Revisiting the Slot-to-Coefficient Transformation for BGV and BFV

Robin Geelen

COSIC, KU Leuven, Leuven, Belgium

Abstract. Numerous algorithms in homomorphic encryption require an operation that moves the slots of a ciphertext to the coefficients of a different ciphertext. We describe an FFT-like method for decomposing this slot-to-coefficient transformation (and its inverse) for BGV and BFV. The proposed method is specific to power-of-two cyclotomic rings and can handle both fully and sparsely packed slots. Previously, such a method was only known for non-power-of-two cyclotomic rings.
Our algorithm admits more freedom in the complexity-depth trade-off than prior works. Moreover, it brings down the computational complexity of the slot-to-coefficient transformation from a linear to a logarithmic number of FHE operations in the best case, which is shown via a detailed complexity analysis. We also provide a proof-of-concept implementation in the Magma bootstrapping library.

Keywords: Homomorphic encryption · Linear transformations · BGV · BFV

1 Introduction

Several fully homomorphic encryption (FHE) schemes offer the capability of encoding multiple numbers in a ciphertext. This functionality of "packing" multiple numbers together is referred to as *batching*, and each entry in the packed vector is called a *plaintext slot*. For example, the closely related BGV [BGV14] and BFV [Bra12, FV12] schemes encode a vector of numbers defined modulo a power of a prime. Similarly, the CKKS [CKKS17] scheme encodes a vector of complex numbers approximated up to a limited precision. A recent attempt was made to introduce batching in third generation schemes as well [LW23a]. However, those techniques remain currently only of theoretical interest and will not be considered in the rest of this paper.

A very common operation in both BGV/BFV and CKKS is converting between slot and coefficient representation. Informally, the slot-to-coefficient transformation is defined as follows: given one or multiple ciphertexts encoding $m_0, m_1, \ldots, m_{N-1}$ in the plaintext slots, compute a ciphertext that encrypts the polynomial $m_0 + m_1 \cdot X + \ldots + m_{N-1} \cdot X^{N-1}$. The most important applications of the slot-to-coefficient transformation include (amortized) bootstrapping [GHS12a, AP13, HS21, CH18, GV23, GIKV23, OPP23, CHK+18, LW23b], scheme conversion [BGGJ20, LHH+21, BCK+23] and transciphering [CHK+21]. The inverse operation (coefficient-to-slot transformation) also appears in these applications.

The slot-to-coefficient transformation and its inverse are linear operations. Several methods exist to compute it homomorphically, each of which is useful in a particular setting. For the CKKS scheme, one typically takes the number of messages N as a power of two, and efficient algorithms have been proposed to compute the slot-to-coefficient transformation in that setting [CCS19, HHC19]. For the BGV and BFV schemes, there are two common options: one can take the number of messages N different from a power of two, which is the direction taken by HElib. In that case, an efficient method exists to

E-mail: robin.geelen@esat.kuleuven.be (Robin Geelen)

compute the slot-to-coefficient transformation [HS18, HS21]. Alternatively, one can take the number of messages as a power of two, in which case no efficient algorithm for the slot-to-coefficient transformation has been proposed to the best of our knowledge.

The main focus of this paper is studying the slot-to-coefficient transformation for BGV and BFV in the power-of-two setting. Similarly to the CKKS case, we propose an FFT-like algorithm to decompose the transformation in multiple stages, which scales well even for a large number of plaintext slots. The algorithm is added to the open source Magma bootstrapping library [GV23]. An operation count is provided at the end of the paper to show that the proposed method has an advantage over prior work.

1.1 Related Work

Research in the slot-to-coefficient transformation can be divided into two categories: on the one hand, several works bring down the computational complexity of generic linear transformations. Many of the techniques on this front are shared between the BGV/BFV and CKKS case. On the other hand, there also exist algorithms to decompose the slot-to-coefficient map into smaller linear transformations, each of which can then be evaluated with the generic approach. Below we give an extensive overview of the existing literature for BGV/BFV and CKKS.

1.1.1 Linear Transformations in BGV and BFV

Generic linear transformations. HElib contains several algorithms to evaluate generic homomorphic linear transformations [HS18, CCLS19]. The following two techniques are used: a linear transformation, which is a sum of D weighted "rotations", can be rewritten as a double summation. By taking $O(\sqrt{D})$ terms in both the inner and outer sum, one can reduce the number of rotations to $O(\sqrt{D})$. This algorithm is called a *baby-step/giant-step* implementation. Moreover, the so-called *hoisting* technique can be used to simultaneously compute all inner-sum rotations, which is more efficient than computing each of them separately. Technical details about both techniques are given in Section 2.4.

FFT-like decomposition. As mentioned earlier, the slot-to-coefficient transformation in HElib (also called the *evaluation map*) employs non-power-of-two message packing [HS21]. More specifically, one chooses a cyclotomic index m and then packs $N = \varphi(m)$ messages where $\varphi(\cdot)$ denotes Euler's totient function. The parameter m is typically chosen as a product of smaller prime powers for two reasons: it gives reasonably high packing capacity of $\mathbb{Z}_{p^e}$-vectors and allows FFT-like decomposition of the slot-to-coefficient transformation. However, this parameter setting also has significant disadvantages compared to the power-of-two case, including less efficient implementations and a larger noise variance of the input ciphertext during bootstrapping (the exact noise variance depends on the number of distinct prime factors in m, following the heuristic analysis of Halevi and Shoup [HS21]).

From the theoretical side, it is known that the slot-to-coefficient transformation can be evaluated in quasilinear time [AP13]. Note that this early work employed a "ring switching" technique to convert between different values of the cyclotomic index m, which was necessary to reach an efficient decomposition of the linear transformation. However, subsequent implementations of bootstrapping (including the one in HElib) do not use ring switching anymore, because it is hypothesized that ring switching would not give a substantial performance benefit in practice [HS21].

Finally, we note that no FFT-like algorithm has yet been proposed for the slot-to-coefficient transformation in the case of power-of-two cyclotomics. To the best of our knowledge, the only approach is the one from SEAL [CH18], but it does not use an efficient decomposition into smaller-dimensional matrices.

1.1.2 Linear Transformations in CKKS

Generic linear transformations. As mentioned earlier, the strategy to evaluate generic linear transformations in CKKS is very similar to BGV/BFV (including optimizations such as baby-step/giant-step implementations and hoisting). Moreover, a *double-hoisting* method was proposed to accelerate the computation of the inner-sum rotations even more [BMTH21]. Notably, the double-hoisting technique carries over to BGV/BFV when hybrid key switching [KPZ21] is used.

FFT-like decomposition. The CKKS scheme is only used in combination with a power-of-two cyclotomic index m, which implies that $N = \varphi(m) = m/2$. Very similar FFT-like algorithms to decompose the slot-to-coefficient transformation were proposed by Chen et al. [CCS19] and by Han et al. [HHC19]. Compared to the BGV/BFV case, the CKKS transformations resemble much more a classical FFT algorithm due to exclusive use of power-of-two packing. Technical details about those algorithms are discussed in Section 2.3, where we approach the problem from the same point of view as Han et al.

1.1.3 Recent Progress in Bootstrapping

An important application of the slot-to-coefficient transformation is bootstrapping, and we expect that our methods will improve bootstrapping as well. More specifically, this section discusses the applicability of our methods to two very recent papers that extend the functionality of BGV and BFV bootstrapping:

- Kim et al. [KSS24] proposed a novel technique to reduce the noise of a BFV ciphertext by using CKKS bootstrapping as a subroutine. Unlike all prior works, their method is not restricted to small prime-power plaintext moduli. However, the amount of reduced noise in their algorithm depends heavily on the number of iterations in the META-BTS [BCC+22] high-precision bootstrapping technique. Our new algorithm is not applicable to the work of Kim et al. because it relies on CKKS.
- Ma et al. [MHWW24] proposed a novel technique to bootstrap a BGV ciphertext for large values of p more efficiently. Their idea is to optimize the digit extraction polynomials via local null polynomials modulo p^e, based on the observation that the noise is much smaller than p. Since power-of-two cyclotomics only give a good packing density for large values of p (see Section 3.1; up to complete splitting in linear factors), we expect that our methods will be very useful in this scenario.

Although both works were implemented for only one scheme (BGV or BFV), they can trivially bootstrap the other scheme as well, based on the very efficient BGV-to-BFV and BFV-to-BGV conversion technique [AP13]. Moreover, both methods have different advantages and disadvantages, so the best algorithm might depend on the exact setting. Finding the best method for given parameters would be interesting future research.

1.2 Contributions and Outline

This work makes the following contributions:

- Section 3 describes new properties of the automorphism group and plaintext packing for BGV/BFV. For a prime-power plaintext modulus p^e, we make a crucial difference between the situation $p = 1 \pmod 4$ (where the rotation group of the plaintext slots is two-dimensional) and $p = 3 \pmod 4$ (where the rotation group of the plaintext slots is one-dimensional). Sparsely packed slots are handled as a special case of fully packed slots by encoding messages in a subring.

Table 1: List of commonly used symbols and their meaning

Symbol	Meaning
m (or m')	Power-of-two cyclotomic index ($m \geq m'$)
N (or N')	Totient of cyclotomic index m (or m')
$\mathcal{R}$ (or $\mathcal{R}'$)	Cyclotomic ring of index m (or m')
E (or E')	Slot algebra for m (or m') and p^e
p^e	Plaintext modulus for BGV/BFV
q	Ciphertext modulus for BGV/BFV
d	Multiplicative order of p in $\mathbb{Z}_m^*$
ℓ	Number of slots (equals N/d)

- Section 4 describes a CKKS-like method to decompose the slot-to-coefficient transformation in BGV and BFV, which scales well even for a large number of slots. We provide a proof-of-concept implementation in the Magma bootstrapping library for BGV and BFV [GV23].[1] The implementation and results are discussed in Section 5.

2 Preliminaries

2.1 Notations

We will use power-of-two cyclotomic indices m and m', where m' divides m. Their totient is written as $N = \varphi(m) = m/2$ and $N' = \varphi(m') = m'/2$ respectively. The involved homomorphic encryption schemes work over the ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and its subring $\mathcal{R}' = \mathbb{Z}[X^{N/N'}]/(X^N + 1)$. For an integer $n \geq 2$, we write the quotient ring of $\mathcal{R}$ modulo n as $\mathcal{R}_n = \mathcal{R}/n\mathcal{R}$ and similarly for $\mathcal{R}'$. All ring elements are shown in bold lower case letters (e.g., $\boldsymbol{a} \in \mathcal{R}$) or explicitly as polynomials (e.g., $a(X) \in \mathcal{R}$). The infinity norm of $\boldsymbol{a} \in \mathcal{R}$ (i.e., its largest coefficient) is denoted by $||\boldsymbol{a}||_\infty$. The unit group of integers modulo m is written as $\mathbb{Z}_m^*$ (this is isomorphic to the automorphism group of $\mathcal{R}/\mathbb{Z}$). The subgroup generated by $g \in \mathbb{Z}_m^*$ is denoted by $\langle g \rangle$. Finally, we summarize commonly used symbols in Table 1 (some of these notations will be introduced in later sections).

2.2 BGV and BFV Encryption

BGV and BFV encrypt plaintexts from the ring $\mathcal{R}_{p^e}$ for some odd prime p and positive integer e. We will see the plaintext space $\mathcal{R}_{p^e}$ as a subset of $\mathcal{R}$ where polynomials have coefficients in $(-p^e/2, p^e/2) \cap \mathbb{Z}$. As mentioned above, we will only consider power-of-two-dimensional rings $\mathcal{R}$, although one can easily generalize encryption to arbitrary cyclotomic rings. A ciphertext is a pair of ring elements, i.e., it lives in $\mathcal{R}_q^2$. For correctness, the ciphertext modulus needs to be much greater than the plaintext modulus ($q \gg p^e$).

A BGV ciphertext $(\boldsymbol{c}_0, \boldsymbol{c}_1) \in \mathcal{R}_q^2$ is said to encrypt the plaintext $\boldsymbol{m} \in \mathcal{R}_{p^e}$ under secret key $\boldsymbol{s} \in \mathcal{R}$ if

$$\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} = \boldsymbol{m} + p^e \boldsymbol{e} \pmod{q}$$

for some noise term $\boldsymbol{e} \in \mathcal{R}$ that satisfies $||\boldsymbol{e}||_\infty \leq (q/p^e - 1)/2$. Similarly, a valid BFV ciphertext satisfies

$$\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s} = \lfloor (q/p^e) \cdot \boldsymbol{m} \rceil + \boldsymbol{e} \pmod{q},$$

where the bound on $\boldsymbol{e}$ is the same.

[1] See https://github.com/KULeuven-COSIC/Bootstrapping_BGV_BFV.

2.2.1 Homomorphic Operations in BGV and BFV

BGV and BFV offer the same set of homomorphic operations over the plaintext space and have the same asymptotic performance. Each homomorphic operation causes a certain amount of noise growth (the term $\boldsymbol{e}$ will grow larger). The following operations are defined:

- Addition: given two ciphertexts that encrypt $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$, compute a new ciphertext that encrypts $\boldsymbol{m}_1 + \boldsymbol{m}_2$. The noise growth of addition is additive (the new noise is roughly equal to the sum of the noise terms). Addition is generally considered a cheap operation in terms of execution time.
- Multiplication: given two ciphertexts that encrypt $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$, compute a new ciphertext that encrypts $\boldsymbol{m}_1 \cdot \boldsymbol{m}_2$. Roughly, one can bound the new noise after multiplication by $c \cdot (||\boldsymbol{e}_1||_\infty + ||\boldsymbol{e}_2||_\infty)$, where c is a constant that grows linearly in p^e and quadratically in N (so larger parameters result in more noise). In terms of execution time, multiplication is much more expensive than addition, because it requires an expensive post-processing step called key switching.
- Automorphism: given a ciphertext that encrypts $a(X)$ and given a number $j \in \mathbb{Z}_m^*$, compute a new ciphertext that encrypts $a(X^j)$ (note that reduction modulo $X^N + 1$ is implicit here). The automorphism $a(X) \mapsto a(X^j)$ will be denoted by τ_j hereafter. The noise growth of automorphism is additive (the new noise is equal to the old noise plus some extra constant term). In terms of execution time, the cost of automorphism is similar to multiplication as it also requires key switching.

Next to the ciphertext-ciphertext addition and multiplication as defined above, one can also define addition and multiplication between a plaintext and a ciphertext. Plaintext-ciphertext multiplication is much cheaper than ciphertext-ciphertext multiplication, because no key switching is required. For more information about the practical performance of the homomorphic operations, we refer to Kim et al. [KPZ21].

2.2.2 Plaintext Slots in BGV and BFV

Based on the Chinese remainder theorem, Smart and Vercauteren [SV14] proposed a method to pack multiple numbers in a ciphertext, and perform "single instruction, multiple data" operations on these numbers. The idea relies on the following lemma.

Lemma 1. *Let p be an odd prime number and let e be a positive integer as above. Let $m \geq 2$ be a power-of-two cyclotomic index and $N = m/2$. Then the m-th cyclotomic polynomial factors modulo p^e as*

$$X^N + 1 = F_1(X) \cdot \ldots \cdot F_\ell(X) \pmod{p^e}. \tag{1}$$

Each factor is of degree d, which is the multiplicative order of p modulo m, and the number of factors is $\ell = N/d$.

As such, using the Chinese remainder theorem, the plaintext ring is isomorphic to

$$\mathcal{R}_{p^e} \cong \mathbb{Z}_{p^e}[X]/(F_1(X)) \times \ldots \times \mathbb{Z}_{p^e}[X]/(F_\ell(X)),$$

where addition and multiplication at the right-hand side correspond to component-wise addition and multiplication. Since all rings $\mathbb{Z}_{p^e}[X]/(F_i(X))$ are Galois rings of the same parameters, they are isomorphic to each other. Therefore, we can see the plaintext space as ℓ copies of $\mathbb{Z}_{p^e}[X]/(F_1(X))$, each of which is called a *plaintext slot*.

Slot permutations. Gentry et al. [GHS12b] noticed that one can homomorphically apply permutations to the plaintext slots using the automorphisms from above. This is most easily seen as follows: first, define the *slot algebra* as $E = \mathbb{Z}_{p^e}[\zeta_m]$ with ζ_m a formal root

of $F_1(X)$, and let $S \subseteq \mathbb{Z}$ be a complete system of representatives for $\mathbb{Z}_m^*/\langle p \rangle$ (as done in HElib [HS18]). In practice, we construct the set S as

$$S = \{g_1^{e_1} \cdot \ldots \cdot g_t^{e_t} \mid 0 \leq e_i < \ell_i\}, \tag{2}$$

where $\ell = \ell_1 \cdot \ldots \cdot \ell_t$ is the number of slots.[2] The plaintext ring is now isomorphic to E^ℓ, which can be explicitly computed as

$$\mathcal{R}_{p^e} \to E^\ell \colon a(X) \mapsto \left\{a(\zeta_m^h)\right\}_{h \in S}. \tag{3}$$

As such, each plaintext slot corresponds to one index $h \in S$ or one tuple $(e_1, \ldots, e_t)$. By associating each plaintext slot with such a tuple, the full plaintext space corresponds to a t-dimensional hypercube [HS20].

To enable permutations on a plaintext $\boldsymbol{m}$, we take $0 \leq v < \ell_i$ and compute

$$\rho_i^v(\boldsymbol{m}) = \boldsymbol{\mu} \cdot \tau_j(\boldsymbol{m}) + (1 - \boldsymbol{\mu}) \cdot \tau_k(\boldsymbol{m}), \tag{4}$$

where $j = g_i^{-v} \pmod m$ and $k = g_i^{\ell_i - v} \pmod m$, and $\boldsymbol{\mu}$ is the "mask" obtained by embedding '0' in the plaintext slots with $e_i < v$, and embedding '1' in the other slots. Letting $\boldsymbol{m'} = \rho_i^v(\boldsymbol{m})$, it is easy to see that the value of $\boldsymbol{m'}$ in slot $(e_1, \ldots, e_i', \ldots, e_t)$ is equal to the value of $\boldsymbol{m}$ in slot $(e_1, \ldots, e_i, \ldots, e_t)$ with $e_i' = e_i + v \pmod{\ell_i}$. Since the permutation ρ_i^v only acts on a single index in the tuple (by mapping each slot v positions forward), it is called a one-dimensional *rotation*.

In general, computing the rotation from Equation (4) requires two automorphisms. However, if $g_i^{\ell_i} = 1 \pmod m$, then the equation simplifies to $\rho_i^v(\boldsymbol{m}) = \tau_j(\boldsymbol{m})$, so we need only one automorphism. For $1 \leq i \leq t$, we call dimension i "good" if only one automorphism is required and "bad" otherwise.

The Frobenius map. The rotations from Equation (4) use automorphisms τ_j and τ_k, where $j^{-1} \pmod m$ and k are in S (which represents the quotient group $\mathbb{Z}_m^*/\langle p \rangle$). However, the full automorphism group of $\mathcal{R}/\mathbb{Z}$ consists of τ_j with $j \in \mathbb{Z}_m^*$. The remaining automorphisms for $j \in \langle p \rangle$ induce automorphisms on E: they act on each plaintext slot individually as the map $a(\zeta_m) \mapsto a(\zeta_m^j)$ for arbitrary $a(X)$. This subgroup of automorphisms is generated by $\sigma = \tau_p$ (the so-called *Frobenius map*), which acts on the slots as $\sigma_E \colon a(\zeta_m) \mapsto a(\zeta_m^p)$.

2.3 The Slot-to-Coefficient Transformation in CKKS

The slot-to-coefficient transformation is achieved by evaluating the decoding function homomorphically, which is a linear transformation over the plaintext slots. Similarly, the coefficient-to-slot transformation evaluates the encoding function, which corresponds to the inverse linear transformation. In the CKKS scheme, this involves multiplication by an $m/4 \times m/4$-matrix defined as

$$S_{m/4} = \left(\zeta_m^{5^i \cdot \mathsf{rev}_{m/4}(j)}\right)_{0 \leq i,j < m/4}, \tag{5}$$

where ζ_m is a primitive m-th root of unity, and $\mathsf{rev}_{m/4}$ denotes the standard bit-reversal permutation of $m/4$ items (the definition of $S_{m/4}$ above specifies entries in row i and column j). Although CKKS defines the matrix over the complex numbers, it trivially extends to other rings that have an m-th root of unity ζ_m (such as the slot algebra E).

In order to efficiently evaluate multiplication by $S_{m/4}$, Han et al. [HHC19] show an FFT-like method to decompose this matrix into a product of sparse matrices. More specifically, they prove the following lemma.

[2] More information about the construction of S for power-of-two cyclotomics will be given in Section 3.

Lemma 2. *Let $S_{m/4}$ be as above, and correspondingly, let $S_{m/8}$ be the $m/8 \times m/8$-matrix defined with respect to $\zeta_{m/2} = \zeta_m^2$. Then for $m \geq 8$, we have*

$$S_{m/4} = \begin{bmatrix} I & W_{m/8} \\ I & -W_{m/8} \end{bmatrix} \cdot \begin{bmatrix} S_{m/8} & 0 \\ 0 & S_{m/8} \end{bmatrix},$$

where $W_{m/8} = \mathsf{diag}(\zeta_m^{5^i})_{0 \leq i < m/8}$.

By applying the above lemma recursively on $S_{m/8}$, we can factor $S_{m/4}$ into a product of $\log_2(m/4)$ sparse matrices (each of which contains only $m/2$ non-zero elements). Finally, note that it can be useful to exploit an "incomplete" factorization of $S_{m/4}$ in which multiple factors are merged, as this leads to less noise growth in homomorphic encryption (additional details will be given later).

2.4 Baby-Step/Giant-Step Algorithm

2.4.1 Linear Transformations on $\mathcal{R}_{p^e}$

To implement multiplication by the matrices from Section 2.3, one can use the baby-step/giant-step algorithm [HS18]. This algorithm multiplies the vector of plaintext slots (in one dimension of S) by a generic matrix. More specifically, one can express an E-linear transformation on a plaintext $\boldsymbol{m} \in \mathcal{R}_{p^e}$ in dimension i as

$$L(\boldsymbol{m}) = \sum_{v=0}^{\ell_i - 1} \boldsymbol{\kappa}(v) \cdot \rho_i^v(\boldsymbol{m}),$$

where $\boldsymbol{\kappa}(v) \in \mathcal{R}_{p^e}$ are appropriate constants. Letting $g = \lceil \sqrt{\ell_i} \rceil$ and $h = \lceil \ell_i / g \rceil$, the idea is to rewrite the linear transformation as

$$\begin{aligned} L(\boldsymbol{m}) &= \sum_{v=0}^{\ell_i - 1} \boldsymbol{\kappa}(v) \cdot (\boldsymbol{\mu}(v) \cdot \tau^v(\boldsymbol{m}) + (1 - \boldsymbol{\mu}(v)) \cdot \tau^{v - \ell_i}(\boldsymbol{m})) \\ &= \sum_{k=0}^{h-1} \tau^{gk} \left[\sum_{j=0}^{g-1} (\boldsymbol{\kappa}'(j+gk) \cdot \tau^j(\boldsymbol{m}) + \boldsymbol{\kappa}''(j+gk) \cdot \tau^j(\tau^{-\ell_i}(\boldsymbol{m}))) \right], \end{aligned} \tag{6}$$

where $\tau = \tau_{g_i}^{-1}$ and

$$\begin{aligned} \boldsymbol{\kappa}'(j+gk) &= \tau^{-gk}(\boldsymbol{\mu}(j+gk) \cdot \boldsymbol{\kappa}(j+gk)), \\ \boldsymbol{\kappa}''(j+gk) &= \tau^{-gk}((1 - \boldsymbol{\mu}(j+gk)) \cdot \boldsymbol{\kappa}(j+gk)). \end{aligned}$$

It is clear that Equation (6) can be computed using $O(\sqrt{\ell_i})$ automorphisms and $O(\ell_i)$ plaintext-ciphertext multiplications. Moreover, if key switching keys are available for all j, then we can use well-known (double-)hoisting techniques [HS18, BMTH21] to speed up the computation of $\tau^j(\boldsymbol{m})$ and $\tau^j(\tau^{-\ell_i}(\boldsymbol{m}))$. This is possible because both are sequences of automorphisms on the same input (respectively $\boldsymbol{m}$ and $\tau^{-\ell_i}(\boldsymbol{m})$).

In a good dimension, Equation (6) collapses to

$$L(\boldsymbol{m}) = \sum_{k=0}^{h-1} \tau^{gk} \left[\sum_{j=0}^{g-1} \boldsymbol{\kappa}'(j+gk) \cdot \tau^j(\boldsymbol{m}) \right], \tag{7}$$

where $\boldsymbol{\kappa}'(j+gk) = \tau^{-gk}(\boldsymbol{\kappa}(j+gk))$. This saves approximately 50% of the automorphisms and plaintext-ciphertext multiplications in the inner-sum computation.

2.4.2 Linear Transformations on E

Similarly to above, one can express a $\mathbb{Z}_{p^e}$-linear transformation on $a \in E$ as a weighted sum of $\sigma_E^v(a)$. As such, for a plaintext $\boldsymbol{m} \in \mathcal{R}_{p^e}$, the map

$$L(\boldsymbol{m}) = \sum_{v=0}^{d-1} \boldsymbol{\kappa}(v) \cdot \sigma^v(\boldsymbol{m}) \tag{8}$$

acts on each slot individually as a $\mathbb{Z}_{p^e}$-linear transformation. This functionality can be implemented in the same manner as Equation (7) if we take $\tau = \sigma$. The cost is dominated by $O(\sqrt{d})$ automorphisms and $O(d)$ plaintext-ciphertext multiplications.

2.4.3 Multidimensional Baby-Step/Giant-Step Algorithm

The baby-step/giant-step algorithm (for E-linear as well as $\mathbb{Z}_{p^e}$-linear transformations) can be generalized to a multidimensional version [CCLS19]. More specifically, the goal is to compute weighted sums of automorphisms τ_i with $i \in I \subseteq \mathbb{Z}_m^*$. We first split the index set I in two components $G, H \subseteq \mathbb{Z}_m^*$ such that each $i \in I$ can be written as $i = jk$ with $j \in G$ and $k \in H$. For a plaintext $\boldsymbol{m} \in \mathcal{R}_{p^e}$, we can now rearrange expressions of the form

$$L(\boldsymbol{m}) = \sum_{i \in I} \boldsymbol{\kappa}(i) \cdot \tau_i(\boldsymbol{m}) = \sum_{k \in H} \tau_k \left[\sum_{j \in G} \boldsymbol{\kappa}'(jk) \cdot \tau_j(\boldsymbol{m}) \right], \tag{9}$$

where $\boldsymbol{\kappa}'(jk) = \tau_k^{-1}(\boldsymbol{\kappa}(jk))$. We note that arbitrary linear transformations can be expressed in the form of Equation (9), so we are neither limited to slot-wise nor one-dimensional linear transformations. Finally, when merging an E-linear and a $\mathbb{Z}_{p^e}$-linear map, one can sometimes evaluate a bad dimension with the same cost as a good dimension, using the strategy of reassigning incomplete rotations [CCLS19].

3 A Different View on Plaintext Encoding

The goal of this section is to introduce useful properties of power-of-two cyclotomics. We first discuss the algebraic structure of the automorphism group and construction of the representative set S which was defined earlier. Then we derive properties about the factorization of cyclotomic polynomials, which will naturally lead to a method for encoding plaintext vectors in a subring of $\mathcal{R}_{p^e}$.

3.1 Structure of the Automorphism Group and Plaintext Slots

For $m \geq 4$ a power of two, it is a well-known fact that $\mathbb{Z}_m^* = \langle 5 \rangle \times \langle -1 \rangle$, where 5 has order $m/4$ and -1 has order 2 [Gau86]. Consequently, this group has two generators and is thus not cyclic for $m \geq 8$. Another useful property is that for $1 \leq r \leq m/4$ with r a power of two, the subgroup $\langle 5^r \rangle$ coincides with the set consisting of all numbers x such that $x = 1 \pmod{4r}$. This is because both sets are subgroups of the cyclic group $\langle 5 \rangle$, and both have the same order $m/(4r)$.

Lemma 3. *Let $m \geq 4$ be a power of two and consider a prime p. If $p = 4r \cdot k + 1$ with r a power of two and k odd, then the number of slots is $\ell = \min(2r, m/2)$. If $p = 4r \cdot k - 1$ with r a power of two and k odd, then the number of slots is $\ell = \min(2r, m/4)$.*

Proof. First note that the proof is trivial if $r \geq m/4$, because $p = \pm 1 \pmod{m}$ in that case. Therefore, the order of p will be equal to 1 or 2, giving $m/2$ or $m/4$ slots respectively. We will therefore assume that $1 \leq r < m/4$ in the rest of the proof.

We now prove the case $p = 4r \cdot k + 1$. Using the property from above, we know that $p \in \langle 5^r \rangle$. On the other hand, we know that $p \notin \langle 5^{2r} \rangle$ since k is odd, so p is not in any strict subgroup of $\langle 5^r \rangle$. It follows that p generates $\langle 5^r \rangle$ and the order of p is equal to $d = m/(4r)$. The number of slots is $\ell = m/(2d) = 2r \leq m/2$.

We now prove the case $p = 4r \cdot k - 1$. Using a similar reasoning as above, we obtain that the order of $-p$ is equal to $m/(4r)$. Since this order is at least 2, it follows that the order of p is also $d = m/(4r)$ and the number of slots is $\ell = m/(2d) = 2r \leq m/4$. □

A noteworthy corollary is that the number of slots is at most $(p+1)/2$. Hence one can only have a good packing density if p is large. The maximum number of $N = m/2$ slots is reached when $p = 1 \pmod m$.

3.1.1 Construction of the Representative Set

The set S from Equation (2) forms a complete system of representatives for $\mathbb{Z}_m^*/\langle p \rangle$ and can therefore be constructed with one or two generators. If $p = 1 \pmod 4$, then $p \in \langle 5 \rangle$ and the group $\mathbb{Z}_m^*/\langle p \rangle$ is not cyclic in general. Therefore, we use generators $g_1 = 5$ (of order $\ell_1 = \ell/2$) and $g_2 = -1$ (of order $\ell_2 = 2$). On the other hand, if $p = 3 \pmod 4$, then $p \in -\langle 5 \rangle$ and $\mathbb{Z}_m^*/\langle p \rangle$ is cyclic. Therefore, we use generator $g_1 = 5$ (of order $\ell_1 = \ell$).

Note that we can also reinterpret the set S as having $\log_2(\ell)$ dimensions each of size 2. This is done by prepending the generators $5^n, 5^{n/2}, \ldots, 5^2$, where $n = \ell/4$ if $p = 1 \pmod 4$ and $n = \ell/2$ if $p = 3 \pmod 4$. This interpretation is useful for decomposing the slot-to-coefficient transformation in multiple stages. Intermediate interpretations with fewer dimensions of larger size are also possible.

3.2 Factorization of Cyclotomic Polynomials

The following lemma can be obtained by merging the results from Lyubashevsky and Seiler [LS18] and Okada et al. [OPP23]. We also provide a unified and simplified proof for comprehensiveness below.

Lemma 4. *Let $m \geq 4$ be a power of two, then each factor in Equation (1) is of the shape $F_i(X) = X^d + a_i \cdot X^{d/2} + b_i$, where $a_i = 0$ if $p = 1 \pmod 4$.*

Proof. We first prove the case $p = 1 \pmod 4$. Using the result from Lemma 3, we know that $p = 4r \cdot k + 1$ where ℓ divides $2r$. Let $m' = m/d$, then substituting $2r$ by ℓ in the previous equation gives $p = 2\ell \cdot k' + 1 = m' \cdot k' + 1$, so $p = 1 \pmod{m'}$. Using Lemma 1, the m'-th cyclotomic polynomial splits modulo p^e into linear factors:

$$X^{N'} + 1 = F'_1(X) \cdot \ldots \cdot F'_\ell(X) \pmod{p^e}.$$

Then we explicitly obtain the factors of $X^N + 1$ as $F_i(X) = F'_i(X^{N/N'}) = F'_i(X^d)$.

We now prove the case $p = 3 \pmod 4$. Using the result from Lemma 3, we know that $p = 4r \cdot k - 1$ where ℓ divides $2r$. Let $m' = 2m/d$, then substituting $2r$ by ℓ in the previous equation gives $p = 2\ell \cdot k' - 1 = (m'/2) \cdot k' - 1$, so $p^2 = 1 \pmod{m'}$ while $p \neq 1 \pmod{m'}$. Using Lemma 1, the m'-th cyclotomic polynomial splits modulo p^e into quadratic factors:

$$X^{N'} + 1 = F'_1(X) \cdot \ldots \cdot F'_\ell(X) \pmod{p^e}.$$

Then we explicitly obtain the factors of $X^N + 1$ as $F_i(X) = F'_i(X^{N/N'}) = F'_i(X^{d/2})$. □

3.2.1 Encoding Plaintext Vectors in a Subring

One does not have to use the full packing capacity of $\mathcal{R}_{p^e}$, but can also encode plaintext vectors in a subring $\mathcal{R}'_{p^e}$. Since the algebraic structure of $\mathcal{R}'_{p^e}$ now depends on m' rather

than m, this will result in fewer plaintext slots (smaller ℓ) or a lower extension degree of the slot algebra (smaller d) or both. Packing in a subring is useful in applications where a small number of messages suffices.

To take a plaintext from $\mathcal{R}_{p^e}$ to $\mathcal{R}'_{p^e}$, we can homomorphically evaluate the trace of $\mathcal{R}/\mathcal{R}'$. This operation is commonly called coefficient selection [CH18] or subsum [CCS19] in bootstrapping. It is defined over the plaintext space by the map

$$\sum_{i=0}^{N-1} m_i \cdot X^i \mapsto (N/N') \cdot \left(\sum_{i=0}^{N'-1} m_{i \cdot N/N'} \cdot X^{i \cdot N/N'} \right).$$

It can be computed efficiently using only $\log_2(N/N')$ automorphisms by going through all intermediate cyclotomic rings between $\mathcal{R}$ and $\mathcal{R}'$, and then iteratively evaluating the trace for all subrings [AP13]. The factor N/N' can be removed by folding $(N/N')^{-1} \pmod{p^e}$ in any subsequent linear transformation. We omit further details.

Example 1. We say that a plaintext is *sparsely packed* if each slot contains only an element from $\mathbb{Z}_{p^e} \subseteq E$. According to the Chinese remainder theorem, such a plaintext is constructed as

$$\boldsymbol{m} = \sum_{i=1}^{\ell} m_i \cdot G_i(X), \quad \text{where} \quad G_i(X) = \frac{X^N+1}{F_i(X)} \cdot \left[\left(\frac{X^N+1}{F_i(X)} \right)^{-1} \pmod{F_i(X)} \right].$$

Here the polynomials $F_i(X)$ denote the factors from Equation (1) and $m_i \in \mathbb{Z}_{p^e}$. Reduction modulo p^e is implicit in the equation above. Following Lemma 4, it is easy to see that $F_i(X) \in \mathbb{Z}_{p^e}[X^c]$ and therefore $G_i(X) \in \mathbb{Z}_{p^e}[X^c]$, where $c = d$ if $p = 1 \pmod 4$ and $c = d/2$ if $p = 3 \pmod 4$. It follows directly that $\boldsymbol{m} \in \mathcal{R}'_{p^e}$ with respect to $m' = m/c$.

4 The New Transformation for BGV and BFV

This section describes the new version of the slot-to-coefficient and coefficient-to-slot transformations. We offer a complete treatment of all situations, where we distinguish the case $p = 1 \pmod 4$ (treated in Section 4.1) from $p = 3 \pmod 4$ (treated in Section 4.2). The linear transformations are fundamentally different in both cases, because the first case gives a non-cyclic and the second case gives a cyclic permutation group.

Notably, our algorithm has several advantages over earlier approaches. In summary, it is based on the following design principles:

- Our method is fully parameterizable (the only restriction we have is a power-of-two cyclotomic index) and can handle both fully and sparsely packed slots.
- The linear transformations can be decomposed in multiple stages, which gives rise to a complexity-depth trade-off. Earlier approaches for power-of-two cyclotomics can only evaluate the transformation directly in one stage [HS21, CH18].
- We propose a simpler variant of the "slot unpacking" procedure, which requires only $O(d)$ instead of $O(d^2)$ FHE operations, and which uses no multiplicative levels.

A more detailed comparison to the state-of-the-art is given in Section 5.2.

4.1 New Method for $p = 1 \pmod 4$

We will identify the slots of a plaintext with a vector in E^ℓ. This is done by "flattening" the representative set as $S = \{1, 5, \ldots, 5^{\ell_1-1}, -1, -5, \ldots, -5^{\ell_1-1}\}$ and filling the plaintext in Equation (3). We also use the notation $\zeta_{m,i} = \zeta_m^{h_i}$, where h_i is the i-th element of S.

4.1.1 Fully Packed Slots

In the slot-to-coefficient transformation, we start from a plaintext $\boldsymbol{m}$ that encodes

$$\overrightarrow{m} = \left(\sum_{j=0}^{d-1} m_{i,j} \cdot \zeta_{m,i}^{j} \right)_{0 \le i < \ell}$$

in the slots. Note that the encoding is done with respect to $\zeta_{m,i}^{j}$ instead of ζ_{m}^{j} to simplify unpacking and repacking later. The goal is to map the elements $m_{i,j}$ to the coefficients of a new plaintext $\boldsymbol{m}'''$. This can be done in three steps:

1. Perform a slot-wise linear transformation M that maps $\zeta_{m,i}^{j} \mapsto \zeta_{m}^{j}$ for $0 \le j < d$ in slot i and extends by $\mathbb{Z}_{p^e}$-linearity. Note that this transformation depends on i and is thus different for each slot. After this step, the plaintext encodes the slot-vector

$$\overrightarrow{m}' = \left(\sum_{j=0}^{d-1} m_{i,j} \cdot \zeta_{m}^{j} \right)_{0 \le i < \ell}.$$

2. Multiply the slot-vector by (a column-permuted version of) the decoding matrix

$$U_\ell = \left(\zeta_{m,i}^{d \cdot j} \right)_{0 \le i,j < \ell},$$

   where the matrix above is specified by the entries in its i-th row and j-th column. Observe that $\zeta_{m}^{d} \in \mathbb{Z}_{p^e}$, which implies $U_\ell \in \mathbb{Z}_{p^e}^{\ell \times \ell}$. After this step, the plaintext encodes the slot-vector

$$\overrightarrow{m}'' = \left(\sum_{k=0}^{\ell-1} \sum_{j=0}^{d-1} m_{k,j} \cdot \zeta_{m}^{j} \cdot \zeta_{m,i}^{d \cdot k} \right)_{0 \le i < \ell}.$$

   A column-permuted version of U_ℓ will simply map the elements $m_{i,j}$ to a different order of the coefficients.
3. Perform the linear transformation M^{-1} that maps $\zeta_{m}^{j} \mapsto \zeta_{m,i}^{j}$ for $0 \le j < d$ in slot i. After this step, the plaintext encodes the slot-vector

$$\overrightarrow{m}''' = \left(\sum_{k=0}^{\ell-1} \sum_{j=0}^{d-1} m_{k,j} \cdot \zeta_{m,i}^{j+d \cdot k} \right)_{0 \le i < \ell},$$

   which corresponds to the plaintext

$$\boldsymbol{m}''' = \sum_{k=0}^{\ell-1} \sum_{j=0}^{d-1} m_{k,j} \cdot X^{j+d \cdot k}.$$

   Here we used the observation that M^{-1} does not act on the entries of U_ℓ (i.e., powers of ζ_{m}^{d}) due to $\mathbb{Z}_{p^e}$-linearity.

We emphasize that the slot-to-coefficient transformation is only $\mathbb{Z}_{p^e}$-linear and not E-linear in general, which is why step 1 and step 3 are required. Step 2 on its own is an E-linear transformation over the plaintext slots. The inverse map (coefficient-to-slot transformation) is given by $M^{-1}U_\ell^{-1}M$ instead of $M^{-1}U_\ell M$.

Matrix decomposition. The matrix from step 2 can be decomposed into a product of sparse matrices. To do this, we use the column-permuted version

$$U_\ell = \begin{bmatrix} S_{\ell/2} & \zeta_4 \cdot S_{\ell/2} \\ S'_{\ell/2} & -\zeta_4 \cdot S'_{\ell/2} \end{bmatrix} = \begin{bmatrix} S_{\ell/2} & 0 \\ 0 & S'_{\ell/2} \end{bmatrix} \cdot \begin{bmatrix} I & \zeta_4 \cdot I \\ I & -\zeta_4 \cdot I \end{bmatrix},$$

where $\zeta_4 = \zeta_m^{m/4}$. The submatrices $S_{\ell/2}$ and $S'_{\ell/2}$ are defined in Equation (5) and generated by the roots of unity ζ_m^d and ζ_m^{-d} respectively. Observe that $S_{\ell/2}$ and $S'_{\ell/2}$ can be further decomposed using Lemma 2. As a result, we can write U_ℓ as a product of $\log_2(\ell)$ sparse matrices, each of which acts on only one dimension of S (in the alternative interpretation where each dimension has size 2). The leftmost factor of this product will act on the first dimension and the rightmost factor on the last dimension.

Unpacking the slots. After the coefficient-to-slot operation, bootstrapping needs to split the ciphertext that encrypts $\boldsymbol{m}$ in d sparsely packed ciphertexts. This can be done as follows: first, we homomorphically split the plaintext in two parts $\boldsymbol{m}_1 = \boldsymbol{m} + \sigma^{d/2}(\boldsymbol{m})$ and $\boldsymbol{m}_2 = X^{-1} \cdot (\boldsymbol{m} - \sigma^{d/2}(\boldsymbol{m}))$. The automorphism $\sigma^{d/2}$ will simply map $X \mapsto -X$ and thus acts on the slots as $\zeta_m \mapsto -\zeta_m$ (note that this is indeed an automorphism of E due to the special shape of $F_i(X)$ in Lemma 4). Therefore, the plaintexts will encode the slot-vectors

$$\overrightarrow{m}_1 = \left(2 \cdot \sum_{j=0}^{d/2-1} m_{i,2j} \cdot \zeta_{m,i}^{2j} \right)_{0 \le i < \ell} \quad \text{and} \quad \overrightarrow{m}_2 = \left(2 \cdot \sum_{j=0}^{d/2-1} m_{i,2j+1} \cdot \zeta_{m,i}^{2j} \right)_{0 \le i < \ell}.$$

We emphasize that multiplication by powers of X does not increase the noise as it just permutes coefficients negacyclicly. To obtain d sparse ciphertexts, we apply this procedure iteratively on the plaintexts $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$: in level $i = 1, \dots, \log_2(d)$ of the iteration, we compute $\boldsymbol{m}_1 = \boldsymbol{m} + \sigma^{d/2^i}(\boldsymbol{m})$ and $\boldsymbol{m}_2 = X^{-2^{i-1}} \cdot (\boldsymbol{m} - \sigma^{d/2^i}(\boldsymbol{m}))$, where $\boldsymbol{m}$ loops over all outputs of the previous iteration. At the end of the unpacking procedure, the undesired factors of 2 in $\overrightarrow{m}_1$ and $\overrightarrow{m}_2$ will accumulate to a factor of d. It can be removed by folding a factor of $d^{-1} \pmod{p^e}$ in the preceding linear transformation.

Repacking the slots. Before the slot-to-coefficient transformation, bootstrapping needs to recombine d sparsely packed ciphertexts into one fully packed ciphertext. This is the inverse of unpacking and can be computed as $\boldsymbol{m} = \boldsymbol{m}_1/2 + X^{2^{i-1}} \cdot \boldsymbol{m}_2/2$, where we loop over i in reverse order than during unpacking. Note that the input of repacking is indeed given by $\boldsymbol{m}_1/2$ and $\boldsymbol{m}_2/2$ rather than $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$, assuming that the factor of d was removed appropriately before unpacking.

4.1.2 Sparsely Packed Slots

According to Example 1, sparsely packed plaintexts (where the slots contain an element from $\mathbb{Z}_{p^e}$) live in the subring $\mathcal{R}'_{p^e}$ with respect to $m' = m/d$. Conversely, it is easy to see that elements from the subring $\mathcal{R}'_{p^e}$ are sparsely packed. As such, both the slot-encoded plaintext $\boldsymbol{m}$ and the coefficient-encoded plaintext $\boldsymbol{m}'''$ are sparsely packed. Step 1 and step 3 of the slot-to-coefficient transformation can now be omitted, because they have no effect on $\mathbb{Z}_{p^e}$. Therefore, the slot-to-coefficient transformation is U_ℓ and the coefficient-to-slot transformation is U_ℓ^{-1}.

In some situations (such as "thin" bootstrapping [CH18]), the input plaintext of the coefficient-to-slot transformation is an element of $\mathcal{R}_{p^e}$ rather than $\mathcal{R}'_{p^e}$. Consequently, we first need to map the plaintext to $\mathcal{R}'_{p^e}$, which is done by removing redundant coefficients. This can be easily achieved with the trace method from Section 3.2.1 if we fold an additional factor of $d^{-1} \pmod{p^e}$ in U_ℓ^{-1}. In summary, the transformation has two steps: (1) evaluate the trace of $\mathcal{R}/\mathcal{R}'$ and (2) multiplication by $(d \cdot U_\ell)^{-1}$.

4.2 New Method for $p = 3 \pmod 4$

We will identify the slots of a plaintext with a vector in E^ℓ. This is done by "flattening" the representative set as $S = \{1, 5, \ldots, 5^{\ell_1 - 1}\}$ and filling the plaintext in Equation (3). Similarly to above, we use the notation $\zeta_{m,i} = \zeta_m^{h_i}$, where h_i is the i-th element of S. We also define $\zeta_4 = \zeta_m^{m/4}$ and $E' = \mathbb{Z}_{p^e}[\zeta_4]$.

4.2.1 Fully Packed Slots

In the slot-to-coefficient transformation, we start from a plaintext $\boldsymbol{m}$ that encodes

$$\overrightarrow{m} = \left(\sum_{j=0}^{d/2-1} (m_{i,j} + n_{i,j} \cdot \zeta_4) \cdot \zeta_{m,i}^j \right)_{0 \le i < \ell}$$

in the slots. Note that the encoding is done with respect to $\zeta_{m,i}^j$ instead of ζ_m^j to simplify unpacking and repacking later. The goal is to map the elements $m_{i,j}$ and $n_{i,j}$ to the coefficients of a new plaintext $\boldsymbol{m}'''$. This can be done in three steps:

1. Perform a slot-wise linear transformation M that maps $\zeta_{m,i}^j \mapsto \zeta_m^j$ for $0 \le j < d/2$ in slot i and extends by E'-linearity. Note that this transformation depends on i and is thus different for each slot. After this step, the plaintext encodes the slot-vector

$$\overrightarrow{m}' = \left(\sum_{j=0}^{d/2-1} (m_{i,j} + n_{i,j} \cdot \zeta_4) \cdot \zeta_m^j \right)_{0 \le i < \ell}.$$

2. Multiply the slot-vector by (a column-permuted version of) the decoding matrix

$$U_\ell = \left(\zeta_{m,i}^{d \cdot j/2} \right)_{0 \le i,j < \ell},$$

where the matrix above is specified by the entries in its i-th row and j-th column. Observe that $\zeta_m^{d/2} \in E'$, which implies $U_\ell \in (E')^{\ell \times \ell}$. After this step, the plaintext encodes the slot-vector

$$\overrightarrow{m}'' = \left(\sum_{k=0}^{\ell-1} \sum_{j=0}^{d/2-1} (m_{k,j} + n_{k,j} \cdot \zeta_4) \cdot \zeta_m^j \cdot \zeta_{m,i}^{d \cdot k/2} \right)_{0 \le i < \ell}.$$

A column-permuted version of U_ℓ will simply map the elements $m_{i,j}$ and $n_{i,j}$ to a different order of the coefficients.

3. Perform the linear transformation M^{-1} that maps $\zeta_m^j \mapsto \zeta_{m,i}^j$ for $0 \le j < d/2$ in slot i. After this step, the plaintext encodes the slot-vector

$$\overrightarrow{m}''' = \left(\sum_{k=0}^{\ell-1} \sum_{j=0}^{d/2-1} (m_{k,j} + n_{k,j} \cdot \zeta_4) \cdot \zeta_{m,i}^{j + d \cdot k/2} \right)_{0 \le i < \ell},$$

which corresponds to the plaintext

$$\boldsymbol{m}''' = \sum_{k=0}^{\ell-1} \sum_{j=0}^{d/2-1} (m_{k,j} + n_{k,j} \cdot X^{m/4}) \cdot X^{j + d \cdot k/2}.$$

Here we used the observation that M^{-1} does not act on the entries of U_ℓ (i.e., powers of $\zeta_m^{d/2}$) due to E'-linearity.

We emphasize that the slot-to-coefficient transformation is only E'-linear and not E-linear in general, which is why step 1 and step 3 are required (step 2 alone is E-linear). The inverse map (coefficient-to-slot transformation) is given by $M^{-1} U_\ell^{-1} M$ instead of $M^{-1} U_\ell M$.

Matrix decomposition. The matrix from step 2 can be decomposed into a product of sparse matrices. To do this, we use the column-permuted version $U_\ell = S_\ell$, which is defined in Equation (5) and generated by the root of unity $\zeta_m^{d/2}$. Observe that S_ℓ can be further decomposed using Lemma 2. As a result, we can write it as a product of $\log_2(\ell)$ sparse matrices, each of which acts on only one dimension of S (in the alternative interpretation where each dimension has size 2). The leftmost factor of this product will act on the first dimension and the rightmost factor on the last dimension.

Unpacking the slots. The unpacking procedure is similar to the case $p = 1 \pmod 4$. First, we iteratively split the ciphertext in $d/2$ ciphertexts, each one encoding elements from E' in the slots. This is done by homomorphically computing $\boldsymbol{m}_1 = \boldsymbol{m} + \sigma^{d/2^i}(\boldsymbol{m})$ and $\boldsymbol{m}_2 = X^{-2^{i-1}} \cdot (\boldsymbol{m} - \sigma^{d/2^i}(\boldsymbol{m}))$ in level $i = 1, \ldots, \log_2(d/2)$ of the iteration, where $\boldsymbol{m}$ loops over all outputs of the previous iteration. Note that multiplication by powers of X does not increase the noise.

After obtaining $d/2$ ciphertexts that encode elements from E', we split each one in two ciphertexts that encode elements from $\mathbb{Z}_{p^e}$. This is done by homomorphically computing $\boldsymbol{m}_1 = \boldsymbol{m} + \sigma(\boldsymbol{m})$ and $\boldsymbol{m}_2 = X^{-m/4} \cdot (\boldsymbol{m} - \sigma(\boldsymbol{m}))$, where $\boldsymbol{m}$ loops over all outputs of the previous step. Again, we can remove the undesired factor of d by merging $d^{-1} \pmod{p^e}$ in the preceding linear transformation.

Repacking the slots. This operation is the inverse of unpacking and can be computed iteratively as $\boldsymbol{m} = \boldsymbol{m}_1/2 + X^{2^{i-1}} \cdot \boldsymbol{m}_2/2$, where $i = \log_2(m/2), \log_2(d/2), \ldots, 1$. Note that the input of repacking is indeed given by $\boldsymbol{m}_1/2$ and $\boldsymbol{m}_2/2$ rather than $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$, assuming that the factor of d was removed appropriately before unpacking.

4.2.2 Sparsely Packed Slots

According to Example 1, sparsely packed plaintexts (where the slots contain an element from $\mathbb{Z}_{p^e}$) live in the subring $\mathcal{R}'_{p^e}$ with respect to $m' = 2m/d$. This result can even be extended to slots encoding values in E'. Conversely, it is easy to see that the subring $\mathcal{R}'_{p^e}$ packs elements from E'. As such, both the slot-encoded plaintext $\boldsymbol{m}$ and the coefficient-encoded plaintext $\boldsymbol{m}'''$ pack elements from E'. Step 1 and step 3 of the slot-to-coefficient transformation can now be omitted, because they have no effect on E'. Therefore, the slot-to-coefficient transformation is U_ℓ and the coefficient-to-slot transformation is U_ℓ^{-1}.

In some situations (such as "thin" bootstrapping [CH18]), the input plaintext of the coefficient-to-slot transformation is an element of $\mathcal{R}_{p^e}$ rather than $\mathcal{R}'_{p^e}$. Consequently, we first need to map the plaintext to $\mathcal{R}'_{p^e}$, which is done by removing redundant coefficients. This can be easily achieved with the trace method from Section 3.2.1. However, using the trace is not sufficient in this case, because the slots can still encode elements from E' rather than $\mathbb{Z}_{p^e}$ after multiplication by U_ℓ^{-1} (the trace cannot remove all coefficients $n_{i,j}$). Therefore, we need to post-process the output of the coefficient-to-slot transformation by computing the trace of $E'/\mathbb{Z}_{p^e}$ in each slot (which is done as $\boldsymbol{m} \mapsto \boldsymbol{m} + \sigma(\boldsymbol{m})$). Again, we fold an additional factor of $d^{-1} \pmod{p^e}$ in U_ℓ^{-1} to remove undesired factors of 2, which were introduced during pre-processing and post-processing with the trace. In summary, the transformation has three steps: (1) evaluate the trace of $\mathcal{R}/\mathcal{R}'$, (2) multiplication by $(d \cdot U_\ell)^{-1}$ and (3) evaluate the trace of $E'/\mathbb{Z}_{p^e}$ slot-wise.

5 Implementation and Results

This section discusses the implementation aspects of our method. We provide a complexity analysis assuming a baby-step/giant-step implementation, and we also compare operation counts for two common parameter sets to prior work.

5.1 Complexity Analysis

5.1.1 Fully Packed Slots

We first analyze the cost of evaluating the fully packed slot-to-coefficient transformation (the cost of the inverse map is identical). We break it down in two components:

- Baby-step/giant-step evaluations of size n in good dimensions. This needs roughly $2\sqrt{n}$ automorphisms ($\sqrt{n}$ can be hoisted) and n plaintext-ciphertext multiplications.
- Baby-step/giant-step evaluations of size n in bad dimensions. This needs roughly $3\sqrt{n}$ automorphisms ($2\sqrt{n}$ can be hoisted) and $2n$ plaintext-ciphertext multiplications.

First, we consider the maps M and M^{-1}, which can be expressed as a sum of d terms using Equation (8). In the case $p = 3 \pmod 4$, both maps are E'-linear and we only need the automorphisms for even v (those correspond to the automorphism group of E/E'), so we have $d/2$ non-zero terms in Equation (8). To simplify notation, we define $c = d$ if $p = 1 \pmod 4$ and $c = d/2$ if $p = 3 \pmod 4$. Direct implementation of M or M^{-1} then requires one baby-step/giant-step evaluation of size c in a good dimension. In terms of noise growth, it uses one level of plaintext-ciphertext multiplications.

For small values of c, one may want to merge the maps M and M^{-1} with a factor of U_ℓ in order not to waste a multiplicative level. Suppose we have $U_\ell = U_{\ell,1} \cdot \ldots \cdot U_{\ell,T}$, where each factor $U_{\ell,i}$ contains L_i non-zero entries in every row/column (such a factorization can be obtained by merging multiple factors of the matrix decomposition). This means that $\ell = L_1 \cdot \ldots \cdot L_T$. The slot-to-coefficient transformation is now given by

$$(M^{-1}U_{\ell,1}) \cdot U_{\ell,2} \cdot \ldots \cdot U_{\ell,T-1} \cdot (U_{\ell,T}M).$$

Each factor can be implemented with the multidimensional baby-step/giant-step algorithm: the rightmost factor $U_{\ell,T}M$ corresponds to a baby-step/giant-step evaluation of size $c \cdot L_T$ in a bad dimension; the intermediate factors $U_{\ell,i}$ correspond to a baby-step/giant-step evaluation of size L_i in a bad dimension; and the leftmost factor $M^{-1}U_{\ell,1}$ corresponds to a baby-step/giant-step evaluation of size $c \cdot L_1$ in a good dimension. In terms of noise growth, it uses T levels of plaintext-ciphertext multiplications. Since larger T reduces the size of each factor L_i, there is a convenient trade-off between computational cost and multiplicative levels.

Unpacking and repacking. The cost of unpacking is dominated by $d-1$ automorphisms; it also requires $d-1$ plaintext-ciphertext multiplications and $2(d-1)$ additions. The cost of repacking is given by $d-1$ plaintext-ciphertext multiplications and equally many additions. Repacking requires no key switching and is therefore much cheaper than unpacking. Finally, note that unpacking and repacking use no multiplicative levels, because all constants are powers of X.

5.1.2 Sparsely Packed Slots

The maps M and M^{-1} can be omitted for sparsely packed slots, so we only need to evaluate $U_\ell = U_{\ell,1} \cdot \ldots \cdot U_{\ell,T}$. Each factor can again be implemented with the multidimensional baby-step/giant-step algorithm: the factors $U_{\ell,i}$ for $i \geq 2$ correspond to a baby-step/giant-step evaluation of size L_i in a bad dimension; and the leftmost factor $U_{\ell,1}$ corresponds to a baby-step/giant-step evaluation of size L_1 in a good dimension. In terms of noise growth, it uses T levels of plaintext-ciphertext multiplications, which gives the same trade-off between computational cost and multiplicative levels as before. Finally, evaluating the trace(s) for the coefficient-to-slot transformation can be done with $\log_2(d)$ automorphisms and no multiplicative levels.

5.2 Comparison to the State-of-the-Art

Earlier approaches for the slot-to-coefficient transformation using power-of-two cyclotomic rings [HS21, CH18] involve homomorphic multiplication by a full $N \times N$-matrix (in the fully packed case) or an $\ell \times \ell$-matrix (in the sparsely packed case). This results in a linear number of FHE operations. On the other hand, full matrix decomposition with our method requires only a logarithmic number of FHE operations (for fully packed slots, this is assuming that d is constant or logarithmic, because the number of FHE operations scales linearly in d).

Furthermore, our procedure for slot unpacking is also asymptotically cheaper than the one from Halevi and Shoup [HS21]. Our method requires only $O(d)$ automorphisms and multiplications, whereas Halevi and Shoup require $O(d^2)$ multiplications. Moreover, our method requires no multiplicative levels, compared to one level for Halevi and Shoup.

Our cheaper version of the slot unpacking procedure does not come completely for free. It necessitates the extra maps M in the slot-to-coefficient transformation and M^{-1} in the coefficient-to-slot transformation, which are not present in the method of Halevi and Shoup. However, these maps can be implemented with $O(d)$ FHE multiplications, which is cheaper than $O(d^2)$ FHE multiplications in Halevi and Shoup's unpacking procedure. Moreover, our method allows for more freedom in parameter selection: for small d, one can merge the maps M and M^{-1} with a factor of U_ℓ or U_ℓ^{-1} in order to save a multiplicative level. This is not possible with the approach of Halevi and Shoup.

5.2.1 Concrete Example of Speedup

Table 2 and Table 3 break down the homomorphic algorithms (fully packed slot-to-coefficient transformation, sparsely packed slot-to-coefficient transformation, unpacking, repacking) into basic operations (plaintext-ciphertext multiplications, automorphisms, multiplicative levels) for two different parameter sets. To show that our method uses fewer operations than the state-of-the-art, the tables also compare to the number of operations using the algorithms from HElib [HS21]. For the fully packed slot-to-coefficient transformation in Table 2, it is assumed that the maps M and M^{-1} are merged with $U_{\ell,3}$ and $U_{\ell,1}$ respectively. In Table 3, the fully packed and sparsely packed slot-to-coefficient transformations coincide, because the cyclotomic polynomial splits completely into linear factors. As a result, unpacking and repacking are not required in this table. We conclude from the tables that our method typically uses more multiplicative levels, but the number of basic operations is always lower than in HElib.

Table 2: Operation count for $N = 2^{16}$, $p = 5 \cdot 2^{12} - 1$ and $L_1 \cdot L_2 \cdot L_3 = 2^3 \cdot 2^5 \cdot 2^3$

	Previous methods			Our method		
	# mult	# auto	# levels	# mult	# auto	# levels
Full SlotToCoeff	65536	510	1	448	71	3
Sparse SlotToCoeff	2048	89	1	88	27	3
Unpacking	1024	31	1	31	31	0
Repacking	32	0	1	31	0	0

Table 3: Operation count for $N = 2^{15}$, $p = 2^{16} + 1$ and $L_1 \cdot L_2 = 2^8 \cdot 2^7$

	Previous methods			Our method		
	# mult	# auto	# levels	# mult	# auto	# levels
SlotToCoeff	32768	361	1	512	63	2

Acknowledgements

This work was supported by CyberSecurity Research Flanders with reference number VR20192203. In addition, this work is supported in part by the European Commission through the Horizon 2020 research and innovation program Belfort ERC Advanced Grant 101020005 and by the Defence Advanced Research Projects Agency (DARPA) under contract No. HR0011-21-C-0034 DARPA DPRIVE BASALISC. Robin Geelen is funded by Research Foundation – Flanders (FWO) under a PhD Fellowship fundamental research (project number 1162123N). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the ERC, DARPA, the U.S. Government, the European Union, CyberSecurity Research Flanders or the FWO. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein. The author would also like to thank Jiayi Kang and Frederik Vercauteren for useful discussions.

References

[AP13] Jacob Alperin-Sheriff and Chris Peikert. Practical bootstrapping in quasi-linear time. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2013. doi:10.1007/978-3-642-40041-4_1.

[BCC+22] Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. META-BTS: bootstrapping precision beyond the limit. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 223–234. ACM, 2022. doi:10.1145/3548606.3560696.

[BCK+23] Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, Jai Hyun Park, and Damien Stehlé. HERMES: efficient ring packing using MLWE ciphertexts and application to transciphering. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 37–69. Springer, 2023. doi:10.1007/978-3-031-38551-3_2.

[BGGJ20] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. CHIMERA: combining ring-lwe-based fully homomorphic encryption schemes. *J. Math. Cryptol.*, 14(1):316–338, 2020. URL: https://doi.org/10.1515/jmc-2019-0026, doi:10.1515/JMC-2019-0026.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3):13:1–13:36, 2014. doi:10.1145/2633600.

[BMTH21] Jean-Philippe Bossuat, Christian Mouchet, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings,*

Part I, volume 12696 of *Lecture Notes in Computer Science*, pages 587–617. Springer, 2021. doi:10.1007/978-3-030-77870-5_21.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012. doi:10.1007/978-3-642-32009-5_50.

[CCLS19] Jung Hee Cheon, Hyeongmin Choe, Donghwan Lee, and Yongha Son. Faster linear transformations in helib, revisited. *IEEE Access*, 7:50595–50604, 2019. doi:10.1109/ACCESS.2019.2911300.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 34–54. Springer, 2019. doi:10.1007/978-3-030-17656-3_2.

[CH18] Hao Chen and Kyoohyung Han. Homomorphic lower digits removal and improved FHE bootstrapping. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 315–337. Springer, 2018. doi:10.1007/978-3-319-78381-9_12.

[CHK+18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 360–384. Springer, 2018. doi:10.1007/978-3-319-78381-9_14.

[CHK+21] Jihoon Cho, Jincheol Ha, Seongkwang Kim, ByeongHak Lee, Joohee Lee, Jooyoung Lee, Dukjae Moon, and Hyojin Yoon. Transciphering framework for approximate homomorphic encryption. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 640–669. Springer, 2021. doi:10.1007/978-3-030-92078-4_22.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017. doi:10.1007/978-3-319-70694-8_15.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012. URL: http://eprint.iacr.org/2012/144.

[Gau86] Carl Friedrich. Gauss. *Disquisitiones arithmeticae*. Springer, Berlin, 1986.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*, volume 7293 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2012. doi:10.1007/978-3-642-30057-8_1.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012. doi:10.1007/978-3-642-29011-4_28.

[GIKV23] Robin Geelen, Ilia Iliashenko, Jiayi Kang, and Frederik Vercauteren. On polynomial functions modulo p^e and faster bootstrapping for homomorphic encryption. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 257–286. Springer, 2023. doi:10.1007/978-3-031-30620-4_9.

[GV23] Robin Geelen and Frederik Vercauteren. Bootstrapping for BGV and BFV revisited. *J. Cryptol.*, 36(2):12, 2023. URL: https://doi.org/10.1007/s00145-023-09454-6, doi:10.1007/S00145-023-09454-6.

[HHC19] Kyoohyung Han, Minki Hhan, and Jung Hee Cheon. Improved homomorphic discrete fourier transforms and FHE bootstrapping. *IEEE Access*, 7:57361–57370, 2019. doi:10.1109/ACCESS.2019.2913850.

[HS18] Shai Halevi and Victor Shoup. Faster homomorphic linear transformations in helib. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 93–120. Springer, 2018. doi:10.1007/978-3-319-96884-1_4.

[HS20] Shai Halevi and Victor Shoup. Design and implementation of helib: a homomorphic encryption library. *IACR Cryptol. ePrint Arch.*, page 1481, 2020. URL: https://eprint.iacr.org/2020/1481.

[HS21] Shai Halevi and Victor Shoup. Bootstrapping for helib. *J. Cryptol.*, 34(1):7, 2021. URL: https://doi.org/10.1007/s00145-020-09368-7, doi:10.1007/S00145-020-09368-7.

[KPZ21] Andrey Kim, Yuriy Polyakov, and Vincent Zucca. Revisiting homomorphic encryption schemes for finite fields. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International*

Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III, volume 13092 of *Lecture Notes in Computer Science*, pages 608–639. Springer, 2021. doi:10.1007/978-3-030-92078-4_21.

[KSS24] Jaehyung Kim, Jinyeong Seo, and Yongsoo Song. Simpler and faster bfv bootstrapping for arbitrary plaintext modulus from ckks. Cryptology ePrint Archive, Paper 2024/109, 2024. https://eprint.iacr.org/2024/109. URL: https://eprint.iacr.org/2024/109.

[LHH+21] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. PEGASUS: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1057–1073. IEEE, 2021. doi:10.1109/SP40001.2021.00043.

[LS18] Vadim Lyubashevsky and Gregor Seiler. Short, invertible elements in partially splitting cyclotomic rings and applications to lattice-based zero-knowledge proofs. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 204–224. Springer, 2018. doi:10.1007/978-3-319-78381-9_8.

[LW23a] Feng-Hao Liu and Han Wang. Batch bootstrapping I: - A new framework for SIMD bootstrapping in polynomial modulus. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 321–352. Springer, 2023. doi:10.1007/978-3-031-30620-4_11.

[LW23b] Zeyu Liu and Yunhao Wang. Amortized functional bootstrapping in less than 7ms, with $\tilde{o}(1)$ polynomial multiplications. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 101–132, Singapore, 2023. Springer Nature Singapore.

[MHWW24] Shihe Ma, Tairong Huang, Anyu Wang, and Xiaoyun Wang. Accelerating bgv bootstrapping for large p using null polynomials over $\mathbb{Z}_{p^e}$. Cryptology ePrint Archive, Paper 2024/115, 2024. https://eprint.iacr.org/2024/115. URL: https://eprint.iacr.org/2024/115.

[OPP23] Hiroki Okada, Rachel Player, and Simon Pohmann. Homomorphic polynomial evaluation using galois structure and applications to bfv bootstrapping. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 69–100, Singapore, 2023. Springer Nature Singapore.

[SV14] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptogr.*, 71(1):57–81, 2014. URL: https://doi.org/10.1007/s10623-012-9720-4, doi:10.1007/S10623-012-9720-4.