the necessary number of bootstrappings in a homomorphic run of the stream-cipher by a factor of $\log(B)$. As such, this approach is expected to provide a notable speedup on the server side.

1.1 Transciphering

Transciphering, also referred to as Proxy-reencryption, uses a symmetric cryptosystem as a way of securely *compressing* FHE ciphertexts prior to their transmission to the cloud. Regarding symmetric cryptosystems, the expansion factor is approximately equal to 1[1]. Meanwhile, *decompressing* the ciphertext returns a full-size FHE ciphertext that will be used as input to the desired computation on the cloud side.

The main idea behind transciphering is to reduce the size of homomorphically encrypted data to be sent to the Cloud by encrypting it symmetrically instead, and preserving the ability to recover a homomorphic encryption of the initial data. To do so, a client encrypts his message m with a symmetric encryption scheme as: $\mathsf{SYM.Enc}_{\mathsf{SYM.sk}}(m)$, and encrypts the symmetric key $\mathsf{SYM.sk}$ with a homomorphic cryptosystem as: $\mathsf{FHE.Enc}_{\mathsf{FHE.pk}}(\mathsf{SYM.sk})$. At the reception of $(\mathsf{SYM.Enc}_{\mathsf{SYM.sk}}(m), \mathsf{FHE.Enc}_{\mathsf{FHE.pk}}(\mathsf{SYM.sk}))$, the Cloud server homomorphically runs the symmetric cryptosystem's decryption function. That is, he evaluates $\mathsf{SYM.Dec}$ using the FHE encryption of the symmetric key: $\mathsf{FHE.Enc}_{\mathsf{FHE.pk}}(\mathsf{SYM.sk})$ to get $\mathsf{FHE.Enc}_{\mathsf{FHE.pk}}(m)$.

With a stream-cipher, the client encrypts his message m with a keystream ks as: $m \oplus ks$, where $\oplus$ is the `XOR` operator. Then, he sends $m \oplus ks$ and the stream-cipher secret key $\mathsf{FHE.Enc}_{\mathsf{FHE.pk}}(\mathsf{SYM.sk})$ to the Cloud server. The latter runs the stream-cipher warm-up homomorphically with $\mathsf{FHE.Enc}_{\mathsf{FHE.pk}}(\mathsf{SYM.sk})$ to get: $\mathsf{FHE.Enc}_{\mathsf{FHE.pk}}(ks)$. Then, he computes $m \oplus ks \oplus \mathsf{FHE.Enc}_{\mathsf{FHE.pk}}(ks)$ to obtain: $\mathsf{FHE.Enc}_{\mathsf{FHE.pk}}(m)$.

The size of the plaintext m can be arbitrarily large, whereas the size of the symmetric key $\mathsf{SYM.sk}$ is constant and typically small enough to be homomorphically encrypted and transmitted only once. This results in compression, which

[1] Symmetric algorithms will usually lead to negligible or small overheads due to some padding rule.

comes at the expense of running the symmetric scheme's decryption function homomorphically on the server side.

In terms of security, as most practically-used symmetric encryption algorithms do not have formally established indistinguishability properties, it should be emphasized that using transciphering jeopardizes the IND-CPA property of the FHE scheme (FHE schemes can be at most IND-CCA1 and all the schemes presently used in practice are only IND-CPA). This should however not be considered an issue in practice, provided that symmetric encryption more often than not teams with provably-secure public-key encryption for efficiency reasons in practical scenarios, and FHE is no exception. Nevertheless, if we assume a perfect PRF on the symmetric side, the resulting construction would be IND-CPA [CCF$^+$18a].

Yet, one important point is to choose the key size of the symmetric encryption algorithm consistently with the parameters of the FHE scheme. At present, common practice generally targets FHE security parameters λ of around 128 bits and not more as FHE performances significantly decreases in the parameter regimes of larger λ. As such, at present, transciphering should consider symmetric algorithms with 128 bits keys.

1.2 Related work

In [CCF$^+$18a] authors introduced transciphering using a stream-cipher, as a secure and efficient way of compressing FHE ciphertexts. They also provide the first FHE-friendly 80-bits-key stream-cipher, Trivium, which was later extended to Kreyvium [CCF$^+$18b] that accommodates a 128-bits key.

With the advent of TFHE, and its fast bootstrapping operation, multiplicative depth is no longer the bottleneck in fast-bootstrapping-based FHE. Therefore, the possibility to homomorphically evaluate more sophisticated ciphers became reachable. In [HMR20] authors evaluate FiLip with third generation homomorphic schemes GSW [GSW13] and TFHE. [CHK$^+$20] propose a stream-cipher dubbed HERA suited for a hybrid HE framework allowing switching from CKKS [CKKS16] to BFV [FV12]. On the other hand, [DGH$^+$21] provides a framework to build transciphering schemes, as well as the a stream-cipher called PASTA, the last one of the RASTA family of ciphers. PASTA is well suited for HE schemes supporting batching such as BGV [BGV12] and BFV [FV12].

Homomorphic evaluations of block-ciphers also has been studied under both levelled and bootstrapping-based HE schemes. Low-MC [ARS$^+$15] was designed as an attempt to provide a minimal multiplicative-depth yet secure block-cipher suited for levelled HE schemes. Other attempts to provide optimized FHE implementations of the advanced encryption standard AES were investigated [GHS12a] but remain unpractical for real-life client/server use-cases. Subsequent works attempting to design FHE-friendly ciphers investigate the design of generic smaller stream or block-cipher components such as reduced-multiplicative depth Sboxes [BP12] or boolean functions [CM19]. These components can then be integrated in the design of FHE-friendly ciphers, providing a reasonable security/FHE-friendliness tradeoff.

As for TFHE, in [GBA21], $\mathbb{Z}_B$ plaintext spaces of more than two elements were investigated. In addition, many novel techniques [GBA21, KS22, OKC20, CZB+22] investigated TFHE functional bootstrapping for evaluating discrete non-linear functions over integers represented by their digits decomposition in a base $B > 2$.

1.3 Contribution

Unlike previous works where the efforts were made to build FHE-friendly ciphers that accommodate an FHE scheme or a class of FHE schemes [2] we focus on existing and standardized symmetric schemes. We provide a generic technique which reduces the number of bootstrappings required to *transcipher* from a symmetric ciphertext to a TFHE one, that applies to a wide range of ciphers. We validate our approach by providing an efficient implementation of Grain128-AEAD with TFHE. Indeed, thanks to its bootstrapping, TFHE offers greater freedom in the choice of the symmetric cryptosystem used for transciphering, allowing to perform more operations and relaxing the constraints imposed by the multiplicative depth [3].

Overall, we improve the work of Bendoukha et al., [BBS22], in which the authors benchmarked implementations of transciphering with various stream-ciphers (Trivium, Kreyvium and Grain128-AEAD) with TFHE, using the straightforward representation of the internal states of stream-ciphers as arrays of TFHE encrypted bits. In this work, we investigate a digit representation of these stream-ciphers internal states in a sequence of k digits in base B.

For this purpose we redefine boolean operations in bases $B > 2$ making the most of TFHE's functional bootstrapping. Reducing the number of ciphertexts reduces the number of bootstrappings needed to update the internal state of a stream-cipher by a factor $\log_2(B)$. Since the bootstrapping is by far the most time-consuming operation in TFHE, this technique results in a significant speed-up of the decompression phase on the server side.

To illustrate the speed-up brought by this technique, we implement a NIST finalist for lightweight cryptography, Grain128-AEAD [HJMM06], using the TFHE library [4]. Our implementation is 40% faster than the one from [BBS22] with binary ciphertexts representation of cipher's internal state. The choice of Grain128-AEAD is motivated by the fact that besides (so far) unpractical attempts to run AES in the homomorphic domain [GHS12b], no other works investigates the use of a standard cipher in a transciphering construction. Therefore, our work aims at demonstrating the feasibility and practicality of standard ciphers under TFHE. In addition, its lightweight design induces a small number of gates which translates in a small number of homomorphic operations when using TFHE (which is

[2] based on their approach : bootstrapping or levelled, and their plaintext-space: binary, real, or $\mathbb{Z}_q$

[3] The multiplicative depth of a circuit is the maximum number of successive multiplications in the circuit.

[4] https://github.com/tfhe/tfhe

not subject to multiplicative depth contraints). Additionally, it is also amenable to efficient x-bits implementations (e.g., as in [BLSK19]) which represents a good starting point for using TFHE beyond gate-bootstrapping. Grain128-AEAD provides also a MAC computation along with the encryption procedure. This feature can be used complementarily to transciphering in order to perform an oblivious integrity check on the server side as described in [BBS21].

1.4 Paper organization

The remainder of this paper is organized as follows. After recalling the principle of transciphering as a generic framework in the introduction (in Section 1.1), we review our target FHE cryptosystem TFHE in Section 2. Then, in Section 3, we describe how to use TFHE functional bootstrapping for computing boolean operations efficiently in the homomorphic domain. Indeed, we specify boolean operations for encrypted digits, from bases 4 and 16. In Section 4, we apply our new operations to transciphering with the stream-cipher Grain128-AEAD [HJM$^+$21]. Finally, we discuss implementation details and performance results in Section 5 before concluding the paper.

2 TFHE

2.1 Notations

We refer to the real torus by $\mathbb{T} = \mathbb{R}/\mathbb{Z}$. $\mathbb{T}$ is the additive group of real numbers modulo 1 ($\mathbb{R}$ $mod[1]$) and it is a $\mathbb{Z}$-module. $\mathbb{T}_N[X]$ denotes the $\mathbb{Z}$-module $\mathbb{R}[X]/(X^N + 1)$ $mod[1]$ of torus polynomials, where N is a power of 2. $\mathcal{R}$ is the ring $\mathbb{Z}[X]/(X^N + 1)$ and its subring of polynomials with binary coefficients is $\mathbb{B}_N[X] = \mathbb{B}[X]/(X^N + 1)$ ($\mathbb{B} = \{0, 1\}$). Finally, we denote respectively by $[x]_{\mathbb{T}}$, $[x]_{\mathbb{T}_N[X]}$ and $[x]_{\mathcal{R}}$ the encryption of x over $\mathbb{T}$, $\mathbb{T}_N[X]$ or $\mathcal{R}$. The plaintext space (prior to encoding as torus elements) with respect to the FHE encryption layer is denoted $\mathcal{M}$.

We refer to vectors by bold letters. $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ is the inner product of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. We denote matrices by capital letters, and the set of matrices with m rows and n columns with entries sampled in $\mathbb{K}$ by $\mathcal{M}_{m,n}(\mathbb{K})$. $x \overset{\$}{\leftarrow} \mathbb{K}$ denotes sampling x uniformly from $\mathbb{K}$, while $x \xleftarrow{\mathcal{N}(\mu,\sigma^2)} \mathbb{K}$ refers to sampling x from $\mathbb{K}$ following a Gaussian distribution of mean μ and variance σ^2.

2.2 TFHE Structures

The TFHE encryption scheme was proposed in 2016 [CGGI16]. It introduces the TLWE problem as an adaptation of the LWE problem to $\mathbb{T}$. TFHE relies on three structures to encrypt plaintexts defined over $\mathbb{T}$, $\mathbb{T}_N[X]$ or $\mathcal{R}$:

- **TLWE Sample:** $(\boldsymbol{a}, b)$ is a valid TLWE sample if $\boldsymbol{a} \overset{\$}{\leftarrow} \mathbb{T}^n$ and $b \in \mathbb{T}$ verifies $b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e$, where $\boldsymbol{s} \overset{\$}{\leftarrow} \mathbb{B}^n$ is the secret key, and $e \xleftarrow{\mathcal{N}(0,\sigma^2)} \mathbb{T}$.

- **TRLWE Sample:** a pair $(\boldsymbol{a}, b) \in \mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$ is a valid TRLWE sample if $\boldsymbol{a} \overset{\$}{\leftarrow} \mathbb{T}_N[X]^k$, and $b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e$, where $\boldsymbol{s} \overset{\$}{\leftarrow} \mathbb{B}_N[X]^k$ is a TRLWE secret key and $e \xleftarrow{\mathcal{N}(0,\sigma^2)} \mathbb{T}_N[X]$ is a noise polynomial.
  Let $\mathcal{M} \subset \mathbb{T}_N[X]$ (or $\mathcal{M} \subset \mathbb{T}$) be the discrete message space[5]. To encrypt a message $m \in \mathcal{M} \subset \mathbb{T}_N[X]$, we add $(\boldsymbol{0}, m) \in \mathbb{T}_N[X]^k \times \mathbb{T}_N[X]$ to a fresh TRLWE sample (or a fresh TLWE sample if $\mathcal{M} \subset \mathbb{T}$). In the following, we refer to an encryption of m with the secret key $\boldsymbol{s}$ as a T(R)LWE ciphertext noted $\boldsymbol{c} \in \mathrm{T(R)LWE}_{\boldsymbol{s}}(m)$.
  To decrypt a sample $\boldsymbol{c} \in \mathrm{T(R)LWE}_{\boldsymbol{s}}(m)$, we compute its *phase* $\phi(\boldsymbol{c}) = b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle = m + e$. Then, we round to it to the nearest element of $\mathcal{M}$. Therefore, if the error e was chosen to be small enough while ensuring security, the decryption will be accurate.
- **TRGSW Sample:** a vector of $(k+1) \cdot l$ TRLWE samples is a TRGSW sample. To encrypt a message $m \in \mathcal{R}$, we add $m \cdot H$ to a TRGSW sample, where H is a gadget matrix[6] using an integer B_g as a basis for its decomposition. Chilotti et al., [CGGI18] defines an external product between a TRGSW ciphertext A encrypting $m_a \in \mathcal{R}$ and a TRLWE ciphertext $\boldsymbol{b}$ encrypting $m_b \in \mathbb{T}_N[X]$. This external product consists in multiplying A by the approximate decomposition of $\boldsymbol{b}$ with respect to H (Definition 3.12 in [CGGI18]). It yields an encryption of $m_a \cdot m_b$ i.e., a TRLWE ciphertext $\boldsymbol{c} \in \mathrm{TRLWE}_{\boldsymbol{s}}(m_a \cdot m_b)$. Otherwise, the external product allows also to compute a controlled MUX gate (CMUX) where the selector is $C_b \in \mathrm{TRGSW}_{\boldsymbol{s}}(b)$, $b \in \{0, 1\}$, and the inputs are $\boldsymbol{c}_0 \in \mathrm{TRLWE}_{\boldsymbol{s}}(m_0)$ and $\boldsymbol{c}_1 \in \mathrm{TRLWE}_{\boldsymbol{s}}(m_1)$.

2.3 TFHE Bootstrapping

TFHE bootstrapping relies mainly on three building blocks:

- **Blind Rotate:** rotates a plaintext polynomial encrypted as a TRLWE ciphertext by an encrypted position ($\boldsymbol{c}_p \in \mathrm{TRLWE}_{\boldsymbol{s}}(p)$). It takes as inputs: a TRLWE ciphertext $\boldsymbol{c} \in \mathrm{TRLWE}_{\boldsymbol{k}}(m)$, a rescaled and rounded vector of $\boldsymbol{c}_p$ represented by $(a_1, \ldots, a_n, a_{n+1} = b)$ where $\forall i,\ a_i \in \mathbb{Z}_{2N}$, and n TRGSW ciphertexts encrypting $(s_1, \ldots, s_n)$ where $\forall i,\ s_i \in \mathbb{B}$. It returns a TRLWE ciphertext $\boldsymbol{c'} \in \mathrm{TRLWE}_{\boldsymbol{k}}(X^{\langle \boldsymbol{a}, \boldsymbol{s} \rangle - b} \cdot m)$. In this paper, we will refer to this algorithm by $\mathsf{BlindRotate}$. With respect to independence heuristic[7] stated in [CGGI18], the variance $\mathcal{V}_{BR}$ of the resulting noise after a $\mathsf{BlindRotate}$ satisfies the formula:

$$\mathcal{V}_{BR} < V_c + \mathcal{E}_{BR} \tag{1}$$

[5] In practice, we discretize the Torus with respect to our plaintext modulus. For example, if we want to encrypt $m \in \mathbb{Z}_4 = \{0, 1, 2, 3\}$, we encode it in $\mathbb{T}$ as one of the following value $\{0, 0.25, 0.5, 0.75\}$.

[6] Refer to Definition 3.6 and Lemma 3.7 in TFHE paper [CGGI18] for more information about the gadget matrix H.

[7] The independence heuristic ensures that all the coefficients of the errors of TLWE, TRLWE or TRGSW samples are independent and concentrated. More precisely, they are σ-subgaussian where σ is the square-root of their variance.

where $\mathcal{E}_{BR} = n\left((k+1)\ell N\left(\frac{B_g}{2}\right)^2\vartheta_{BK} + \frac{(1+kN)}{4.Bg^{2l}}\right)$.
V_c is the variance of the noise of the input ciphertext $\boldsymbol{c}$, and ϑ_{BK} is the the variance of the error of the bootstrapping key. Note that the noise of the BlindRotate is independent from the noise of the encrypted position $\boldsymbol{c}_p$.

- **TLWE Sample Extract:** takes as inputs both a ciphertext $\boldsymbol{c} \in \text{TRLWE}_{\boldsymbol{k}}(m)$ and a position $p \in [\![0, N[\![$, and returns a TLWE ciphertext $\boldsymbol{c'} \in \text{TLWE}_{\boldsymbol{k}}(m_p)$ where m_p is the p^{th} coefficient of the polynomial m. In this paper, we will refer to this algorithm by SampleExtract. This algorithm does not add any noise to the ciphertext.
- **Public Functional Keyswitching:** transforms a set of p ciphertexts $\boldsymbol{c}_i \in \text{TLWE}_{\boldsymbol{k}}(m_i)$ into the resulting ciphertext $\boldsymbol{c'} \in \text{T(R)LWE}_{\boldsymbol{s}}(f(m_1, \dots, m_p))$, where $f()$ is a public linear morphism from $\mathbb{T}^p$ to $\mathbb{T}_N[X]$. This algorithm requires 2 parameters: the decomposition basis B_{KS} and the precision of the decomposition t. In this paper, we will refer to this algorithm by KeySwitch. As stated in [CGGI18, GBA21], the variance $\mathcal{V}_{KS}$ of the resulting noise after KeySwitch follows the formula:
$$\mathcal{V}_{KS} < R^2 \cdot V_c + \mathcal{E}_{KS}^{n,N} \tag{2}$$
where $\mathcal{E}_{KS}^{n,N} = nN\left(t\vartheta_{KS} + \frac{base^{-2t}}{4}\right)$.
V_c is the variance of the noise of the input ciphertext c, R is the Lipschitz constant of f and ϑ_{KS} the variance of the error of the keyswitching key. In this paper and in most cases, $R = 1$.

TFHE specifies a gate bootstrapping to reduce the noise level of a TLWE sample that encrypts the result of a boolean gate evaluation on two ciphertexts, each of them encrypting a binary input. TFHE gate bootstrapping steps are summarized in Algorithm 1. The step 1 consists in selecting a value $\hat{m} \in \mathbb{T}$ which will serve later for setting the coefficients of the test polynomial $testv$ (in step 3). The step 2 rescales the components of the input ciphertext $\boldsymbol{c}$ as elements of $\mathbb{Z}_{2N}$. The step 3 defines the test polynomial $testv$. Note that for all $p \in [\![0, 2N[\![$, the constant term of $testv \cdot X^p$ is $\hat{m}$ if $p \in]\!]\frac{N}{2}, \frac{3N}{2}]\!]$ and $-\hat{m}$ otherwise. The step 4 returns an accumulator $ACC \in \text{TRLWE}_{\boldsymbol{s'}}(testv \cdot X^{\langle \bar{\boldsymbol{a}}, \boldsymbol{s}\rangle - \bar{b}})$. Indeed, the constant term of ACC is $-\hat{m}$ if $\boldsymbol{c}$ encrypts 0, or $\hat{m}$ if $\boldsymbol{c}$ encrypts 1 as long as the noise of the ciphertext is small enough. Then, step 5 creates a new ciphertext $\bar{\boldsymbol{c}}$ by extracting the constant term of ACC and adding to it $(\boldsymbol{0}, \hat{m})$. That is, $\bar{\boldsymbol{c}}$ either encrypts 0 if $\boldsymbol{c}$ encrypts 0, or m if $\boldsymbol{c}$ encrypts 1 (By choosing $m = \frac{1}{2}$, we get a fresh encryption of 1).

Since a bootstrapping operation is a BlindRotate over a noiseless TRLWE followed by a Keyswitch, the bootstrapping noise ($\mathcal{V}_{BS}$) satisfies:
$$\mathcal{V}_{BS} < \mathcal{E}_{BS}, \text{ where } \mathcal{E}_{BS} = \mathcal{E}_{BR} + \mathcal{E}_{KS}^{N,1} \tag{3}$$

2.4 TFHE Functional Bootstrapping

Functional bootstrapping [CJP21, KS21, YXS$^+$21, CLOT21, CZB$^+$22] refers to TFHE ability of implementing a Look-Up Table (LUT) of any function through

Algorithm 1 TFHE gate bootstrapping [CGGI18]

```
Input: a constant $m \in \mathbb{T}$, a TLWE sample $\boldsymbol{c} = (\boldsymbol{a}, b) \in \mathrm{TLWE}_{\boldsymbol{s}}(x \cdot \frac{1}{2})$ with $x \in \mathbb{B}$,
    a bootstrapping key $BK_{\boldsymbol{s} \rightarrow \boldsymbol{s'}} = (BK_i \in \mathrm{TRGSW}_{S'}(s_i))_{i \in [\![1,n]\!]}$ where $S'$ is the
    TRLWE interpretation of a secret key $\boldsymbol{s'}$
Output: a TLWE sample $\overline{\boldsymbol{c}} \in \mathrm{TLWE}_{\boldsymbol{s}}(x.m)$
1: Let $\hat{m} = \frac{1}{2}m \in \mathbb{T}$ (pick one of the two possible values)
2: Let $\bar{b} = \lfloor 2Nb \rceil$ and $\bar{a}_i = \lfloor 2Na_i \rceil \in \mathbb{Z}, \forall i \in [\![1,n]\!]$
3: Let $testv := (1 + X + \cdots + X^{N-1}) \cdot X^{\frac{N}{2}} \cdot \hat{m} \in \mathbb{T}_N[X]$
4: $ACC \leftarrow \mathsf{BlindRotate}((\mathbf{0}, testv), (\bar{a}_1, \ldots, \bar{a}_n, \bar{b}), (BK_1, \ldots, BK_n))$
5: $\overline{\boldsymbol{c}} = (\mathbf{0}, \hat{m}) + \mathsf{SampleExtract}(ACC)$
6: return $\mathsf{KeySwitch}_{\boldsymbol{s'} \rightarrow \boldsymbol{s}}(\overline{\boldsymbol{c}})$
```

the bootstrapping. In particular, TFHE is well-suited for negacyclic function[8], as the plaintext space for TFHE is $\mathbb{T}$, where $[0, \frac{1}{2}[$ corresponds to positive values and $[\frac{1}{2}, 1[$ to negative ones, and the bootstrapping step 2 of the Algorithm 1 encodes elements from $\mathbb{T}$ into powers of X modulo $(X^N + 1)$, and $X^{\alpha+N} \equiv -X^{\alpha} mod[X^N + 1]$.

3 Functional-bootstrapping-defined boolean operators

TFHE was initially presented in [CGGI16] as a bit-oriented cryptosystem. The homomorphic operations in this setting are limited to boolean operations (homomorphic XOR and homomorphic AND). Addition and multiplication of integers are defined using their respective binary circuits, and a bootstrapping is performed after every boolean gate.

Representing the internal state of a stream-cipher as an array of ciphertexts encrypting two bits or more allows the computation of the round function of a given stream-cipher with less bootstrappings compared to the binary representation (where each bit is encrypted separately). A non-realistic, yet compelling illustration, is to represent the entire internal state as a single ciphertext, and to encode the round function of a stream-cipher in a test vector as a $\mathbb{Z}_q \rightarrow \mathbb{Z}_q$ function, where q is a large enough integer. Doing so, a single functional bootstrapping would be required to update the internal state. Unfortunately, achieving this is not practical in real-world scenarios, as it requires enormous parameters for TFHE, resulting in computation times that are not realistic.

Our work aims at reducing the number of ciphertexts encrypting the internal state of a stream-cipher, and consequently reduces the number of necessary bootstrappings per round of the stream-cipher. Boolean operators in higher bases (i.e., non-binary bases) have to be redefined using Look-Up Tables (LUTs), which perfectly complies with the functional bootstrapping supported by TFHE.

[8] Negacyclic functions are antiperiodic functions over $\mathbb{T}$ with period $\frac{1}{2}$, i.e., $f(x) = -f(x + \frac{1}{2})$.

3.1 Bitwise operators

We provide a generic approach to evaluate any $\mathbb{Z}_B^2 \to \mathbb{Z}_B$ boolean function over integer ciphertexts with $\mathcal{M} = \mathbb{Z}_B$, using TFHE functional bootstrapping and [GBA21] chaining idea.

We denote by Op the logic gate that we want to evaluate. We set:

$$g_i^{Op} : \begin{array}{rcl} \mathbb{Z}_B & \to & \mathbb{Z}_B \\ x & \mapsto & Op(x,i) \end{array}$$

and $G_i^{Op}(X) = \sum_{k=0}^{B-1} g_i^{Op}(k) X^k$ its polynomial representation.

The test vector of our functional bootstrapping is then constructed from the polynomial representation of the lookup table which is of degree $B^2 - 1$:

$$v(X) = \sum_{i=0}^{B-1} G_i^{Op}(X) \cdot X^{iB} \tag{4}$$

For two values x and $y \in [\![0, B-1]\!]$, we observe that the $(Bx+y)$-th coefficient of v is equal to $Op(x,y)$.

The test vector TV_B^{Op} of degree $(N-1)$ used in the corresponding functional bootstrapping procedure is constructed from v by repeating each coefficient $\frac{N}{B^2}$ times, as follows:

$$TV_B^{Op}(X) = \sum_{i=0}^{B} \sum_{j=0}^{B} \sum_{k=0}^{\frac{N}{B^2}-1} g_i^{Op}(j) X^{k + j \cdot \frac{N}{B^2} + i \cdot \frac{N}{B}} \tag{5}$$

Once the test vector constructed, running a bootstrapping on the TLWE encryption of the linear combination of $(Bx+y)$ and the input TRLWE ciphertext from TV_B^{Op}, outputs a TRLWE ciphertext encryption of $Op(x,y)$.

Fact. Let's consider two TLWE ciphertexts c_1 and c_2 encrypting respectively the integer values a and $b \in \mathbb{Z}_B$. Calling TFHE gate-bootstrapping on the TLWE sample $c_1 B + c_2$, with a test vector constructed from v as in Equation 5 outputs a TLWE sample encrypting the value $Op(a,b)$ with probability[9]:

$$\mathbb{P}(success) = \mathbb{P}(Err(c_1 B + c_2) \leq \frac{1}{4B^2}) = \mathrm{erf}(\frac{1}{4B^2 \sqrt{B^2 V_{c_1} + V_{c_2} + V_{rounding}} \cdot \sqrt{2}})$$

where:

- $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ is the Gaussian error function.
- V_{c_1} is the variance of the error of c_1.
- V_{c_2} is the variance of the error of c_2.
- $V_{rounding} = \frac{n+1}{48N^2}$ is the variance of the rounding operation after re-scaling in the bootstrapping algorithm (Line 2 of algorithm 1).

[9] Please refer to [CZB+22] for a complete description on how to compute this probability.

3.2 Example: AND gate with $B = 4$

We consider a message m in $[\![0, 3]\!]$ as the concatenation of two bits thanks to its binary decomposition. We describe in this example how to compute a bitwise `AND` gate between the ciphertexts c_1 and c_2 encrypting 2 messages m_1 and m_2.

We refer by $\texttt{AND}_i$ to the truth table of the bitwise `AND` gate between a message m and i. Thus, we get:

$$\texttt{AND}_0 = (0, 0, 0, 0),\ \texttt{AND}_1 = (0, 1, 0, 1),\ \texttt{AND}_2 = (0, 0, 2, 2),\ \texttt{AND}_3 = (0, 1, 2, 3).$$

Then, we build a test vector $TV_4^{\texttt{AND}}$, as described in Equation 5, from the polynomial

$$v(X) = X^5 + X^7 + 2X^{10} + 2X^{11} + X^{13} + 2X^{14} + 3X^{15}$$

Finally, we get, by applying a functional bootstrapping to $(c_1 + B \cdot c_2)$ with $TV_4^{\texttt{AND}}$ as a test vector, an encryption of $\texttt{AND}(m_1, m_2)$ by selecting the element $\frac{N}{B^2} \cdot (m_1 + B \cdot m_2 + e)$ of the test vector where e is an error term.

Note that a similar procedure can be applied to any bitwise operator.

3.3 Byte shifts

An 8-bit implementation of stream-cipher often requires the bytes of the internal state to be shifted left or right in order to align the bits to be operated according to the encryption/decryption specifications. When representing the internal state as base 2 ciphertexts, a byte shift is a literal shift (simply consisting of memory access and copy operations). However, when a byte is represented as k ciphertexts encrypting base B integers, operating a shift with a position that is not a multiple of $\log_2(B)$ will have the effect of propagating a *carry* to the next digit, and will require additional computation of the non-linear function that provides both the output of the shift from the digit and the carry to propagate to the next one. The best way to compute non-linear functions in TFHE is to use a functional bootstrapping. Hence, our technique performs $\log_2(B)$-times less bootstrappings to compute `AND`, `OR`, and `XOR` operations, but an additional number of bootstrappings is required to perform the shifts. Despite this drawback, our method succeeds in significantly reducing the overall number of bootstrappings. Table 1 gives an estimation of the number of bootstrappings per round for an 8-bits implementation of Grain128-AEAD in different bases representations.

Base 4 A base 4 ciphertext representation of a byte is composed of four digits $b = (d_0, d_1, d_2, d_3)$, encrypted as $c = (\mathrm{TLWE}_{sk}(d_0), \mathrm{TLWE}_{sk}(d_1), \mathrm{TLWE}_{sk}(d_2), \mathrm{TLWE}_{sk}(d_3))$. Shifting by an odd position $r = 2s + 1$ is performed as s direct shifts, followed by a shift by one position that implies a functional bootstrapping-based algorithm.

We would like to extract two elements from every digit in a sequential fashion: (1) the result of the shift on the current digit $res(d_i)$, and (2) the carry it will propagate to the next digit $carry(d_i)$. The carry cannot be obtained using a linear function, so we extract it by using a functional bootstrapping corresponding to the following look-up tables (LUTs):

$$v^+ = [0,0,1,1] \text{ if } r > 0$$
$$v^- = [0,1,0,1] \text{ if } r < 0$$

The TRLWE test vectors $TV^+(X)$ and $TV^-(X)$ of degree $(N-1)$ are afterwards generated by using the same method as in Equation 5 for v^+ and v^-, respectively.

Once the carry from digit d_i is obtained, the next element to extact is the result of the shift over d_i, which is equal to $2 \cdot d_i \mod 4$. TFHE provides a natural modulo $|\mathcal{M}|$ operation due to the circular nature of the torus, but any other homomorphic evaluation of a modulo with a different moduli smaller than $|\mathcal{M}|$ would require extra bootstrapping. In our case $|\mathcal{M}| = 2B^2 = 32$. However we observed that the required value can be obtained using the linear combination $res(d_i) = (2 \cdot d_i - 4 \cdot carry(d_i)) + carry(d_{i-1})$ if $r > 0$, and $res(d_i) = (2 \cdot d_i - 4 \cdot carry(d_i)) + carry(d_{i+1})$ otherwise. This provides a byte shift operation at the cost of (at most) 3 bootstrappings, and four linear of combinations. These steps are summarized in Algorithm 2. Note that the inserted s digits (at the beginning or at the end depending on the sign of r) are noiseless-trivial samples of 0 [10]. Thus, in subsequent computations, when evaluating a gate with an operand equal to a noiseless-trivial sample of 0, no bootstrapping is needed [11].

Base 16 A base 16 ciphertext representation of a byte is composed of two digits. $b = (d_0, d_1)$. The carry extraction only applies to d_0. To shift by a positive position $pos = 4s + r$, we perform s literal ciphertexts shifts (free in terms of bootstrappings) followed by $0 \leq r \leq 3$ shifts involving carry-handling operations.

The three possible values for r are handled using three lookup-tables C-LUT$_1^+$, C-LUT$_2^+$, and C-LUT$_3^+$ such that: C-LUT$_i^+[d]$ stores the carry value that corresponds to shifting the digit d by i positions. Once the carry is obtained from d, we run a second bootstrapping to extract the resulting ciphertext from the operation $(d >> r)$ using the same approach. Three look-up tables S-LUT$_1^+$ S-LUT$_2^+$ S-LUT$_3^+$ such that S-LUT$_i^+[d] = (d >> i)$ are used to extract the resulting digit from the shift operation. The carry is afterwards propagated to d_1 followed by an evaluation of the same shift lookup table. Therefore, (at most) three calls to TFHE functional bootstrappings are needed to shift a byte ciphertext representation in the internal state. Following the same reasoning as with base 4 byte shifts, in the case where $pos > 4$, $\lceil \frac{pos}{4} \rceil$ noiseless-trivial encryptions of 0 are inserted at the first digits resulting in bootstrapping-free operations over these digits in subsequent operations. The same mechanism is applied for backward

[10] $c = (\boldsymbol{a}, b)$ is a noiseless-trivial encryption of 0 if $a_i = 0 \ \forall i \in [n]$ and $b = 0$. In other words, it represents an encoding of 0 as TLWE or TRLWE sample.

[11] The output of `OR` and `XOR` gates when one of the two operands is a noiseless trivial sample of 0 is equal to the other operand, while the output of an `AND` gate is a noiseless-trivial encryption of 0. The same stands for subsequent shifts of such samples, both the carry and the results are set to noiseless trivial samples of 0.

Algorithm 2 Byte shift with base 4

```
Input: 4 TLWE samples encrypting a byte $b$ in little-endian representation [12] $c = (c_0, c_1, c_2, c_3)$, an odd position $r = 2s + 1$, and a bootstrapping key $BK_{\boldsymbol{s} \to \boldsymbol{s'}} = (BK_i \in \mathrm{TRGSW}_{S'}(s_i))_{i \in [\![1,n]\!]}$ where $S'$ is the TRLWE interpretation of a secret key $\boldsymbol{s'}$
Output: 4 TLWE samples ($\hat{c_0}$, $\hat{c_1}$, $\hat{c_2}$, $\hat{c_3}$) encrypting $b$ shifted by $r$ positions in little-endian representation.
 1: if $r > 0$ then                                                    ▷ right shift
 2:     for i = 0 to (3 - s) do
 3:         $\hat{c}_{i+s} = c_i$
 4:     carry($c_0$) = FunctionalBootstrapping($c_0$, $TV^+(X)$, $BK$)
 5:     $\hat{c_0} = 2 \cdot c_0 - 4 \cdot carry(c_0)$
 6:     for  i = 1 to 2 do
 7:         carry($c_i$) = FunctionalBootstrapping($c_i$, $TV^+(X)$, $BK$)
 8:         $\hat{c_i} = (2 \cdot c_i - 4 \cdot carry(c_i)) + carry(c_{i-1})$
 9:     carry($c_3$) = FunctionalBootstrapping($c_3$, $TV^+(X)$, $BK$)
10:     $\hat{c_3} = 2 \cdot c3 - 4 \cdot carry(c_3)$
11: if $r < 0$ then                                                    ▷ left shift
12:     for i = 0 to (3 - s) do
13:         $\hat{c}_i = c_{i+s}$
14:     carry($c_3$) = FunctionalBootstrapping($c_3$, $TV^-(X)$, $BK$)
15:     $\hat{c_3} = 2 \cdot c_3 - 4 \cdot carry(c_3)$
16:     for  i = 2 to 1 do
17:         carry($c_i$) = FunctionalBootstrapping($c_i$, $TV^-(X)$, $BK$)
18:         $\hat{c_i} = (2 \cdot c_i - 4 \cdot carry(c_i)) + carry(c_{i+1})$
19:     carry($c_0$) = FunctionalBootstrapping($c_0$, $TV^-(X)$, $BK$)
20:     $\hat{c_0} = 2 \cdot c_0 - 4 \cdot carry(c_0)$
    return ($\hat{c_0}$, $\hat{c_1}$, $\hat{c_2}$, $\hat{c_3}$)
```

byte shifts with respective S-LUT$_i^-$ and C-LUT$_i^-$ lookup tables. Algorithm 3 describes this procedure.

3.4 Stream-cipher adaptation

Representing the internal state of a stream-cipher as base B integers will have the effect of producing the keystream as base B integers as well, but the plaintext space with respect to the TFHE encryption layer is of size $2B^2$. The encryption (resp. decryption) process must then be slightly adapted. In order to take advantage from the fact that the FHE encrypted keystream is added to a symmetrically encrypted data, which is a plaintext-ciphertext operation with respect to the FHE encryption layer, and such operations do not increase the noise. Thus, no bootstrapping is required in this step of transciphering.

Instead of the classical `XOR` between the keystream and the message (resp. ciphertext), one makes an addition (respectively substraction) modulo $2B^2$ [14].

[14] The modulo operation is never performed per se since the sum of two elements of $\mathbb{Z}_B$ is at most equal to $2(B-1)$.

Algorithm 3 Byte shift with base 16

```
Input: 2 TLWE samples encrypting a byte $b$ in little-endian representation [13]
        $c = (c_0, c_1)$, a position $r = 4s + q$ with $|r| < 8$
        a bootstrapping key $BK_{\boldsymbol{s}\to\boldsymbol{s'}} = (BK_i \in \mathrm{TRGSW}_{S'}(s_i))_{i\in[\![1,n]\!]}$ where $S'$ is the
    TRLWE interpretation of a secret key $\boldsymbol{s'}$
Output: 2 TLWE samples $(\hat{c_0}, \hat{c_1})$ encrypting $b$ shifted by $r$ positions in little-endian
    representation.
 1: if $r > 0$ then                                                        ▷ right shift
 2:     if s == 1 then
 3:         $\hat{c}_0 \leftarrow$ Noiseless Trivial sample of 0
 4:         $\hat{c}_1$ = FunctionalBootstrapping($c_0$, S-LUT$_q^+(X)$, $BK$)
 5:     else
 6:         carry($c_0$) = FunctionalBootstrapping($c_0$, C-LUT$_q^+(X)$, $BK$)
 7:         $\hat{c}_0$ = FunctionalBootstrapping($c_0$, S-LUT$_q^+(X)$, $BK$)
 8:         $\hat{c}_1$ = FunctionalBootstrapping($c_1$, S-LUT$_q^+(X)$, $BK$)
 9:         $\hat{c}_1 +=$ carry($c_0$)
10: if $r < 0$ then                                                         ▷ left shift
11:     if s == 1 then
12:         $\hat{c}_1 \leftarrow$ Noiseless Trivial sample of 0
13:         $\hat{c}_0$ = FunctionalBootstrapping($c_0$, S-LUT$_q^-(X)$, $BK$)
14:     else
15:         carry($c_1$) = FunctionalBootstrapping($c_1$, C-LUT$_q^-(X)$, $BK$)
16:         $\hat{c}_1$ = FunctionalBootstrapping($c_1$, S-LUT$_q^-(X)$, $BK$)
17:         $\hat{c}_0$ = FunctionalBootstrapping($c_0$, S-LUT$_q^-(X)$, $BK$)
18:         $\hat{c}_0 +=$ carry($c_1$)
    return $(\hat{c_0}, \hat{c_1})$
```

Indeed, it would require a functional bootstrapping to evaluate a modulo B addition of plaintext-ciphertext elements when the ciphertext lies in a greater set than $\mathbb{Z}_B$ (the keystream digits). This modification of the encryption/decryption process has no effect whatsoever on the security of the stream-cipher.

Plaintext space size limitations. The bootstrapping operation as described in [BMMP18] stores all the elements of a plaintext space $\mathcal{M}$ as coefficients of the test vector. The potential lack of precision in the `BlindRotate` operation due to the noise of the input ciphertext is solved, up to a certain level, thanks to the redundancy of the coefficients in the test vector. Indeed, in practice, $|\mathcal{M}|$ divides N so every element in $\mathcal{M}$ is repeated consecutively $\frac{N}{|\mathcal{M}|}$ times in the TRLWE polynomial which will be rotated. Therefore, if the input ciphertext's noise and the rounding error make the `BlindRotate` operation fail to bring the target coefficient of the test vector to the constant term, then the added redundancy ensures that any adjacent coefficient by at most $\frac{N}{2|\mathcal{M}|} - 1$ positions, to the left or right, according to the sign of the error, is still equal to the target coefficient. Increasing the size of $\mathcal{M}$ reduces the redundancy factor, and thus, the precision of the (functional) bootstrapping. Increasing N allows to proportionally increase the size of $\mathcal{M}$ without loss of precision in the bootstrapping. However, this solution also increases the size of a TRLWE ciphertext, and the runtime of the bootstrapping operation, which is quasi-linear in the degree of the TRLWE polynomial encoding the test vector. A challenging task is to find the right balance between the size of $\mathcal{M}$, the desired precision, and a parameter set for TFHE which provides a reasonable ciphertext size, in order to still ensure a significant speedup on the server-end. To that end, we provide a set of three parameters-set for TFHE presented in Table 1.

4 Grain128-AEAD

Grain128-AEAD [HJM+21] is a widely-used stream-cipher that draws inspiration from Grain128a [ÅHJM11]. It is a finalist in the NIST competition on lightweight cryptography [15] and features slight modifications from its predecessor. Grain128-AEAD has an Authenticated Encryption with Associated Data (AEAD) mode, which allows for the encryption of a subset of plaintext bits using a mask d with the formula $c_i = m_i \oplus (ks_i \cdot d_i)$. Additionally, a larger 64-bit MAC is computed on the encrypted data. The internal state of Grain128-AEAD is 256 bits in length and consists of a 128-bit Non-linear Feedback Shift Register (NFSR) and a 128-bit Linear Feedback Shift Register (LFSR), along with two 64-bit accumulator and shift registers for MAC computation.

Once warmed up with 384 rounds, Grain128-AEAD can generate two streams of bits, namely the encryption keystream (ks) and the MAC keystream (ms), which are extracted from the main keystream using bit parity. Specifically, ks_i is calculated as y_{384+2i} and ms_i is calculated as $y_{384+2i+1}$, where y_{383} represents the last output bit from the warm-up phase of the cipher.

5 Experimental results

We ran *single core* performance tests on an 12th Gen Intel(R) Core(TM) i7-12700H v6 @ 2.60GHz and 22GB RAM.

Table 1 shows the performance of transciphering after applying a base $B \geq 2$ representation of the internal state of Grain128-AEAD. The performance metrics are the number of bootstrappings needed to homomorphically evaluate the warm-up phase of Grain128-AEAD. The chosen set of parameters impacts the runtime of a single bootstrapping operation, and therefore the runtime of entire the warm-up operation.

With every parameter-set we aim at finding a good balance between the bootstrapping's runtime and its accuracy, under the constraint of a security level of $\lambda = 128$ bits, and a circuit accuracy of $1-2^{-32}$. The runtime of the bootstrapping operation is quasi-linear in the TRLWE polynomial degree N as well as linear in the TLWE dimension n and the gadget decomposition parameter l [CGGI18]. As such, we aim to minimize these parameters for better performances. The security of the scheme is linked to the parameters n, N, and the standard deviations

[15] https://csrc.nist.gov/Projects/lightweight-cryptography

used to define the noise of TLWE ciphertexts and TRLWE ciphertexts. We use lattice-estimator [APS15] to find secure sets of parameters for relatively small values of n and N. The other parameters are chosen to optimize correctness of computations under the chosen security. The selected sets of parameters are presented in Table 1.

It is clear from Table 1 that our base 2 implementation slightly outperforms base 2 implementations from [BBS21] (around 10% with equivalent computational resources) thanks to the introduced optimization where noiseless trivial samples are inserted when a byte-shift is performed, resulting in bootstrapping-free operations. The base 4 (4 ciphertexts) implementation results in a noticeable speedup compared to base 2: 20% less bootstrappings, and 10% faster due to larger parameters, while base 16 provides a significant improvement of almost 65% less bootstrappings compared to [BBS21]. However, base 16 warm-up time is bigger than base 2 or base 4 times due to the use of huge parameters in order to respect the fixed security level ($\lambda = 128$) and the bound on the error rate (2^{-32}).

Table 1. Summary of the performance results.

		Base		
		2	4	16
TFHE Params	N	1024	2048	65536
	n	595	740	930
	$log_2(B_g)$	5	11	32
	l	4	3	1
	keyswitch standard-dev.	$1.26e^{-4}$	$9.17e^{-6}$	$3e^{-7}$
	bootstrapping standard-dev.	$5.6e^{-8}$	$9.6e^{-11}$	$1.0e^{-100}$
	security level λ	128	128	128
Runtime	# of bootstrappings (warm-up circuit)	18912	16608	7718
	runtime (a single bootstrapping)	$13ms$	$17ms$	$92ms$
	runtime (warm-up circuit)	$4.80min$	$3.98min$	$11.83min$
Error estimation	error probability (a single bootstrapping)	2^{-47}	2^{-46}	2^{-45}
	error probability (warm-up circuit)	2^{-31}	2^{-32}	$2^{-31.5}$

6 Conclusion

In this work we revisit transciphering with stream-ciphers using TFHE's functional bootstrapping by specifying a set of operators with inputs represented by digits in base B. Despite the small adaptations that have to be taken into account when using this technique, regarding plaintext-space size, the decryption procedure, and the careful choice of FHE parameters, these operators can be readily employed to implement a wide range of stream-ciphers, and often result in non-negligible improvements regarding the required number of bootstrappings as shown with a standard encryption algorithm Grain128-AEAD.

While the need for larger parameters in larger bases increases the runtime for a single bootstrapping operation, this work also aims at leveraging cloud hardware acceleration modules for FHE [BBTV23, MAM20, WSKB22] which enables efficient FHE (LWE, RLWE, and RGSW) operations with a significantly better accommodation of larger parameter-sets than software implementation.

We plan to extend this approach to the homomorphic evaluation of block-ciphers, and other cryptographic primitives such as hash functions as investigated in [BSS+23]. Subsequent works involve implementing non-homogeneous base representation of a byte, using two digits in base 4 and a single digit in base 8. This could potentially improve the accuracy/number of ciphertexts trade-off, and showcase the versatility of encrypted byte representations, not only for transciphering-related applications, but also for other use-cases of FHE. A further optimization would be to take advantage of the symmetric aspect of the lookup tables (from the fact that boolean operators are commutative) to reduce the plaintext space size from $2B^2$ to B^2 by providing a more compact way to homomorphically reference elements inside the boolean operations lookup tables.

References

ÅHJM11. Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of grain-128 with optional authentication. *IJWMC*, 5:48–59, 2011.

APS15. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. Cryptology ePrint Archive, Paper 2015/046, 2015. https://eprint.iacr.org/2015/046.

ARS+15. Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT*, pages 430–454, 2015.

BBS21. Adda Akram Bendoukha, Aymen Boudguiga, and Renaud Sirdey. Revisiting stream-cipher-based homomorphic transciphering in the tfhe era. 14-th International Symposium of Foundation and Practice of Security, 2021.

BBS22. Adda-Akram Bendoukha, Aymen Boudguiga, and Renaud Sirdey. Revisiting stream-cipher-based homomorphic transciphering in the tfhe era. In Esma Aïmeur, Maryline Laurent, Reda Yaich, Benoît Dupont, and Joaquin Garcia-Alfaro, editors, *Foundations and Practice of Security*, pages 19–33, Cham, 2022. Springer International Publishing.

BBTV23. Jonas Bertels, Michiel Van Beirendonck, Furkan Turan, and Ingrid Verbauwhede. Hardware acceleration of fhew. Cryptology ePrint Archive, Paper 2023/618, 2023. https://eprint.iacr.org/2023/618.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

BLSK19. Aymen Boudguiga, Jerome Letailleur, Renaud Sirdey, and Witold Klaudel. Enhancing CAN security by means of lightweight stream-ciphers and protocols. In Alexander B. Romanovsky, Elena Troubitsyna, Ilir Gashi, Erwin Schoitsch, and Friedemann Bitsch, editors, *Computer Safety, Reliability,* *and Security - SAFECOMP 2019 Workshops, ASSURE, DECSoS, SASSUR, STRIVE, and WAISE, Turku, Finland, September 10, 2019, Proceedings*, volume 11699 of *Lecture Notes in Computer Science*, pages 235–250. Springer, 2019.

BMMP18. Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Advances in Cryptology – CRYPTO 2018*, volume 10993 of *Lecture Notes in Computer Science*, pages 483–512. Springer, 2018.

BP12. Joan Boyar and René Peralta. A small depth-16 circuit for the aes s-box. In Dimitris Gritzalis, Steven Furnell, and Marianthi Theoharidou, editors, *Information Security and Privacy Research*, pages 287–298, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

BSS$^+$23. Adda Akram Bendoukha, Oana Stan, Renaud Sirdey, Nicolas Quero, and Luciano Freitas. Practical homomorphic evaluation of block-cipher-based hash functions with applications. In Guy-Vincent Jourdan, Laurent Mounier, Carlisle Adams, Florence Sèdes, and Joaquin Garcia-Alfaro, editors, *Foundations and Practice of Security*, pages 88–103, Cham, 2023. Springer Nature Switzerland.

CCF$^+$18a. 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.

CCF$^+$18b. 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, 01 2018.

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*, pages 3–33, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

CGGI18. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: Fast fully homomorphic encryption over the torus. Cryptology ePrint Archive, Report 2018/421, 2018. https://eprint.iacr.org/2018/421.

CHK$^+$20. Jihoon Cho, Jincheol Ha, Seongkwang Kim, Byeonghak Lee, Joohee Lee, Jooyoung Lee, Dukjae Moon, and Hyojin Yoon. Transciphering framework for approximate homomorphic encryption (full version). Cryptology ePrint Archive, Paper 2020/1335, 2020. https://eprint.iacr.org/2020/1335.

CJP21. Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In Shlomi Dolev, Oded Margalit, Benny Pinkas, and Alexander Schwarzmann, editors, *Cyber Security Cryptography and Machine Learning*, pages 1–19, Cham, 2021. Springer International Publishing.

CKKS16. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Report 2016/421, 2016. https://eprint.iacr.org/2016/421.

CLOT21. Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe. Cryptology ePrint Archive, Report 2021/729, 2021. https://ia.cr/2021/729.

CM19. Claude Carlet and Pierrick Méaux. Boolean functions for homomorphic-friendly stream ciphers. Cryptology ePrint Archive, Paper 2019/1446, 2019. https://eprint.iacr.org/2019/1446.

CZB$^+$22. Pierre-Emmanuel Clet, Martin Zuber, Aymen Boudguiga, Renaud Sirdey, and Cédric Gouy-Pailler. Putting up the swiss army knife of homomorphic calculations by means of tfhe functional bootstrapping. Cryptology ePrint Archive, Paper 2022/149, 2022. https://eprint.iacr.org/2022/149.

DGH$^+$21. Christoph Dobraunig, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Pasta: A case for hybrid homomorphic encryption. *Cryptology ePrint Archive*, 2021.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://eprint.iacr.org/2012/144.

GBA21. Antonio Guimarães, Edson Borin, and Diego F. Aranha. Revisiting the functional bootstrap in tfhe. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):229–253, Feb. 2021.

GHS12a. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the aes circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 850–867, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

GHS12b. Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the aes circuit. In *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 850–867. Springer, 2012.

GSW13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

HJM$^+$21. Martin Hell, Thomas Johansson, Alexander Maximov, Willi Meier, and Hirotaka Yoshida. Grain-128aeadv2: Strengthening the initialization against key reconstruction. Cryptology ePrint Archive, Report 2021/751, 2021. https://ia.cr/2021/751.

HJMM06. Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. A stream cipher proposal: Grain-128. In *2006 IEEE International Symposium on Information Theory*, pages 1614–1618, 2006.

HMR20. Clément Hoffmann, Pierrick Méaux, and Thomas Ricosset. Transciphering, using filip and tfhe for an efficient delegation of computation. Cryptology ePrint Archive, Paper 2020/1373, 2020. https://eprint.iacr.org/2020/1373.

KS21. Kamil Kluczniak and Leonard Schild. Fdfb: Full domain functional bootstrapping towards practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2021/1135, 2021. https://ia.cr/2021/1135.

KS22. Kamil Kluczniak and Leonard Schild. Fdfb: Full domain functional bootstrapping towards practical fully homomorphic encryption. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(1):501–537, Nov. 2022.

MAM20. Toufique Morshed, Md Momin Al Aziz, and Noman Mohammed. Cpu and gpu accelerated fully homomorphic encryption. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 142–153, 2020.

OKC20. Hiroki Okada, Shinsaku Kiyomoto, and Carlos Cid. Integerwise functional bootstrapping on tfhe. In Willy Susilo, Robert H. Deng, Fuchun Guo, Yannan Li, and Rolly Intan, editors, *Information Security*, pages 107–125, Cham, 2020. Springer International Publishing.

WSKB22. Zhehong Wang, Dennis Sylvester, Hun-Seok Kim, and David Blaauw. Hardware acceleration for third-generation fhe and psi based on it, 2022.

YXS$^+$21. Zhaomin Yang, Xiang Xie, Huajie Shen, Shiying Chen, and Jun Zhou. Tota: Fully homomorphic encryption with smaller parameters and stronger security. Cryptology ePrint Archive, Report 2021/1347, 2021. https://ia.cr/2021/1347.