Faster BGV Bootstrapping for Power-of-Two Cyclotomics through Homomorphic NTT

Shihe Ma[1], Tairong Huang[2], Anyu Wang[2,3,4(✉)], and Xiaoyun Wang[2,3,4,5,6]

[1] Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China, msh21@mails.tsinghua.edu.cn

[2] Institute for Advanced Study, BNRist, Tsinghua University, Beijing, China, htr19@mails.tsinghua.edu.cn, anyuwang,xiaoyunwang@tsinghua.edu.cn

[3] Zhongguancun Laboratory, Beijing, China

[4] National Financial Cryptography Research Center, Beijing, China

[5] Shandong Institute of Blockchain, Jinan, China

[6] Key Laboratory of Cryptologic Technology and Information Security (Ministry of Education), School of Cyber Science and Technology, Shandong University, China

Abstract. Power-of-two cyclotomics is a popular choice when instantiating the BGV scheme because of its efficiency and compliance with the FHE standard. However, in power-of-two cyclotomics, the linear transformations in BGV bootstrapping cannot be decomposed into sub-transformations for acceleration with existing techniques. Thus, they can be highly time-consuming when the number of slots is large, degrading the advantage brought by the SIMD property of the plaintext space. By exploiting the algebraic structure of power-of-two cyclotomics, this paper derives explicit decomposition of the linear transformations in BGV bootstrapping into NTT-like sub-transformations, which are highly efficient to compute homomorphically. Moreover, multiple optimizations are made to evaluate homomorphic linear transformations, including modified BSGS algorithms, trade-offs between level and time, and specific simplifications for thin and general bootstrapping. We implement our method on HElib. With the number of slots ranging from 4096 to 32768, we obtain a 7.35x$\sim$143x improvement in the running time of linear transformations and a 4.79x$\sim$66.4x improvement in bootstrapping throughput, compared to previous works or the naive approach.

Keywords: Fully Homomorphic Encryption · BGV · Bootstrapping · NTT.

1 Introduction

Fully homomorphic encryption (FHE) allows anyone to compute over encrypted data without access to the decryption key or the underlying plaintext. Thus, FHE is useful in privacy-preserving computing like outsourced computation and privacy-preserving machine learning [24,4]. Among the various FHE schemes, when the data to be computed homomorphically are represented as integers, the common choice of the underlying FHE scheme is BGV [5] or BFV [14].

BGV/BFV offers the single instruction multiple data (SIMD) functionality, in which a plaintext encodes an array of elements and homomorphic operations are performed simultaneously on each slot of the array.

The bootstrapping technique first proposed by Gentry [17] plays an important role in FHE. By homomorphically decrypting the ciphertext, it refreshes the noise in the ciphertext before the validity of the ciphertext is corrupted, thus allowing for an unlimited number of homomorphic operations. The bootstrapping of BGV has been studied extensively in the past years [18,10,21,15,16,25], leading to significant improvements in its performance.

From an implementation standpoint, power-of-two cyclotomics are frequently employed to instantiate BGV. A majority of FHE libraries, including SEAL [28], OpenFHE [3], and lattigo [23], exclusively use power-of-two cyclotomics, which is also the only cyclotomics recommended in the FHE standard [1]. However, in the context of power-of-two cyclotomics, the existing techniques [21,10,16] for computing the linear transformations in BGV bootstrapping are highly inefficient when dealing with a large number of slots.

Let M denote the cyclotomic order and p the prime of the plaintext modulus in the BGV scheme. Halevi and Shoup [21] propose a method for enabling fast linear transformations in bootstrapping, which requires M to have multiple distinct prime factors so that the linear transformations can be decomposed into multiple sub-transformations by leveraging the structure of the powerful basis. Each sub-transformation has a dimension much smaller than the entire transformation, making it more computationally efficient. However, this decomposition is impossible when M is a power of two, as M only has a single prime factor 2 and a trivial powerful basis structure. Furthermore, Halevi and Shoup's method requires that $\mathbb{Z}_M^*/\langle p \rangle$ is a cyclic group, which is not the case when M is a power of two and $p \equiv 1 \bmod 4$.

To circumvent the cyclicity constraint on $\mathbb{Z}_M^*/\langle p \rangle$ when M is a power of two, Chen and Han [10] design a linear transformation tailored for thin bootstrapping where each slot stores only an integer. The algorithm is later revised by Geelen and Vercauteren [16]. However, this method still computes the linear transformations as a whole, which means it still suffers from long running time when the number of slots is large.

Since FHE applications over integers typically seek a large number of slots to fully exploit the SIMD property [27,12]. Given that the dimension of the linear transformations is equal to the number of slots, the poor performance of linear transformations with a large dimension in power-of-two cyclotomics greatly limits the flexibility of BGV bootstrapping, resulting in diminished compatibility with the SIMD feature. This may account for why previous works opt for parameters supporting at most 128 slots for BGV bootstrapping in power-of-two cyclotomics [10,11] and why most FHE libraries (except HElib) do not support BGV/BFV bootstrapping. Therefore, accelerating the linear transformations in BGV bootstrapping is crucial if we want to exploit both the NTT efficiency of power-of-two cyclotomics and the SIMD property of BGV.

1.1 Our Techniques and Results

Our basic observation is that the primary component of the linear transformation in BGV bootstrapping can be interpreted as an NTT, and thus can be decomposed into linear sub-transformations based on fast-NTT algorithms (such as the Cooley-Tukey algorithm [13]). This opens up the potential for an accelerated linear transformation in BGV bootstrapping by considering the homomorphic evaluation of these sub-transformations. Although NTT in plaintext has been extensively studied and various fast-NTT algorithms are known, the scope of homomorphic evaluation presents unique challenges. General BGV linear transformations are typically implemented using a combination of fundamental transformations (i.e., one-dimensional linear transformations [19]). The evaluation complexity of a general linear transformation is determined by its specific form. Therefore, to achieve an efficient linear transformation in BGV bootstrapping, it is essential to first ascertain the feasibility of decomposing the NTT into multiple linear sub-transformations that can be evaluated efficiently. This paper addresses this problem by proposing a concrete construction for such a decomposition. Furthermore, we introduce several novel optimizations to both the decomposition and the evaluation of sub-transformations. Our contributions can be summarized as follows.

(1) We provide an explicit framework for homomorphic NTT in BGV bootstrapping by leveraging the algebraic properties of power-of-two cyclotomics. Specifically, we demonstrate that for any power-of-two M and prime $p > 2$, both the NTT and its inverse can be decomposed into one-dimensional linear sub-transformations. These sub-transformations exhibit different forms for different p, as p affects the hypercube structure and the number of non-zero coefficients in each factor of $X^{M/2}+1$. For $p \equiv 1 \bmod 4$, these one-dimensional linear transformations all fall within the MatMul1D type as defined in [19]. Furthermore, we show that, based on the specific vector representation of each slot, the matrix for each one-dimensional linear transformation is tridiagonal, which allows for highly efficient homomorphic evaluation. For $p \equiv 3 \bmod 4$, we demonstrate that all but the first one of these one-dimensional linear transformations are of the MatMul1D type, which can be represented as matrices with $6 \sim 7$ diagonals. For further optimization, we illustrate how we can 'fold' multiple non-zero diagonals of the matrices inside a single slot, thereby producing new tridiagonal matrices that correspond to one-dimensional linear transformations of the BlockMatMul1D type. This leads to reduced running time in most cases.

(2) We propose several further optimizations for the homomorphic evaluation of linear transformations. Firstly, we introduce a modified Baby-Step Giant-Step (BSGS) technique, which accelerates the homomorphic linear transformations under certain conditions. Secondly, we demonstrate that our framework is applicable to both thin and general bootstrapping, each with different optimizations. For thin bootstrapping, where each slot stores an integer, we observe that some sub-transformations can either be omitted or computed on a subfield (or subring) of each slot, thereby reducing the running time. For general bootstrapping, where each slot stores a Galois field/ring element, we reorder the final transformation

that moves slot coefficients from the power basis to the normal basis, resulting in improved performance. Lastly, we show that the level-collapsing method used in CKKS bootstrapping [9,22] can be adapted to our framework, which allows for a trade-off between the time and depth consumption of homomorphic linear transformations.

(3) We implement our approach for both general and thin bootstrapping based on HElib with the optimization in [25]. The parameters have slot numbers ranging from 4096 to 32768. For thin bootstrapping, we reduce the running time of linear transformations in bootstrapping by 7.35$\sim$63 times and obtain a bootstrapping throughput 4.79x$\sim$36.0x that of prior works or the naive approach. For general bootstrapping, the improvement in the running time of linear transformations is 48.9x$\sim$143x, while the improvement in bootstrapping throughput is 28.6x$\sim$66.4x.

1.2 Related Works.

FFT Based Linear Transformations in CKKS Bootstrapping. In [9,22], it was shown that the homomorphic linear transformations in CKKS bootstrapping can be decomposed into FFT-like matrices for acceleration. Our idea can be viewed as an analogue of this approach for BGV bootstrapping. However, the decomposition of linear transformations in BGV bootstrapping into NTT-like matrices is significantly more complex than in CKKS. Firstly, the cyclotomic polynomial $X^{M/2}+1$ splits in $\mathbb{C}$, implying that the linear transformations evaluated during CKKS bootstrapping closely resemble the standard FFT. Conversely, in BGV, $X^{M/2}+1$ can be factorized into binomials or trinomials of degrees greater than one, which correspond to incomplete NTT or incomplete Bruun-like NTT [6]. Secondly, each slot in a CKKS ciphertext stores a scalar value in $\mathbb{C}$, while a slot in BGV may store an element in a Galois field or Galois ring, which can be interpreted as a vector of integers modulo the plaintext modulus. Consequently, the linear transformations are purely inter-slot in CKKS bootstrapping, while they are both inter-slot and intra-slot in BGV bootstrapping. This fact complicates the form of the linear transformations and provides multiple design possibilities. Thirdly, the slots in CKKS always form a one-dimensional vector, while slots in BGV can form a hypercube with multiple dimensions. This further complicates the linear transformations in BGV compared to those in CKKS. Finally, when the plaintext modulus of BGV is a prime power p^r and each slot stores an element in a Galois ring, it remains unexplored whether the factorization of $X^{M/2}+1$ modulo p^r still enables efficient homomorphic NTT. Although NTT in arbitrary algebras has been investigated by Cantor and Kaltofen, it is realized through root adjoining [7], which is infeasible in the FHE setting.

Optimized Digit Removal for Large Plaintext Prime. In BGV bootstrapping, the digit removal procedure is also a computationally expensive component. This is particularly true when facilitating SIMD for power-of-two cyclotomics, where the plaintext prime p scales with the number of slots. For instance, to achieve 2^A slots, p should be at least $2^{A+1}+1$ if $p \equiv 1 \bmod 4$, or at least $2^{A+1}-1$ if $p \equiv 3 \bmod 4$ [26]. As a result, it is necessary to leverage the technique introduced in [25] to expedite the digit removal procedure in BGV bootstrapping with a large p. However, in [25], the powerful basis decomposition method of HElib [21] is employed to compute linear transformations, implying that the linear transformations will dominate the running time of BGV bootstrapping when the slot number is large. Therefore, our approach to accelerate the linear transformations contributes to completing the final piece for efficient BGV bootstrapping for highly-SIMD integer arithmetic in power-of-two cylotomics (e.g., $p = 65537$ with 2^{15} slots for $M = 2^{16}$ cyclotomics).

2 Preliminary

2.1 Notations

- Let $\Phi_M(X)$ represent the M-th cyclotomic polynomial, and let R_q be the quotient ring $\mathbb{Z}_q[X]/(\Phi_M(X))$, where $q \geq 2$ is an integer. The Euler function is denoted by $\varphi(\cdot)$, and thus $\deg(\Phi_M) = \varphi(M)$. This paper primarily focuses on the case where M is a power of two, impling that $\varphi(M) = M/2$ and $\Phi_M(X) = X^{M/2} + 1$.
- Let G be a finite group. The order of an element g in G is denoted by $\mathrm{ord}_G(g)$, and the subgroup generated by elements $g_1, \ldots, g_l$ in G is represented as $\langle g_1, \ldots, g_l \rangle$.
- For positive integers a and b, we denote the set $\{0, 1, \ldots, a-1\}$ as $[a]$, and denote the remainder of a modulo b as $[a]_b \in [b]$. For a set S and an integer a, we denote $a \times S$ for $\{a \cdot s \mid s \in S\}$, $a + S$ for $\{a + s \mid s \in S\}$ and $[S]_a$ for $\{[s]_a \mid s \in S\}$.
- Let $a = \sum_{i=0}^{k-1} a_i 2^i$ be the bit decomposition of a k-bit nonnegative integer a, we define $\mathrm{BitRev}_{k,t}(a) = [a]_{2^t} + \sum_{i=t}^{k-1} a_{k-1-i} 2^i$ for $0 \leq t \leq k$, and $\mathrm{BitRev}'_{k,t}(a) = [a]_{2^t} + a_{k-1}2^{k-1} + \sum_{i=t}^{k-2} a_{k-2-i} 2^i$ for $t \in [k]$. In other words, $\mathrm{BitRev}_{k,t}$ reverses all but the lowest t bits in a, while $\mathrm{BitRev}'_{k,t}$ preserves the highest bit and the lowest t bits in a, reversing all other bits.
- Given an array of size 2^k with elements $a_i, i \in [2^k]$, we define $\mathrm{BR}_{k,t}(a_i) = a_{\mathrm{BitRev}_{k,t}(i)}$ and $\mathrm{BR}'_{k,t}(a_i) = a_{\mathrm{BitRev}'_{k,t}(i)}$. Both $\mathrm{BR}_{k,t}$ and $\mathrm{BR}'_{k,t}$ are order-two permutations on the array.
- All vectors are assumed to be column vectors, and all linear transformations correspond to left-multiplying a column vector by a matrix. For a vector $\mathbf{v}$ of length n, its i-th entry is denoted as $\mathbf{v}[i]$ for $i \in [n]$, and the notation $\mathbf{v}[i +\!: \Delta]$ stands for the Δ-sized subvector $(\mathbf{v}[i], \mathbf{v}[i+1], \ldots, \mathbf{v}[i+\Delta-1])$. For a polynomial $m(x) = \sum_{i=0}^{n-1} m_i x^i$, the notation $m[i +\!: \Delta]$ stands for the coefficient vector $(m_i, m_{i+1}, \ldots, m_{i+\Delta-1})$.

– For an $n \times n$ matrix $\mathbf{N}$, the entry at the i-th row and j-th column is denoted by $\mathbf{N}[i,j]$, with $i,j \in [n]$. The i-th diagonal of $\mathbf{N}$ is the vector whose j-th entry is $\mathbf{N}[j,[i+j]_n]$. Note that the i-th and j-th diagonals coincide if $i \equiv j \bmod n$. Let $\mathbf{I}_n$ be the identity matrix of size n.
– The power basis of R_q consists of X^i for $i \in [\varphi(M)]$. Let $M = M_1 M_2 \dots M_k$ be the factorization of M into prime powers. The powerful basis of R_q consists of $\prod_{i=1}^{k} X_i^{e_i}$, where $X_i = X^{M/M_i}$ and $e_i \in [\varphi(M_i)]$. We note that the powerful basis is identical to the standard basis when M is a power of 2.

2.2 Galois Fields and Rings

Let p be a prime number. The Galois field with characteristic p and cardinality p^d is denoted by $\mathsf{GF}(p^d)$, and the Galois ring with characteristic p^r and cardinality p^{rd} is denoted by $\mathsf{GR}(p^r;d)$. In the special case where $r=1$, it has $\mathsf{GR}(p;d) = \mathsf{GF}(p^d)$. We introduce some conclusions about Galois rings that will be used in subsequent proofs. Refer to [29] for the details of the following conclusions.

Hensel's Lemma. Let f be a monic polynomial in $\mathbb{Z}_{p^r}[X]$, and denote $\bar{f} = f \bmod p \in \mathbb{Z}_p[X]$. Assume that $\bar{f} = g_1 g_2 \dots g_n$, where $g_1, g_2, \dots, g_n \in \mathbb{Z}_p[X]$ are pairwise coprime monic polynomials. Then Hensel's lemma guarantees that there exist pairwise coprime monic polynomials $f_1, f_2, \dots, f_n \in \mathbb{Z}_{p^r}[x]$ such that $f = f_1 f_2 \dots f_n$ and $\bar{f}_i = g_i$ for $1 \le i \le n$.

Hensel's Lemma can be generalized to extension rings. Let f be a monic polynomial in $\mathsf{GR}(p^r;d)[X]$, and denote $\bar{f} = f \bmod p \in \mathsf{GF}(p^d)[X]$. Assume that $\bar{f} = g_1 g_2 \dots g_n \in \mathsf{GF}(p^d)[X]$, where $g_1, g_2, \dots, g_n \in \mathsf{GF}(p^d)[X]$ be pairwise coprime monic polynomials. Then there exist pairwise coprime monic polynomials $f_1, f_2, \dots, f_n \in \mathsf{GR}(p^r;d)[X]$ such that $f = f_1 f_2 \dots f_n$ and $\bar{f}_i = g_i$ for $1 \le i \le n$.

The Group of Units. Assume p is an odd prime number. Let $R = \mathsf{GR}(p^r;d)$ and let R^* denote the group of multiplicative units in R. Then it has $R^* = G_1 \times G_2$, where G_1 is a cyclic group of order $p^d - 1$ and G_2 is a direct product of d cyclic groups each of order p^{r-1}.

Primitive Element. There exists a nonzero element $\gamma \in \mathsf{GR}(p^r;ml)$ such that

a) γ has multiplicative order $p^{ml} - 1$;
b) γ is a root of a basic primitive polynomial[7] $h(X)$ of degree l over $\mathsf{GR}(p^r;m)$, where $h(X)$ divides $X^{p^{ml}-1} - 1$ over $\mathsf{GR}(p^r;m)$;
c) $\mathsf{GR}(p^r;ml) = \mathsf{GR}(p^r;m)[\gamma] = \{a_0 + a_1\gamma + \dots + a_{l-1}\gamma^{l-1} : a_i \in \mathsf{GR}(p^r;m)\}$.

[7] A non-constant monic polynomial $h(X)$ over $\mathsf{GR}(p^r;m)$ is a monic basic primitive polynomial if $\bar{h}(X)$ is a primitive polynomial over $\mathsf{GF}(p^m)$.

Frobenius Automorphism. Let $R = \mathsf{GR}(p^r; m)$ and $R' = \mathsf{GR}(p^r; ml) = R[\gamma]$, where $\gamma \in R'$ is a primitive element. Define a map $\pi : R' \to R'$ by

$$\pi(a_0 + a_1\gamma + \ldots + a_{l-1}\gamma^{l-1}) = a_0 + a_1\gamma^{p^m} + \ldots + a_{l-1}\gamma^{(l-1)p^m}$$

for all $a_0, a_1, \ldots, a_{l-1} \in R$. Then π is an automorphism of R' leaving R fixed elementwise. Moreover, for $\alpha \in R'$, $\pi(\alpha) = \alpha$ if and only if $\alpha \in R$.

Throughout the remainder of this paper, the symbol $\mathcal{E}$ will always denote the Galois ring $\mathsf{GR}(p^r; d)$. Besides, if $\mathsf{GF}(p^d)$ is represented as $\mathbb{Z}_p[X]/f(X)$ for some irreducible polynomial $f(X)$, its power basis is defined as X^i for $i \in [d]$. The power basis of a Galois ring is defined similarly.

2.3 BGV Plaintext Space

The BGV plaintext space is $R_{p^r} = \mathbb{Z}_{p^r}[X]/(\Phi_M(X))$, where p is a prime number, M is coprime to p, and r is a positive integer (known as the Hensel lifting parameter). Let $d = \mathrm{ord}_{\mathbb{Z}_M^*}(p)$. It is known that $\Phi_M(X)$ factorizes into $L = \varphi(M)/d$ irreducible and pairwise coprime monic polynomials of degree d over $\mathbb{Z}_{p^r}$, i.e., $\Phi_M(X) = \prod_{i=0}^{L-1} F_i(X)$. The Chinese Reminder Theorem provides an isomorphism between R_{p^r} and $\prod_{0 \le i < L} \mathbb{Z}_{p^r}[X]/(F_i(X))$. Specifically, let $\eta = X \bmod F_0(X)$ and let $S \subseteq \mathbb{Z}_M^*$ be a set of representatives of $\mathbb{Z}_M^*/\langle p \rangle$, then for any $m(X) \in R_{p^r}$ the isomorphism can be explicitly expressed as

$$\mathtt{Decode}(m(X)) = (m(\eta^{s_0}), \ldots, m(\eta^{s_{L-1}}))_{s_i \in S}\,.$$

Note that $\mathbb{Z}_{p^r}[X]/(F_i(X)) \cong \mathsf{GR}(p^r; d)$. By denoting $\mathcal{E} = \mathsf{GR}(p^r; d)$, `Decode` eventually induces an isomorphism between R_{p^r} and $\mathcal{E}^L$, and the L coordinates of $\mathcal{E}^L$ are referred to as *slots* in the plaintext.

In the context of rotation operations in BGV, S is typically expressed as the products of several generators, i.e., $S = \{\prod_{i=1}^n g_i^{e_i}\}_{e_i \in [L_i]}$, where L_i is the order of g_i in $\mathbb{Z}_M^*/\langle p, g_1, \cdots, g_{i-1} \rangle$. By assigning the index $(e_1, \ldots, e_n)$ to the slot $\prod_{i=1}^n g_i^{e_i}$, the L slots can be organized into an n-dimensional *hypercube*. A *hypercolumn along the s-th dimension* is composed of L_s slots, where e_j remains constant for $j \neq s$ and e_s varies from 0 to $L_s - 1$. It is evident that there are L/L_s hypercolumns in the s-th dimension.

A dimension s is referred to as a *good dimension* if $\mathrm{ord}_{\mathbb{Z}_M^*}(g_s) = L_s$, otherwise, it is termed a *bad dimension*. It is known that we can rotate all the L/L_s hypercolumns along the s-th dimension simultaneously with one Galois automorphism in a good dimension, or two in a bad dimension. Specifically, let ρ_s be the rotation-up-by-one-slot operation along the s-th dimension that moves the slot at index $(e_1, \ldots, e_n)$ to $(e_1, \ldots, e_{s-1}, [e_s - 1]_{L_s}, e_{s+1}, \ldots, e_n)$. Let θ_s be the Galois automorphism that sends $m(X)$ to $m(X^{g_s})$. If this dimension is good, it has $\rho_s = \theta_s$. Otherwise, for $i \in [L_s]$, it has $\rho_s^i(m) = \theta_s^i(m) \cdot \mu_s(i) + \theta_s^{i-L_s}(m) \cdot \mu_s(i)'$ for some constants $\mu_s(i)$ and $\mu_s(i)'$ [19,20]. This rotation operation plays a pivotal role in executing homomorphic linear transformations on the slots.

2.4 Homomorphic Linear transformations

Let $\mathtt{T}$ be a linear transformation from $\mathcal{E}^L$ to $\mathcal{E}^L$. We say that $\mathtt{T}$ is a one-dimensional linear transformation along the s-th dimension if the value in any slot of $\mathtt{T}(\alpha)$ only depends on the slots of the same hypercolumn along the s-th dimension of α. One-dimensional linear transformations have been studied extensively due to their role as fundamental building blocks of arbitrary linear transformations on slots [19].

The one-dimensional transformations fall into two categories. The first type, called MatMul1D in HElib, is the one-dimensional $\mathcal{E}$-linear transformation. Specifically, a MatMul1D transformation $\mathtt{T}$ along the s-th dimension can be expressed as

$$\mathtt{T}(m) = \sum_{i \in [L_s]} \kappa(i)\rho_s^i(m), \text{ for } m \in R_{p^r}, \tag{1}$$

where $\kappa(i) \in R_{p^r}$ are constants determined by $\mathtt{T}$. When considering the restriction of $\mathtt{T}$ on a hypercolumn k along the s-th dimension, it can be represented as a matrix $\mathbf{T}_k \in \mathcal{E}^{L_s \times L_s}$. Besides, $\mathtt{Decode}(\kappa(i))$ is composed of the i-th diagonals of all $\mathbf{T}_k$'s.

The other type, called BlockMatMul1D, is the one-dimensional $\mathbb{Z}_{p^r}$-linear transformation. Specifically, a BlockMatMul1D transformation $\mathtt{T}'$ along the s-th dimension can be expressed as

$$\mathtt{T}'(m) = \sum_{j \in [d]} \sum_{i \in [L_s]} \kappa(i,j)\sigma^j(\rho_s^i(m)), \text{ for } m \in R_{p^r}, \tag{2}$$

where $\kappa(i,j) \in R_{p^r}$ are constants determined by $\mathtt{T}'$, and σ is the Frobenius automorphism. When considering the restriction of $\mathtt{T}'$ on a hypercolumn k along the s-th dimension, it can be represented as an $L_s \times L_s$ matrix $\mathbf{T}'_k$ such that each of its entries is a $\mathbb{Z}_{p^r}$-linear transformation on $\mathcal{E}$. Such an entry can be represented as either a matrix in $\mathbb{Z}_{p^r}^{d \times d}$ or a linearized polynomial $f(v) = \sum_{j \in [d]} a_j \sigma^j(v)$, where $a_j \in \mathcal{E}$. Again, $\mathtt{Decode}(\kappa(i,j))$ is composed of the j-th coefficients of the i-th diagonals in all $\mathbf{T}'_k$'s (in the linearized polynomial form).

For a MatMul1D or BlockMatMul1D type one-dimensional linear transformation $\mathtt{T}$ along the s-th dimension, define $\mathrm{DiagSet}_s(\mathtt{T}) \subseteq [L_s]$ as the union of the sets of the indices of nonzero diagonals in $\mathbf{T}_k$ for $k \in [L/L_s]$, where $\mathbf{T}_k$ is the restriction of $\mathtt{T}$ on a hypercolumn k. For convenience in proof, we relax the definition of DiagSet by allowing $\mathrm{DiagSet}_s(\mathtt{T})$ to include the indices of some zero diagonals. Since $\kappa(i)$ in Equation 1 and $\kappa(i,j)$ in Equation 2 are composed of the i-th diagonals in all $\mathbf{T}_k$, we can replace '$i \in [L_s]$' with '$i \in \mathrm{DiagSet}_s(\mathtt{T})$' by omitting the zero diagonals. Moreover, for two one-dimensional linear transformations $\mathtt{T}$ and $\mathtt{T}'$ on the s-th dimension, their composition satisfies

$$\mathrm{DiagSet}_s(\mathtt{T}' \circ \mathtt{T}) = \{[a+b]_{L_s} \mid a \in \mathrm{DiagSet}_s(\mathtt{T}), b \in \mathrm{DiagSet}_s(\mathtt{T}')\}$$

due to Equation 1 and Equation 2.

Hoisting. When multiple automorphisms need to be computed on the *same* ciphertext, the hoisting technique could be used to significantly speed up the computation [10,19]. In an ordinary automorphism, the decomposition of the ciphertext before re-linearization is the most expensive part because it requires NTTs. When hoisting is applied, the ciphertext is decomposed and moved into the NTT domain in the first step. Utilizing this pre-computed result, we can perform multiple automorphisms on this ciphertext without further decomposition or NTTs.

2.5 BGV Bootstrapping

BGV bootstrapping is categorized into two types, general bootstrapping [18,21] and thin bootstrapping [10]. The general bootstrapping consists of four steps: (1) decryption formula simplification; (2) CoeffToSlot transformation; (3) digit removal; (4) SlotToCoeff. Given $m \in R_{p^r}$, the CoeffToSlot moves the powerful basis coefficients of m into the slots, where each slot is identified as a d-dimension vector space w.r.t. the normal basis of $\mathcal{E}$. In contrast, the SlotToCoeff is almost the inverse of CoeffToSlot, moving the coefficients in slots (w.r.t. the power basis of $\mathcal{E}$) into the powerful basis in R_{p^r}. We omit the descriptions of (1) and (3) because they are not the focus of this work. We can consider a simplified version of CoeffToSlot that homomorphically computes the encoding map $\texttt{Encode}(\cdot) = \texttt{Decode}^{-1}(\cdot)$, which is the most complicated part of CoeffToSlot and only needs to be composed with lightweight transformations to be converted to the actual CoeffToSlot. SlotToCoeff is also simplified as the decoding map $\texttt{Decode}(\cdot)$.

If each slot stores only an integer instead of a Galois ring/field element, the bootstrapping is called a thin bootstrapping. In thin bootstrapping, the steps come in a different order, namely (4)(1)(2)(3). The input ciphertext to SlotToCoeff now encrypts a plaintext whose slots store integers instead of Galois ring elements, which reduces the cost of SlotToCoeff. Since step (1) adds undesired coefficients into the plaintext polynomial, an extra linear map is needed to clear these extra coefficients. This map can be performed after CoeffToSlot in general cyclotomics [21] or before CoeffToSlot in power-of-two cyclotomics [10].

2.6 Number Theoretic Transform (NTT)

In this paper, we focus on the NTT mapping which maps $m \in R_{p^r}$ to $(m \bmod F_0(X), \ldots, m \bmod F_{L-1}(X)) \in \prod_{i \in [L]} \mathbb{Z}_{p^r}[X]/F_i(X)$, where $F_i(X)$'s are the irreducible factors of $\Phi_M(X)$ defined in Section 2.3. The inverse NTT (iNTT) is defined as the inverse of this map. There has been plenty of research about the NTT/iNTT on the plaintext [8], and various fast NTT algorithms have been proposed, such as Cooley-Tukey [13] and Bruun [6]. These algorithms typically decompose NTT/iNTT into multiple layers to speed up the computation. We do not delve into their details here, as we will present explicit decompositions of NTT/iNTT within the framework of BGV linear transformations.

3 The Decomposition of Linear Transformations

As discussed previously, this section focuses on the decomposition of `Decode` and `Encode`. Let $\Phi_M(X) = \prod_{i=0}^{L-1} F_i(X)$, where $F_i(X)$ is the minimal polynomial of η^{s_i} and $\{s_i\}_{i\in[L]} \subseteq \mathbb{Z}_M^*$ is a set of representatives of $\mathbb{Z}_M^*/\langle p\rangle$. Then `Decode` can be decomposed into two sub-maps `Red` and `Eval`, i.e., `Decode` = `Eval` ∘ `Red`, where `Red` is an NTT map from R_{p^r} to $\prod_{i\in[L]} \mathbb{Z}_{p^r}[X]/F_i(X)$ such that

$$\mathtt{Red}(m) = (m \bmod F_0, m \bmod F_1, \ldots, m \bmod F_{L-1}), \text{ for } m \in R_{p^r},$$

and `Eval` is a map from $\prod_{i\in[L]} \mathbb{Z}_{p^r}[X]/F_i(X)$ to $\mathcal{E}^L$ such that

$$\mathtt{Eval}(m_0(X), \ldots, m_{L-1}(X)) = (m_0(\eta^{s_0}), \ldots, m_{L-1}(\eta^{s_{L-1}})).$$

Both `Red` and `Eval` are $\mathbb{Z}_{p^r}$-linear transformations, and they can be represented as matrices in $(\mathbb{Z}_{p^r}^{d\times d})^{L\times L}$ by identifying the input and output as vectors in $(\mathbb{Z}_{p^r}^d)^L$ via coefficient embedding. Specifically, for $m(X) \in R_{p^r}$, the i-th entry is the vector $m[id +\colon d]$ for $i \in [L]$. For $(m_i(X))_{i\in[L]} \in \prod_{i\in[L]} \mathbb{Z}_{p^r}[X]/F_i(X)$, the i-th entry is the coefficient vector of $m_i(X)$. For $\mathcal{E}^L$, the i-th entry is the coefficient vector of the i-th slot with respect to the power basis of $\mathcal{E} = \mathbb{Z}_{p^r}[X]/F_0(X)$. When we represent a homomorphic linear transformation as a matrix, each of its entries is an element in $\mathbb{Z}_{p^r}^{d\times d}$.

Clearly `Eval` is a BlockMatMul1D type one-dimensional linear transformation such that its main diagonal is the only nonzero diagonal (in terms of an $L \times L$ block matrix). Thus `Eval` and $\mathtt{Eval}^{-1}$ can be computed by evaluating a linearized polynomial in Equation 2 with $i = 0$. In the remainder of this section, we focus on the decomposition of `Red` (and $\mathtt{Red}^{-1}$) into linear sub-transformations for power-of-two cyclotomics.

In the case when M is a power of two, it is known that $\mathbb{Z}_M^* = \langle -1, 5\rangle \cong \mathbb{Z}_2 \times \mathbb{Z}_{M/4}$. If $p \equiv 1 \bmod 4$, $\mathbb{Z}_M^*/\langle p\rangle = \langle -1, 5\rangle \cong \mathbb{Z}_2 \times \mathbb{Z}_{M/(4d)}$, implying a 2 by $\frac{M}{4d}$ sized hypercube generated by $g_1 = -1, g_2 = 5$. If $p \equiv 3 \bmod 4$, $\mathbb{Z}_M^*/\langle p\rangle = \langle 5\rangle \cong \mathbb{Z}_{M/(2d)}$. The hypercube has a single generator $g_1 = 5$ and collapses into a single dimension of size $\frac{M}{2d}$. We call the dimension generated by 5 (in both cases of p) *the major dimension* and denote its size as D, i.e., $D = L/2 = M/(4d)$ for $p = 1 \bmod 4$ and $D = L = M/(2d)$ for $p \equiv 3 \bmod 4$. We call the dimension generated by -1 (in case of $p \equiv 1 \bmod 4$) *the minor dimension*, which has a size of 2. We omit the subscript s in $\rho_s, \theta_s, \mu_s, \mu'_s, \mathrm{DiagSet}_s$ when they are related to the one-dimensional linear transformations on the major dimension. The main result of this section can be summarized as follows.

Theorem 1. (1) *If $p \equiv 1 \bmod 4$, we have the decomposition*

$$\mathtt{Red}^{-1} = \mathtt{BR}'_{\log_2(2dD),\log_2(d)} \circ \mathtt{Red}_{\mathrm{BR}}^{-1} \textit{ and}$$

$$\mathtt{Red}_{\mathrm{BR}}^{-1} = \mathtt{N}_{\log_2(D)+1} \circ \ldots \circ \mathtt{N}_1,$$

where $\mathtt{BR}'$ is interpreted as a permutation on $(\mathbb{Z}_{p^r}^d)^{2D}$ in the natural manner. For $j \in [1, \log_2(D)]$, both $\mathtt{N}_j$ and $\mathtt{N}_j^{-1}$ are MatMul1D transformations on the major *dimension with nonzero diagonals indexed by $2^{-j}D \times \{-1,0,1\}$. $\mathtt{N}_{\log_2(D)+1}$ and its inverse are MatMul1D transformations on the minor dimension.*

(2) *If $p \equiv 3 \bmod 4$, we have the Bruun style decomposition*

$$\mathtt{Red}^{-1} = \mathtt{BR}_{\log_2(dD),\log_2(d)} \circ \mathtt{Red}_{\mathrm{BR}}^{-1} \text{ and}$$

$$\mathtt{Red}_{\mathrm{BR}}^{-1} = \mathtt{N}_{\log_2(D)} \circ \ldots \circ \mathtt{N}_1,$$

where $\mathtt{N}_1$ and $\mathtt{N}_1^{-1}$ are BlockMatMul1D transformations with nonzero diagonals indexed by $D/2 \times \{-1,0,1\}$. For $j \in [2, \log_2(D)]$, $\mathtt{N}_j$ is a MatMul1D transformation with nonzero diagonals indexed by $2^{-j}D \times [-3,3]$, and $\mathtt{N}_j^{-1}$ is a MatMul1D transformation with nonzero diagonals indexed by $2^{-j}D \times [-3,2]$. Alternatively, $\mathtt{Red}^{-1}$ can also be decomposed in a Radix-2 style into

$$\mathtt{Red}^{-1} = \mathtt{BR}_{\log_2(dD),\log_2(d)-1} \circ \mathtt{Red}_{\mathrm{BR}}'^{-1} \text{ and}$$

$$\mathtt{Red}_{\mathrm{BR}}'^{-1} = \mathtt{N}'_{\log_2(D)} \circ \ldots \circ \mathtt{N}'_1,$$

where both $\mathtt{N}'_j$ and $\mathtt{N}_j'^{-1}$ are BlockMatMul1D transformations with nonzero diagonals indexed by $2^{-j}D \times \{-1,0,1\}$ for $j \in [1, \log_2(D)]$.

Recall that for a one-dimensional linear transformation $\mathtt{N}$ along the s-th dimension, the number of rotations required to evaluate it equals $|\mathrm{DiagSet}(\mathtt{N})|$. According to Theorem 1, both $|\mathrm{DiagSet}(\mathtt{N}_j)|$ and $|\mathrm{DiagSet}(\mathtt{N}_j^{-1})|$ are small (usually two to three) because they have only a few diagonals. Therefore, the computation time for the linear transformations in bootstrapping can be significantly reduced by utilizing the decomposition presented in Theorem 1. In the subsequent discussion, we provide the derivation of Theorem 1 for two cases of p. Moreover, in Section 3.1 and Section 3.2 we make the assumption that $r = 1$ in the plaintext modulus, implying that each slot corresponds to the Galois field $\mathsf{GF}(p^d)$. The general case where $r > 1$ (corresponding to the Galois ring case) will be addressed in Section 3.3.

3.1 The Case of $p \equiv 1 \bmod 4$

In this case, we can select the set of representatives $\{s_i\}_{i\in[L]}$ such that $s_{e_1 D + e_2} = (-1)^{e_1} 5^{e_2}$ for $e_1 \in [2], e_2 \in [D]$, which constructs an arrangement of the slots into the hypercube. We note that the minor dimension is always good, while the major dimension is good whenever $p \equiv 1 \bmod M$. By [26], it has $\Phi_M(X) = \prod_{i \in \mathbb{Z}_{4D}^*} (X^d - \zeta^i)$ over $\mathbb{Z}_p$, where $\zeta \in \mathbb{Z}_p$ is a primitive $4D$-th root of unity and each factor is irreducible over $\mathbb{Z}_p$. Without loss of generality, we can assume that $F_0(X) = X^d - \zeta$, which leads to $F_i(X) = X^d - \zeta^{s_i}$ for $i \in [L]$. To begin with, we prove the following lemma.

Lemma 1. *Let $F_i^{(0)} = F_i(X)$ for $i \in [L]$, and $F_i^{(j)} = F_i^{(j-1)} F_{i+2^{-j}D}^{(j-1)}$ for $1 \le j \le \log_2(D)$ and $i \in [0, 2^{-j}D) \cup [D, D + 2^{-j}D)$, then it has*

$$F_i^{(j)} = X^{d \cdot 2^j} - \zeta^{s_i \cdot 2^j}, \text{ for } j \in [0, \log_2(D)], i \in [0, 2^{-j}D) \cup [D, D + 2^{-j}D).$$

Proof. Clearly, the statement is true for $j = 0$. Now let $1 \leq j \leq \log_2(D)$ and suppose the statement holds for $j-1$ and $i \in [0, 2^{-(j-1)}D) \cup [D, D + 2^{-(j-1)}D)$. By the definition of $F_i^{(j)}$ it has

$$F_i^{(j)} = F_i^{(j-1)} F_{i+2^{-j}D}^{(j-1)} = (X^{d \cdot 2^{j-1}} - \zeta^{s_i \cdot 2^{j-1}})(X^{d \cdot 2^{j-1}} - \zeta^{s_{i+2^{-j}D} \cdot 2^{j-1}})$$

for $i \in [0, 2^{-j}D) \cup [D, D + 2^{-j}D)$. Denote $i = e_1 D + e_2$ for $0 \leq e_1 \leq 1$ and $0 \leq e_2 < 2^{-j}D$, then $s_i = (-1)^{e_1} 5^{e_2}$ and $s_{i+2^{-j}D} = (-1)^{e_1} 5^{e_2 + 2^{-j}D}$. Since ζ is a primitive $4D$-th root of unity and $5^{2^{-j}D} \cdot 2^{j-1} \equiv 2D + 2^{j-1} \bmod 4D$, we have $\zeta^{s_{i+2^{-j}D} \cdot 2^{j-1}} = -\zeta^{s_i \cdot 2^{j-1}}$. Then it follows directly that $F_i^{(j)} = X^{d \cdot 2^j} - \zeta^{s_i \cdot 2^j}$. $\square$

In addition, we denote $F_0^{(\log_2(D)+1)} = \prod_{i \in [2D]} F_i^{(0)} = \Phi_M(X)$.

The Definition of $\mathtt{N}_j$. Suppose $m \in R_{p^r}$, then $\mathtt{N}_j$ can be roughly viewed as the linear transformation that maps $(m \bmod F_i^{(j-1)})_{i \in I_{j-1}}$ to $(m \bmod F_i^{(j)})_{i \in I_j}$, where I_j is the range of i defined in Lemma 1. For the specific definition of $\mathtt{N}_j$, we need to handle the bit-reversal phenomenon to design matrices that can be homomorphic evaluated efficiently. In our case, the bit-reversal primarily arises due to the slots occupied by the two factors that combine into $F_i^{(j)}$ are in an interleaving order. As an example, we illustrate the bit-reversal phenomenon in the computation of $m \bmod F_i^{(2)}$ from $m \bmod F_i^{(1)}$ and $m \bmod F_{i+D/4}^{(1)}$ in Figure 1. Taking this into consideration, we first define vectors $\boldsymbol{\alpha}_j \in (\mathbb{Z}_{p^r}^d)^L$ for $0 \leq j \leq \log_2(D) + 1$ as follows. The vector $\boldsymbol{\alpha}_0$ corresponds to $\alpha = \mathtt{Red}(m) \in \mathcal{E}^L$. For $1 \leq j \leq \log_2(D)$, we define $\boldsymbol{\alpha}_j$ such that

$$\boldsymbol{\alpha}_j[i + k \cdot 2^{-j}D] = (m \bmod F_i^{(j)})[\mathrm{BitRev}_{j,0}(k) \cdot d +: d]$$

for $i \in [0, 2^{-j}D) \cup [D, D + 2^{-j}D)$, $k \in [2^j]$. For $j = \log_2(D) + 1$, we define

$$\boldsymbol{\alpha}_{\log_2(D)+1}[k] = m[\mathrm{BitRev}'_{\log_2(D)+1,0}(k) \cdot d +: d]$$

for $k \in [2D]$.

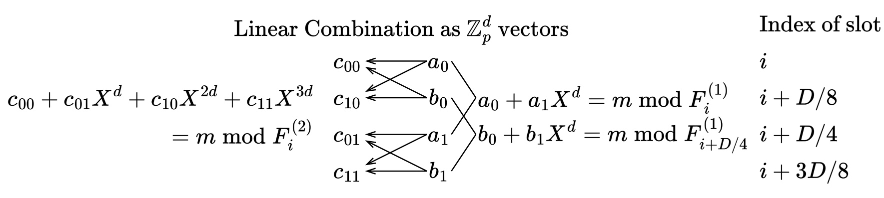

Fig. 1. An example of the butterfly structures in $\mathtt{Red}_{\mathrm{BR}}^{-1}$ that leads to bit-reversal. a_i,b_i and c_{ij} are degree $d-1$ polynomials in $\mathbb{Z}_p[X]$.

For $1 \leq j \leq \log_2(D) + 1$, we define $\mathtt{N}_j$ as the linear transformation that maps $\boldsymbol{\alpha}_{j-1}$ to $\boldsymbol{\alpha}_j$, where the coefficients of m are regarded as independent variables.

Denote $\texttt{Red}_{\mathrm{BR}}^{-1} = \texttt{N}_{\log_2(D)+1} \circ \ldots \circ \texttt{N}_1$, then it can be readily checked that

$$\mathrm{BR}'_{\log_2(2dD),\log_2(d)}(\texttt{Red}_{\mathrm{BR}}^{-1}(\alpha))) = m.$$

Notably, the output of $\texttt{Red}_{\mathrm{BR}}^{-1}(\alpha)$ is a permutated version of m's coefficients, which is a common phenomenon in fast NTT algorithms. As in [9,22], we will not reorder the slots into their ordinary order by computing the inverse permutation homomorphically. Instead, we directly pass the output of $\texttt{Red}_{\mathrm{BR}}$ and $\texttt{Red}_{\mathrm{BR}}^{-1}$ to the next stage of bootstrapping. This will not affect the correctness of bootstrapping, similar to the observations in previous works on CKKS bootstrapping. This is because: (1) the digit removal step is performed in a SIMD manner and is insensitive to the order of the values in the slots; (2) the coefficients in each slot remain as a whole group during the permutation, which makes it possible to repack the output ciphertexts of digit removal.

Let $\mathbf{N}_j \in (\mathbb{Z}_{p^r}^{d\times d})^{L\times L}$ denote the matrix corresponding to $\texttt{N}_j$. In the following lemma, we discuss the structure of the $\mathbf{N}_j$'s. An example illustrating the $\mathbf{N}_j$'s for $D = 4$ is provided in Figure 2 for a better understanding.

$$\left[\begin{array}{cccc:cccc} * & & * & & & & & \\ & * & & * & & & & \\ * & & * & & & & & \\ & * & & * & & & & \\ \hdashline & & & & * & & * & \\ & & & & & * & & * \\ & & & & * & & * & \\ & & & & & * & & * \end{array}\right] \begin{bmatrix} *, F_0^{(0)} \\ *, F_1^{(0)} \\ *, F_2^{(0)} \\ *, F_3^{(0)} \\ \hdashline *, F_4^{(0)} \\ *, F_5^{(0)} \\ *, F_6^{(0)} \\ *, F_7^{(0)} \end{bmatrix} = \begin{bmatrix} 0*, F_0^{(1)} \\ 0*, F_1^{(1)} \\ 1*, F_0^{(1)} \\ 1*, F_1^{(1)} \\ \hdashline 0*, F_4^{(1)} \\ 0*, F_5^{(1)} \\ 1*, F_4^{(1)} \\ 1*, F_5^{(1)} \end{bmatrix}, \quad \left[\begin{array}{cccc:cccc} * & * & & & & & & \\ * & * & & & & & & \\ & & * & * & & & & \\ & & * & * & & & & \\ \hdashline & & & & * & * & & \\ & & & & * & * & & \\ & & & & & & * & * \\ & & & & & & * & * \end{array}\right] \begin{bmatrix} 0*, F_0^{(1)} \\ 0*, F_1^{(1)} \\ 1*, F_0^{(1)} \\ 1*, F_1^{(1)} \\ \hdashline 0*, F_4^{(1)} \\ 0*, F_5^{(1)} \\ 1*, F_4^{(1)} \\ 1*, F_5^{(1)} \end{bmatrix} = \begin{bmatrix} 00*, F_0^{(2)} \\ 10*, F_0^{(2)} \\ 01*, F_0^{(2)} \\ 11*, F_0^{(2)} \\ \hdashline 00*, F_4^{(2)} \\ 10*, F_4^{(2)} \\ 01*, F_4^{(2)} \\ 11*, F_4^{(2)} \end{bmatrix}$$

$$\left[\begin{array}{cccc:cccc} * & & & & * & & & \\ & * & & & & * & & \\ & & * & & & & * & \\ & & & * & & & & * \\ \hdashline * & & & & * & & & \\ & * & & & & * & & \\ & & * & & & & * & \\ & & & * & & & & * \end{array}\right] \begin{bmatrix} 00*, F_0^{(2)} \\ 10*, F_0^{(2)} \\ 01*, F_0^{(2)} \\ 11*, F_0^{(2)} \\ \hdashline 00*, F_4^{(2)} \\ 10*, F_4^{(2)} \\ 01*, F_4^{(2)} \\ 11*, F_4^{(2)} \end{bmatrix} = \begin{bmatrix} 000*, F_0^{(3)} \\ 010*, F_0^{(3)} \\ 001*, F_0^{(3)} \\ 011*, F_0^{(3)} \\ \hdashline 100*, F_0^{(3)} \\ 110*, F_0^{(3)} \\ 101*, F_0^{(3)} \\ 111*, F_0^{(3)} \end{bmatrix}$$

Fig. 2. An illustration of $\texttt{Red}_{\mathrm{BR}}^{-1}$ for $D = 4$ and $p \equiv 1 \bmod 4$. A '$*$' in matrices stands for a nonzero entry that is a multiple of $\mathbf{I}_d$, while a '$*$' in the vectors means $\log_2(d)$ bits ranging from all zeros to all ones. Each slot stores part of the coefficients of m mod $F_i^{(j)}$. The (binary format of) indices of the coefficients are displayed along with the corresponding $F_i^{(j)}$. E.g., '$01*, F_0^{(2)}$' means that this slot stores $(m \bmod F_0^{(2)})[d +\!: d]$.

Lemma 2. (1) *For* $j \in [1, \log_2(D)]$, $\mathbf{N}_j$ *can be viewed as a* $2^j \times 2^j$ *diagonal block matrix. Each block has a size of* $2^{-j+1}D \times 2^{-j+1}D$*, which has three non-zero diagonals indexed by* $2^{-j}D \times \{-1, 0, 1\}$.

(2) *When viewed as an* $L \times L$ *block matrix,* $\mathbf{N}_{\log_2(D)+1}$ *has three non-zero diagonals indexed by* $D \times \{-1, 0, 1\}$*.*

Besides, for $j \in [1, \log_2(D)+1]$*, all non-zero entries of* $\mathbf{N}_j$ *in* $\mathbb{Z}_{p^r}^{d\times d}$ *are multiples of* $\mathbf{I}_d$*. All the above properties also hold for* $\mathbf{N}_j^{-1}$*.*

Proof. For a fixed $j \in [1, \log_2(D)]$, let $i \in [0, 2^{-j}D) \bigcup [D, D + 2^{-j}D)$. For $k \in [2^{j-1}]$, let

$$\mathbf{u}' = (m \bmod F_i^{(j)})[kd +: d] \qquad \mathbf{u} = (m \bmod F_i^{(j-1)})[kd +: d]$$
$$\mathbf{v}' = (m \bmod F_i^{(j)})[2^{j-1}d + kd +: d] \qquad \mathbf{v} = (m \bmod F_{i+2^{-j}D}^{(j-1)})[kd +: d].$$

By traversing k and i, $\mathbf{u}, \mathbf{v}$ and $\mathbf{u}', \mathbf{v}'$ cover all the inputs and outputs of $\mathbf{N}_j$. According to Lemma 1, $F_i^{(j)} = F_i^{(j-1)} F_{i+2^{-j}D}^{(j-1)}$ and $F_i^{(j-1)} = X^{2^{j-1}d} - a_{i,j-1}$, $F_{i+2^{-j}D}^{(j-1)} = X^{2^{j-1}d} + a_{i,j-1}$ for some $a_{i,j-1} \in \mathbb{Z}_p$, thus it can be deduced that

$$\begin{cases} \mathbf{u}' & = (\mathbf{u} + \mathbf{v})/2 \\ \mathbf{v}' & = (\mathbf{u} - \mathbf{v})/(2a_{i,j-1}) \end{cases}, \begin{cases} \mathbf{u} & = \mathbf{u}' + a_{i,j-1}\mathbf{v}' \\ \mathbf{v} & = \mathbf{u}' - a_{i,j-1}\mathbf{v}' \end{cases}.$$

Using the definition of $\boldsymbol{\alpha}_j$ and $\boldsymbol{\alpha}_{j-1}$, the index of $\mathbf{u}'$ in $\boldsymbol{\alpha}_j$ and the index of $\mathbf{u}$ in $\boldsymbol{\alpha}_{j-1}$ are both $l = i + \mathrm{BitRev}_{j,0}(k) \cdot 2^{-j}D$. $\mathbf{v}'$ and $\mathbf{v}$ also have the identical index of $h = i + 2^{-j}D + \mathrm{BitRev}_{j,0}(k) \cdot 2^{-j}D$. Thus, the $\mathbb{Z}_p$-linear combinations of $\mathbf{u}, \mathbf{v}$ into $\mathbf{u}', \mathbf{v}'$ correspond to the following 2×2 submatrix in $\mathbf{N}_j$

$$\begin{bmatrix} \mathbf{N}_j[l,l] & \mathbf{N}_j[l,h] \\ \mathbf{N}_j[h,l] & \mathbf{N}_j[h,h] \end{bmatrix},$$

where each entry is a multiple of $\mathbf{I}_d$. Let $i = e_1 D + e_2$ for $e_1 \in [2]$ and $e_2 \in [2^{-j}D]$, traversing e_2 for a fixed value of the pair (e_1, k) will extend the submatrix above into a $2^{-j+1}D$-sized diagonal block of $\mathbf{N}_j$. As indicated by the indices of $\mathbf{u}, \mathbf{v}, \mathbf{u}', \mathbf{v}'$ in $\boldsymbol{\alpha}_j$ and $\boldsymbol{\alpha}_{j-1}$, each diagonal block has three nonzero diagonals indexed as $\{0, \pm(l-h)\} = 2^{-j}D \times \{-1, 0, 1\}$. The structure of $\mathbf{N}_j^{-1}$ can be deduced similarly.

Concerning $\mathbf{N}_{\log_2(D)+1}$, for $k \in [D], i = 0, j = \log_2(D)+1$, we have

$$\mathbf{u}' = (m \bmod F_0^{(j)})[kd +: d] \qquad \mathbf{u} = (m \bmod F_0^{(j-1)})[kd +: d]$$
$$\mathbf{v}' = (m \bmod F_0^{(j)})[Dd + kd +: d] \qquad \mathbf{v} = (m \bmod F_D^{(j-1)})[kd +: d],$$

where $\mathbf{u}', \mathbf{u}$ share the same index $\mathrm{BitRev}'_{j,0}(k)$ while $\mathbf{v}', \mathbf{v}'$ share the same index $\mathrm{BitRev}'_{j,0}(k) + D$. The remaining proof is similar to the case of $j \in [1, \log_2(D)]$. □

Proof of (1) in Theorem 1. According to Lemma 2, for $j \in [1, \log_2(D)]$, $\mathbf{N}_j$ and $\mathbf{N}_j^{-1}$ can be viewed as

$$\begin{bmatrix} \mathbf{A}_0 & 0 \\ 0 & \mathbf{A}_1 \end{bmatrix},$$

where $\mathbf{A}_0$ and $\mathbf{A}_1$ are $D \times D$ matrices, and $\mathbf{A}_t$ is a linear transformation on the t-th hypercolumn of the major dimension for $0 \le t \le 1$. Thus $\mathbf{N}_j$ and $\mathbf{N}_j^{-1}$ are linear transformations on the major dimension

For $\mathbf{N}_{\log_2(D)+1}$ and its inverse, the t-th hypercolumn of the minor dimension consists of the t-th and $(t+D)$-th slot, where $t \in [D]$. The 2×2 submatrix

$$\begin{bmatrix} \mathbf{N}_{\log_2(D)+1}[t,t] & \mathbf{N}_{\log_2(D)+1}[t,t+D] \\ \mathbf{N}_{\log_2(D)+1}[t,t+D] & \mathbf{N}_{\log_2(D)+1}[t+D,t+D] \end{bmatrix}$$

is a linear transformation on the t-th hypercolumn of the minor dimension. Thus both $\mathbf{N}_{\log_2(D)+1}$ and its inverse are linear transformations on the minor dimension.

For $j \in [1, \log_2(D)+1]$, $\mathbf{N}_j$ is a $\mathsf{MatMul1D}$ transformation because each entry of $\mathbf{N}_j$ is a multiple of $\mathbf{I}_d$. The indices of nonzero diagonals in $\mathbf{N}_j$ and $\mathbf{N}_j^{-1}$ follow directly from Lemma 2.

3.2 The Case of $p \equiv 3 \bmod 4$

In this case, we have $s_{e_1} = 5^{e_1}$ for $e_1 \in [D]$, and the only dimension in the hypercube is always bad. According to [26], $\phi_M(X)$ factors into trinomials for $d \ge 2$, i.e.,

$$\Phi_M(X) = \prod_{i \in \mathbb{Z}_{4D}^*/\langle p \rangle} (X^d - (\zeta^i + \zeta^{ip})X^{d/2} + \zeta^{i(p+1)}),$$

where $\zeta \in \mathsf{GF}(p^2)$ is a primitive $4D$-th root of unity, and each factor is an irreducible polynomial in $\mathbb{Z}_p[X]$. Without loss of generality, we can assume that $F_0(X) = X^d - (\zeta + \zeta^p)X^{d/2} + \zeta^{p+1}$, which leads to $F_i(X) = X^d - (\zeta^{s_i} + \zeta^{s_i p})X^{d/2} + \zeta^{s_i(p+1)}$ for $i \in [D]$. Similarly, we can prove the following lemma.

Lemma 3. *Let $F_i^{(0)} = F_i$ for $i \in [D]$, and $F_i^{(j)} = F_i^{(j-1)} F_{i+2^{-j}D}^{(j-1)}$ for $1 \le j \le \log_2(D)$, $i \in [2^{-j}D]$. Then it has*

$$F_i^{(j)} = X^{2^j d} - (\zeta^{2^j \cdot s_i} + \zeta^{2^j \cdot s_i p})X^{2^{j-1} d} + \zeta^{2^j \cdot s_i(p+1)},$$

for $0 \le j \le \log_2(D)$ and $i \in [2^{-j}D]$. Moreover, the middle term is nonzero except for $j = \log_2(D)$.

Proof. Clearly, the statement is true for $j = 0$. Now let $1 \le j \le \log_2(D)$ and suppose the statement holds for $j-1$. Similar to Lemma 1, it can be proved that $F_{i+2^{-j}D}^{(j-1)} = X^{2^{j-1} d} + (\zeta^{2^{j-1} \cdot s_i} + \zeta^{2^{j-1} \cdot s_i p})X^{2^{j-1-1} d} + \zeta^{2^{j-1} \cdot s_i(p+1)}$. Thus for $i \in [2^{-j}D]$ we have

$$\begin{aligned} F_i^{(j)} &= F_i^{(j-1)} F_{i+2^{-j}D}^{(j-1)} \\ &= (X^{2^{j-1} d} - (\zeta^{2^{j-1} \cdot s_i} + \zeta^{2^{j-1} \cdot s_i p})X^{2^{j-1-1} d} + \zeta^{2^{j-1} \cdot s_i(p+1)}) \\ &\quad \times (X^{2^{j-1} d} + (\zeta^{2^{j-1} \cdot s_i} + \zeta^{2^{j-1} \cdot s_i p})X^{2^{j-1-1} d} + \zeta^{2^{j-1} \cdot s_i(p+1)}) \\ &= X^{2^j d} - (\zeta^{2^j \cdot s_i} + \zeta^{2^j \cdot s_i p})X^{2^{j-1} d} + \zeta^{2^j \cdot s_i(p+1)}. \end{aligned}$$

For the middle term, $\zeta^{2^j \cdot s_i} + \zeta^{2^j \cdot s_i p} = 0 \iff \zeta^{2^j \cdot s_i (p-1)} = -1$. Since $s_i = 5^i$ and ζ is a primitive $4D$-th root of unity, this condition is equivalent to $2^j \cdot 5^i(p-1) \equiv 2D \bmod 4D$. Thus for $j < \log_2(D)$, the maximum power of two that divides $2^j \cdot 5^i(p-1)$ is $2^{j+1} < 2D$, which implies that the middle term is nonzero. For $j = \log_2(D)$, it can be verified that $D \cdot 5^i(p-1) \equiv 2D \bmod 4D$, which implies that the middle term is zero. $\square$

The Definition of $\mathtt{N}_j$. Suppose $m \in R_{p^r}$, we first define vectors $\boldsymbol{\alpha}_j \in (\mathbb{Z}_{p^r}^d)^L$ for $0 \le j \le \log_2(D)$ as follows. The vector $\boldsymbol{\alpha}_0$ corresponds to $\alpha = \mathtt{Red}(m) \in \mathcal{E}^L$. For $1 \le j \le \log_2(D)$, we define $\boldsymbol{\alpha}_j$ such that

$$\boldsymbol{\alpha}_j[i + k \cdot 2^{-j} D] = (m \bmod F_i^{(j)})[\mathrm{BitRev}_{j,0}(k) \cdot d +\!: d]$$

for $i \in [2^{-j}D], k \in [2^j]$.

For $1 \le j \le \log_2(D)$, we define $\mathtt{N}_j$ as the linear transformation that maps $\boldsymbol{\alpha}_{j-1}$ to $\boldsymbol{\alpha}_j$. Denote $\mathtt{Red}_{\mathrm{BR}}^{-1} = \mathtt{N}_{\log_2(D)} \circ \ldots \circ \mathtt{N}_1$, then it can be checked that

$$\mathtt{BR}_{\log_2(2dD), \log_2(d)}(\mathtt{Red}_{\mathrm{BR}}^{-1}(\alpha))) = m.$$

In contrast to the case of $p \equiv 1 \bmod 4$, the fact the $F_i^{(j)}$'s are trinomials complicates the butterfly structure, turning its outputs from linear combinations of two terms into linear combinations of four terms. For example, given two polynomials $f_0(X) = X^{2k} + sX^k + t$ and $f_1(X) = X^{2k} - sX^k + t$ of degree $2k$, let $l + hX^k \in \mathbb{Z}_p[X]/f_0(X)$ and $l' + h'X^K \in \mathbb{Z}_p[X]/f_1(X)$, where $s, t \in \mathbb{Z}_p$ and $l, h, l', h' \in \mathbb{Z}_p[X]$ with degrees less than k. Denote the polynomial reconstructed from $l + hX^k$ and $l' + h'X^k$ as $a_{00} + a_{01}X^k + a_{10}X^{2k} + a_{11}X^{3k} \in \mathbb{Z}_p[X]/(f_1(X)f_2(X))$, where $a_{00}, \ldots, a_{11}$ are polynomials with degree less than k. Then we have the following *Bruun butterfly* structure, where '$*$' represents a non-zero entry in $\mathbb{Z}_p$.

$$\begin{bmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ & * & & * \\ * & & * & \end{bmatrix} \times \begin{bmatrix} l \\ h \\ l' \\ h' \end{bmatrix}, \quad \begin{bmatrix} l \\ h \\ l' \\ h' \end{bmatrix} = \begin{bmatrix} * & & * & * \\ & * & * & * \\ * & & * & * \\ & * & * & * \end{bmatrix} \times \begin{bmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{bmatrix}. \tag{3}$$

In the first layer of $\mathtt{Red}_{\mathrm{BR}}^{-1}$, the i-th Bruun butterfly has two input slots $\boldsymbol{\alpha}_0[i]$ and $\boldsymbol{\alpha}_0[i + D/2]$, where the former stores l and h while the latter stores l' and h'. The output of this butterfly is stored in $\boldsymbol{\alpha}_1[i]$ and $\boldsymbol{\alpha}_1[i + D/2]$.

The natural approach is to store the lower coefficients a_{00} and a_{01} in $\boldsymbol{\alpha}_1[i]$, while the higher ones a_{10} and a_{11} are stored in $\boldsymbol{\alpha}_1[i+D]$, i.e., in a non-bit-reversed order. In this case, for $j \ge 2$, the four inputs to each Bruun butterfly in $\mathtt{N}_j$ lie in four distinct slots, which means each entry in $\boldsymbol{\alpha}_j$ are $\mathbb{Z}_p$-linear combinations of entries in $\boldsymbol{\alpha}_{j-1}$ and each entry of $\mathbf{N}_j$ is a multiple of $\mathbf{I}_d$. We call this way of constructing $\mathtt{N}_j$ as the *Bruun style*. An example for $D = 8$ is presented in Figure 3 for better illustration. The formal statements about the structure of $\mathtt{N}_j$ are given in Lemma 4 and proved in Supplementary Material A.

$$
\begin{bmatrix}
\# & & & & \# & & & \\
& \# & & & & \# & & \\
& & \# & & & & \# & \\
& & & \# & & & & \# \\
\# & & & & \# & & & \\
& \# & & & & \# & & \\
& & \# & & & & \# & \\
& & & \# & & & & \#
\end{bmatrix}
\begin{bmatrix}
*, F_0^{(0)} \\ *, F_1^{(0)} \\ *, F_2^{(0)} \\ *, F_3^{(0)} \\ *, F_4^{(0)} \\ *, F_5^{(0)} \\ *, F_6^{(0)} \\ *, F_7^{(0)}
\end{bmatrix}
=
\begin{bmatrix}
0*, F_0^{(1)} \\ 0*, F_1^{(1)} \\ 0*, F_2^{(1)} \\ 0*, F_3^{(1)} \\ 1*, F_0^{(1)} \\ 1*, F_1^{(1)} \\ 1*, F_2^{(1)} \\ 1*, F_3^{(1)}
\end{bmatrix},
\begin{bmatrix}
* & & * & & * & & * & \\
& * & & * & & * & & * \\
& & & & * & & * & \\
& & & & & * & & * \\
* & & * & & * & & * & \\
& * & & * & & * & & * \\
* & & * & & & & & \\
& * & & * & & & &
\end{bmatrix}
\begin{bmatrix}
0*, F_0^{(1)} \\ 0*, F_1^{(1)} \\ 0*, F_2^{(1)} \\ 0*, F_3^{(1)} \\ 1*, F_0^{(1)} \\ 1*, F_1^{(1)} \\ 1*, F_2^{(1)} \\ 1*, F_3^{(1)}
\end{bmatrix}
=
\begin{bmatrix}
00*, F_0^{(2)} \\ 00*, F_1^{(2)} \\ 10*, F_0^{(2)} \\ 10*, F_1^{(2)} \\ 01*, F_0^{(2)} \\ 01*, F_1^{(2)} \\ 11*, F_0^{(2)} \\ 11*, F_1^{(2)}
\end{bmatrix}
$$

$$
\begin{bmatrix}
* & * & * & * & & & & \\
& & * & * & & & & \\
* & * & * & * & & & & \\
* & * & & & & & & \\
& & & & * & * & * & * \\
& & & & & & * & * \\
& & & & * & * & * & * \\
& & & & * & * & &
\end{bmatrix}
\begin{bmatrix}
00*, F_0^{(2)} \\ 00*, F_1^{(2)} \\ 10*, F_0^{(2)} \\ 10*, F_1^{(2)} \\ 01*, F_0^{(2)} \\ 01*, F_1^{(2)} \\ 11*, F_0^{(2)} \\ 11*, F_1^{(2)}
\end{bmatrix}
=
\begin{bmatrix}
000*, F_0^{(3)} \\ 100*, F_0^{(3)} \\ 010*, F_0^{(3)} \\ 110*, F_0^{(3)} \\ 001*, F_0^{(3)} \\ 101*, F_0^{(3)} \\ 011*, F_0^{(3)} \\ 111*, F_0^{(3)}
\end{bmatrix}
$$

Fig. 3. An illustration of $\mathtt{Red}_{\mathrm{BR}}^{-1}$ in Bruun style for $D = 8$ and $p \equiv 3 \bmod 4$. A '#' in matrices stands for a nonzero entry with the form of $\begin{bmatrix} a_0\mathbf{I}_{d/2} & a_1\mathbf{I}_{d/2} \\ a_2\mathbf{I}_{d/2} & a_3\mathbf{I}_{d/2} \end{bmatrix}$ for $a_i \in \mathbb{Z}_p$. Other symbols have the same meaning as in Figure 2.

Lemma 4. (1) *In the Bruun style decomposition, when viewed as $D \times D$ matrices, $\mathbf{N}_1$ and its inverse have only three non-zero diagonals indexed by $D/2 \times \{-1, 0, 1\}$. Each entry in $\mathbf{N}_1$ and $\mathbf{N}_1^{-1}$ has the form of $\begin{bmatrix} a_0\mathbf{I}_{d/2} & a_1\mathbf{I}_{d/2} \\ a_2\mathbf{I}_{d/2} & a_3\mathbf{I}_{d/2} \end{bmatrix}$ for $a_i \in \mathbb{Z}_p$ that may vary for each entry.*

(2) *For $j \in [2, \log_2(D)]$, $\mathbf{N}_j$ can be viewed as a $2^{j-2} \times 2^{j-2}$ diagonal block matrix. Each block has a size of $2^{2-j}D \times 2^{2-j}D$, which has 7 non-zero diagonals indexed by $2^{-j}D \times [-3, 3]$. Each entry in $\mathbf{N}_j$ is a multiple of $\mathbf{I}_d$. These properties also hold for $\mathbf{N}_j^{-1}$, except that the nonzero diagonals of $\mathbf{N}_j^{-1}$ are indexed by $2^{-j}D \times [-3, 2]$.*

Reducing the Number of Nonzero Diagonals. As an optimization, we can reduce the number of nonzero diagonals in the Bruun style decomposition from 6∼7 to only three by folding some nonzero diagonals inside each entry of $\mathbf{N}_j$.

To achieve this effect, we need to modify the output of the i-th Bruun butterfly in the first layer by storing a_{00} and a_{10} in $\boldsymbol{\alpha}_1[i]$ with a_{01} and a_{11} in $\boldsymbol{\alpha}_1[i+D/2]$, i.e., in a bit-reversed order.

Suppose $m \in R_{p^r}$, we first define vectors $\boldsymbol{\alpha}_j \in (\mathbb{Z}_{p^r}^d)^L$ and $\boldsymbol{\alpha}'_j \in (\mathbb{Z}_{p^r}^{d/2})^{2L}$ for $0 \le j \le \log_2(D)$ as follows. The vector $\boldsymbol{\alpha}_0$ corresponds to $\alpha = \mathtt{Red}(m) \in \mathcal{E}^L$. $\boldsymbol{\alpha}'_0$ is defined by $\boldsymbol{\alpha}'_0[2i] = \boldsymbol{\alpha}_0[i][0 +\colon d/2]$ and $\boldsymbol{\alpha}'_0[2i+1] = \boldsymbol{\alpha}_0[i][d/2 +\colon d/2]$ for $i \in [D]$. For $1 \le j \le \log_2(D)$, we define $\boldsymbol{\alpha}'_j$ such that

$$\boldsymbol{\alpha}'_j[2(i + k \cdot 2^{-j}D) + k_0] = (m \bmod F_i^{(j)})[\mathrm{BitRev}_{j+1,0}(2k + k_0)d/2 +\colon d/2]$$

for $i \in [2^{-j}D]$, $k \in [2^j]$ and $k_0 \in [2]$. Moreover, $\boldsymbol{\alpha}_j$ is defined by $\boldsymbol{\alpha}_j[i][0 +: d/2] = \boldsymbol{\alpha}'_j[2i]$ and $\boldsymbol{\alpha}_j[i][d/2 +: d/2] = \boldsymbol{\alpha}'_j[2i+1]$ for $i \in [D]$.

For $1 \leq j \leq \log_2(D)$, we define $\mathtt{N}'_j$ as the linear transformation that maps α_{j-1} to α_j. Denote $\mathtt{Red}'^{-1}_{\mathrm{BR}} = \mathtt{N}'_{\log_2(D)} \circ \cdots \circ \mathtt{N}'_1$, then

$$\mathtt{Red}'^{-1}_{\mathrm{BR}} = \mathtt{BR}_{\log_2(dD), \log_2(d)-1} \circ \mathtt{Red}^{-1}.$$

We call this kind of $\mathtt{Red}'_{\mathrm{BR}}$ as a *Radix-2 style* one. An example for $D = 8$ is shown in Figure 4. The formal statements about and the structure of $\mathtt{N}'_j$ are given in Lemma 5 and its proof is provided in Supplementary Material A.

$$\begin{bmatrix} \# &&&& \# &&& \\ & \# &&&& \# && \\ && \# &&&& \# & \\ &&& \# &&&& \# \\ \# &&&& \# &&& \\ & \# &&&& \# && \\ && \# &&&& \# & \\ &&& \# &&&& \# \end{bmatrix} \begin{bmatrix} X*, F_0^{(0)} \\ X*, F_1^{(0)} \\ X*, F_2^{(0)} \\ X*, F_3^{(0)} \\ X*, F_4^{(0)} \\ X*, F_5^{(0)} \\ X*, F_6^{(0)} \\ X*, F_7^{(0)} \end{bmatrix} = \begin{bmatrix} X0*, F_0^{(1)} \\ X0*, F_1^{(1)} \\ X0*, F_2^{(1)} \\ X0*, F_3^{(1)} \\ X1*, F_0^{(1)} \\ X1*, F_1^{(1)} \\ X1*, F_2^{(1)} \\ X1*, F_3^{(1)} \end{bmatrix}, \begin{bmatrix} \# && \# &&&&& \\ & \# && \# &&&& \\ \# && \# &&&&& \\ & \# && \# &&&& \\ &&&& \# && \# & \\ &&&&& \# && \# \\ &&&& \# && \# & \\ &&&&& \# && \# \end{bmatrix} \begin{bmatrix} X0*, F_0^{(1)} \\ X0*, F_1^{(1)} \\ X0*, F_2^{(1)} \\ X0*, F_3^{(1)} \\ X1*, F_0^{(1)} \\ X1*, F_1^{(1)} \\ X1*, F_2^{(1)} \\ X1*, F_3^{(1)} \end{bmatrix} = \begin{bmatrix} X00*, F_0^{(2)} \\ X00*, F_1^{(2)} \\ X10*, F_0^{(2)} \\ X10*, F_1^{(2)} \\ X01*, F_0^{(2)} \\ X01*, F_1^{(2)} \\ X11*, F_0^{(2)} \\ X11*, F_1^{(2)} \end{bmatrix}$$

$$\begin{bmatrix} \# & \# &&&&&& \\ \# & \# &&&&&& \\ && \# & \# &&&& \\ && \# & \# &&&& \\ &&&& \# & \# && \\ &&&& \# & \# && \\ &&&&&& \# & \# \\ &&&&&& \# & \# \end{bmatrix} \begin{bmatrix} X00*, F_0^{(2)} \\ X00*, F_1^{(2)} \\ X10*, F_0^{(2)} \\ X10*, F_1^{(2)} \\ X01*, F_0^{(2)} \\ X01*, F_1^{(2)} \\ X11*, F_0^{(2)} \\ X11*, F_1^{(2)} \end{bmatrix} = \begin{bmatrix} X000*, F_0^{(3)} \\ X100*, F_0^{(3)} \\ X010*, F_0^{(3)} \\ X110*, F_0^{(3)} \\ X001*, F_0^{(3)} \\ X101*, F_0^{(3)} \\ X011*, F_0^{(3)} \\ X111*, F_0^{(3)} \end{bmatrix}$$

Fig. 4. An illustration of $\mathtt{Red}'^{-1}_{\mathrm{BR}}$ in Radix-2 style for $D = 8$ and $p \equiv 1 \bmod 4$. A '$*$' in vectors means $\log_2(d) - 1$ bits ranging from all zeros to all ones while a 'X' means a single bit running from 0 to 1. For example, when $d = 8$, '$X0*$' stands for $(0000, 0001, 0010, 0011, 1000, 1001, 1010, 1011)$. Other symbols have the same meaning as in Figure 2 and Figure 3.

Lemma 5. *In the Radix-2 style* $\mathtt{Red}'^{-1}_{\mathrm{BR}}$*, for* $j \in [1, \log_2(D)]$, $\mathbf{N}'_j$ *can be viewed as a* $2^{j-1} \times 2^{j-1}$ *diagonal block matrix. Each block has a size of* $2^{-j+1}D \times 2^{-j+1}D$*, which has three non-zero diagonals indexed by* $2^{-j}D \times \{-1, 0, 1\}$*. Each entry in* $\mathbf{N}'_j$ *has the form of* $\begin{bmatrix} a_0 \mathbf{I}_{d/2} & a_1 \mathbf{I}_{d/2} \\ a_2 \mathbf{I}_{d/2} & a_3 \mathbf{I}_{d/2} \end{bmatrix}$ *for* $a_i \in \mathbb{Z}_p$ *that may vary for each entry. These properties also hold for* $\mathbf{N}'^{-1}_j$.

Proof of (2) in Theorem 1. Clearly, all $\mathtt{N}_j, \mathtt{N}'_j$ and their inverses are linear transformations on the major dimension because it is the only dimension. The indices of the nonzero diagonals stated in Theorem 1 can be directly derived from Lemma 4 and Lemma 5.

According to Lemma 4, the entries of $\mathbf{N}_j$ and $\mathbf{N}_j^{-1}$ are multiples of $\mathbf{I}_d$ if $j \in [2, \log_2(D)]$. Consequently, these linear transformations are in $\mathsf{MatMul1D}$ type. The entries of $\mathbf{N}_1$ and $\mathbf{N}_1^{-1}$ have the form

$$\begin{bmatrix} a_0\mathbf{I}_{d/2} & a_1\mathbf{I}_{d/2} \\ a_2\mathbf{I}_{d/2} & a_3\mathbf{I}_{d/2} \end{bmatrix}$$

for $a_i \in \mathbb{Z}_p$. These entries generally cannot be represented as a $\mathcal{E}$-linear map. Therefore, these matrices should be implemented as $\mathsf{BlockMatMul1D}$ type transformations.

On the other hand, according to Lemma 5, the entries of $\mathbf{N}'_j$ and $\mathbf{N}'^{-1}_j$ have the same form as $\mathbf{N}_1$ in the Bruun style decomposition. Thus, they should be implemented as $\mathsf{BlockMatMul1D}$ as well.

3.3 The Galois Ring Case

In this subsection, we give the proof of Theorem 1 for the case $r > 1$. Again, the derivation is different for the two cases of p.

The Case of $p \equiv 1 \bmod 4$. To begin with, we provide the factorization of $\Phi_M(X)$ over $\mathbb{Z}_{p^r}$ using Hensel's lifting.

Lemma 6. *For $p \equiv 1 \bmod 4$, it has $\Phi_M(X) = \prod_{i \in \mathbb{Z}_{4D}^*}(X^d - \zeta^i)$, where $\zeta \in \mathbb{Z}_{p^r}$ is a $4D$-th primitive root of unity.*

Proof. Let $\Phi_M(X) = \prod_{i \in \mathbb{Z}_{4D}^*}(X^d - \zeta_0^i)$ be the factorization into irreduible polynomials over $\mathbb{Z}_p$, where $\zeta_0 \in \mathbb{Z}_p$ is a primitive $4D$-th root of unity. By substituting $Y = X^d$, we obtain $\Phi_{M/d}(Y) = \prod_{i \in \mathbb{Z}_{4D}^*}(Y - \zeta_0^i)$. This factorization can be lifted to $\mathbb{Z}_{p^r}$ using Hensel's lemma, giving

$$\Phi_{M/d}(Y) = \prod_{i \in \mathbb{Z}_{4D}^*}(Y - u_i) \text{ for some distinct } u_i \in \mathsf{GR}(p^r).$$

Note that $u_i^{4D} - 1 = \Phi_{M/d}(u_i) = 0$. Furthermore, the u_i's are primitive $4D$-th root of unity due to $u_i \equiv \zeta_0^i \bmod p$ and $\zeta_0^i \in \mathbb{Z}_p$ is a primitive $4D$-th root of unity. Since $\mathbb{Z}_{p^r}^*$ is a cyclic group, we can assume that $u_i = \zeta^i$ for $i \in \mathbb{Z}_{4D}^*$, where $\zeta \in \mathbb{Z}_{p^r}$ is a $4D$-th primitive root of unity. The lemma then follows directly by replacing Y with X^d. $\square$

Note that the hypercube structure for the Galois ring case is identical to that of $r = 1$. Based on the factorization presented in Lemma 6, we can define $F_i^{(j)}$ and prove properties that are analogous to those stated in Lemma 1. Then by defining the linear transformation $\mathtt{N}_j$ in the same manner as in Section 3.1, we can prove statement (1) of Theorem 1 using the method outlined in Lemma 2.

The Case of $p \equiv 3 \bmod 4$. Again, we first provide the factorization of $\Phi_M(X)$ over $\mathbb{Z}_{p^r}$ using Hensel's lifting.

Lemma 7. *For $p \equiv 3 \bmod 4$, it has $\Phi_M(X) = \prod_{i \in \mathbb{Z}_{4D}^*/\langle p \rangle}(X^d - (\zeta^i + \zeta^{ip})X^{d/2} + \zeta^{i(p+1)})$, where $\zeta \in \mathsf{GR}(p^2;2)$ is a $4D$-th primitive root of unity and each factor is a polynomial in $\mathbb{Z}_{p^r}[X]$.*

Proof. Let $\Phi_M(X) = \prod_{i \in \mathbb{Z}_{4D}^*}(X^{d/2} - \zeta_0^i)$ be the factorization into irreduible polynomials over $\mathsf{GF}(p^2)$, where $\zeta_0 \in \mathsf{GF}(p^2)$ is a primitive $4D$-th root of unity. By substituting $Y = X^{d/2}$, we get $\Phi_{2M/d}(Y) = \prod_{i \in \mathbb{Z}_{4D}^*}(Y - \zeta_0^i)$ over $\mathsf{GF}(p^2)$. This factorization can be lifted from $\mathsf{GF}(p^2)$ to $\mathsf{GR}(p^r;2)$ using Hensel's lemma, i.e.,

$$\Phi_{2M/d}(Y) = \prod_{i \in \mathbb{Z}_{4D}^*}(Y - u_i), u_i \in \mathsf{GR}(p^r;2).$$

Similarly, the u_i's form the complete set of $4D$-th primitive roots of unity in $\mathsf{GR}(p^r;2)$, and we can assume that $u_i = \zeta^i$ for a primitive $4D$-th root of unity $\zeta \in \mathsf{GF}(p^2)$. It only remains to prove that $(Y^i - \zeta^i)(Y^i - \zeta^{ip}) \in \mathbb{Z}_{p^r}[X]$, which is equivalent to proving both $-(\zeta^i + \zeta^{ip})$ and $\zeta^{i(p+1)}$ are in $\mathbb{Z}_{p^r}$.

Let γ be a primitive element such that $\mathsf{GR}(p^r;2) = \mathbb{Z}_{p^r}[\gamma]$. According to Section 2.2, the unit group $\mathsf{GR}(p^r;2)^*$ is isomorphic to $C_{p^2-1} \times C_{p^{r-1}} \times C_{p^{r-1}}$, where C_i denotes a cyclic group of order i. Given that $\mathrm{ord}_{\mathsf{GR}(p^r;2)^*}(\gamma) = p^2 - 1$ and $\mathrm{ord}_{\mathsf{GR}(p^r;2)^*}(\zeta) = 4D$ are both coprime to p, it follows that ζ is a power of γ. Furthermore, as $4D$ divides $p^2 - 1$, we can deduce that $\zeta = \gamma^k$ for some integer k that is divisible by $(p^2-1)/4D$. Let π be the Frobenius automorphism, we have

$$\pi(\zeta^i + \zeta^{ip}) = \pi(\gamma^{ki} + \gamma^{kip}) = \gamma^{kip} + \gamma^{kip^2} = \gamma^{kip} + \gamma^{ki} = \zeta^i + \zeta^{ip},$$
$$\pi(\zeta^{i(p+1)}) = \pi(\gamma^{ki(p+1)}) = \gamma^{ki(p^2+p)} = \gamma^{ki(p+1)} = \zeta^{i(p+1)}.$$

Thus, $(\zeta^i + \zeta^{ip})$ and $\zeta^{i(p+1)}$ are in $\mathbb{Z}_{p^r}$, and the lemma follows directly. □

Drawing upon the factorization presented in Lemma 7, we are able to define $F_i^{(j)}$ and establish properties that are same to those stated in Lemma 3. Subsequently, we can construct the linear transformation $\mathtt{N}_j$ in a manner consistent with Section 3.2, and validate properties that are same to those in Lemma 4. In addition, it can be verified that the methodology presented in Lemma 5 is still applicable, thereby enabling us to prove statement (2) of Theorem 1.

4 Algorithmic Optimizations of Homomorphic NTT

In this section, we introduce multiple optimizations based on the decomposition in Theorem 1. In Section 4.1, we combine consecutive $\mathtt{N}_j$'s to realize a tradeoff between level consumption and running time. In Section 4.2, we modify the logic of the BSGS-style linear transformation to reduce the number of unhoisted automorphisms for better efficiency. In Section 4.3, we discuss the interaction of our decomposed CoeffToSlot/SlotToCoeff with general and thin bootstrapping. Finally, we analyze and compare the asymptotic complexity of the previous and our method in Section 4.4.

4.1 Combining Consecutive $\mathtt{N}_j$'s

Note that the evaluation of each MatMul1D or BlockMatMul1D consumes a multiply-by-constant depth. Thus evaluating all the $\mathtt{N}_i$'s one by one will consume a depth of $\log_2(L)$, which can significantly diminish the remaining depth after bootstrapping when L is large. This issue can be mitigated by combining several consecutive $\mathtt{N}_i$'s and evaluating the resulting composite linear transformations as a whole. We note that a similar technique, known as level-collapsing, has been proposed for FFT-based CKKS bootstrapping in [9,22].

The properties of the composite linear transformations can be stated as follows.

Lemma 8. *Let $k \in [1, \log_2(D)]$ and $1 \le j \le k$.*

If $p \equiv 1 \bmod 4$, then it has

$$\mathit{DiagSet}(\mathtt{N}_k \dots \mathtt{N}_j) = \mathit{DiagSet}(\mathtt{N}_j^{-1} \dots \mathtt{N}_k^{-1}) = 2^{-k}D \times [-2^{1+k-j}+1, 2^{1+k-j}-1]_{2^k}.$$

If $p \equiv 3 \bmod 4$, then it has

$$\mathit{DiagSet}(\mathtt{N}_k \dots \mathtt{N}_j) = 2^{-k}D \times [-3(2^{1+k-j}-1), 3(2^{1+k-j}-1)]_{2^k},$$

$$\mathit{DiagSet}(\mathtt{N}_j^{-1} \dots \mathtt{N}_k^{-1}) = 2^{-k}D \times [-3(2^{1+k-j}-1), 2(2^{1+k-j}-1)]_{2^k},$$

$$\mathit{DiagSet}(\mathtt{N}'_k \dots \mathtt{N}'_j) = \mathit{DiagSet}(\mathtt{N}'^{-1}_j \dots \mathtt{N}'^{-1}_k) = 2^{-k}D \times [-2^{1+k-j}-1, 2^{1+k-j}-1]_{2^k}.$$

Specifically, if $j = 1$, all the RHS become $2^{-k}D \times [2^k]$.

Proof. We prove the conclusions about $\mathrm{DiagSet}(\mathtt{N}_k \dots \mathtt{N}_j)$ by induction on k. When $k = j$, the conclusions are true due to Theorem 1. Suppose they hold for some k_0 with $j \le k = k_0 < \log_2(D)$, we prove they still hold for $k+1$. Since

$$\mathrm{DiagSet}(\mathtt{N}_{k+1} \dots \mathtt{N}_j) = \bigcup_{a \in \mathrm{DiagSet}(\mathtt{N}_{k+1})} [a + \mathrm{DiagSet}(\mathtt{N}_k \dots \mathtt{N}_j)]_D,$$

substituting $\mathrm{DiagSet}(\mathtt{N}_{k+1})$ and $\mathrm{DiagSet}(\mathtt{N}_k \dots \mathtt{N}_j)$ with the corresponding values in each case will lead to the desired results.

For the inverses of the transformations above, the conclusions can be obtained similarly. □

In the case of $p \equiv 1 \bmod 4$, the composition of multiple $\mathtt{N}_i$ may not be a one-dimensional linear transformation if $\mathtt{N}_{\log_2(D)+1}$ is included. Let ρ_1 be the rotation operation along the minor dimension. According to Theorem 1, $\mathtt{N}_{\log_2(D)+1}$ represents a MatMul1D in the minor dimension, which can be implemented as $\mathtt{N}_{\log_2(D)+1}(m) = \kappa_0(0)m + \kappa_0(1)\rho_1(m)$ for some $\kappa_0(0), \kappa_0(1) \in R_{p^r}$. Thus, for $\mathtt{N} = \mathtt{N}_k \circ \dots \circ \mathtt{N}_j$ with $1 \le k \le \log_2(D)$ as in Lemma 8, which is a one-dimensional linear transformation along the major dimension, the cross-dimensional transformation $\mathtt{N}_{\log_2(D)+1} \circ \mathtt{N}$ can be computed in the form of

$$\mathtt{N}_{\log_2(D)+1} \circ \mathtt{N}(m) = \sum_{i \in \mathrm{DiagSet}(\mathtt{N})} \kappa_1(i)\rho^i(m) + \rho_1 \left(\sum_{i \in \mathrm{DiagSet}(\mathtt{N})} \kappa_2(i)\rho^i(m) \right)$$

for some $\kappa_1(i)$ and $\kappa_2(i) \in R_{p^r}$. This is called a MatMulFull transformation [19].

4.2 Modified BSGS Style Linear Transformations

We note that a large number of slots L implies that the size D of the main dimension is large. Thus the rotation keys for the main dimension should be generated in a baby-step-giant-step (BSGS) way, which can reduce the number of rotation keys from D to about $2\sqrt{D}$. As stated in [19], the BSGS method chooses $g = \lceil \sqrt{D} \rceil$ as the 'giant step'. Denote $h = \lceil D/g \rceil$, it generates the rotation keys for Galois rotations θ^i, where either $i \in [g]$ (i.e., the baby steps) or $i \in g \cdot [h]$ (i.e., the giant steps). Then for a good dimension, it has $\rho = \theta$ and $\mathsf{MatMul1D}$ is implemented as

$$T_N(m) = \sum_{k \in [h]} \rho^{gk} \left(\sum_{j \in [g]} \kappa'(j+gk)\rho^j(m) \right), \text{ for } m \in R_{p^r}, \tag{4}$$

where $\kappa'(j+gk) = \rho^{-gk}(\kappa(j+gk))$. The $\rho^j(m)$'s are computed using the hoisting technique, while the ρ^{gk}'s cannot be computed with hoisting because they have different inputs. For a bad dimension, $\mathsf{MatMul1D}$ is implemented as

$$T_N(m) = \sum_{k \in [h]} \theta^{gk} \left(\sum_{j \in [g]} \kappa'(j+gk)\theta^j(m) + \kappa''(j+gk)\theta^{j-D}(m) \right) \tag{5}$$

for $m \in R_{p^r}$, where $\kappa'(j+gk) = \theta^{-gk}(\mu(j+gk)\kappa(j+gk))$ and $\kappa''(j+gk) = \theta^{-gk}(\mu'(j+gk)\kappa(j+gk))$. Again, $\theta^j(m)$ and $\theta^{j-D}(m)$ are computed with hoisting on m and $\theta^{-D}(m)$ while θ^{gk} are computed without hoisting.

Modified BSGS Method for $\mathsf{MatMul1D}$. For a $\mathsf{MatMul1D}$ map $\mathtt{N}$ along the major dimension, define $\mathrm{GiantSet}(\mathtt{N}) = \{\lfloor \frac{[i]_D}{g} \rfloor \mid i \in \mathrm{DiagSet}(\mathtt{N})\}$ and $\mathrm{BabySet}(\mathtt{N}) = \{[i]_D \bmod g \mid i \in \mathrm{DiagSet}(\mathtt{N})\}$. Then, we can replace '$[h]$' with '$\mathrm{GiantSet}(\mathtt{N})$' and '$[g]$' with '$\mathrm{BabySet}(\mathtt{N})$' in Equation 4 and Equation 5.

Our key observation is that the matrices that $\mathtt{Red}_{\mathrm{BR}}^{-1}$ splits into usually have either a small GiantSet or a small BabySet. For example, consider the case of $p \equiv 1 \bmod 4$ and $D = 2^{2k}$ for some integer k. Using Theorem 1 and Lemma 8, consider two composite linear transformations $\mathtt{N}^{(1)} = \mathtt{N}_k \ldots \mathtt{N}_1$ and $\mathtt{N}^{(2)} = \mathtt{N}_{2k} \ldots \mathtt{N}_{k+1}$. We have $\mathrm{DiagSet}(\mathtt{N}^{(2)}) = [-2^k+1, 2^k-1]$ and $\mathrm{DiagSet}(\mathtt{N}^{(1)}) = 2^k \times [2^k]$. Since $g = h = 2^k$, we have $\mathrm{GiantSet}(\mathtt{N}^{(2)}) = \{-1, 0, 1\}$, $\mathrm{BabySet}(\mathtt{N}^{(2)}) = [2^k]$ and $\mathrm{GiantSet}(\mathtt{N}^{(1)}) = [2^k]$, $\mathrm{BabySet}(\mathtt{N}^{(1)}) = \{0\}$. If $|\mathrm{GiantSet}(\mathtt{N})|$ is small for a linear transformation $\mathtt{N}$, the number of unhoisted automorphisms (i.e., ρ^{gk} and θ^{gk}) in Equation 4 and Equation 5 is greatly reduced.

In the other case where $\mathrm{BabySet}(\mathtt{N})$ is small, we exchange the role of j, k to obtain the revised $\mathsf{MatMul1D}$ in a good dimension

$$\mathtt{N}(m) = \sum_{j \in \mathrm{BabySet}(\mathtt{N})} \rho^j \left(\sum_{k \in \mathrm{GiantSet}(\mathtt{N})} \kappa'(j+gk)\rho^{gk}(m) \right), \tag{6}$$

where $\kappa'(j+gk) = \rho^{-j}\kappa(j+gk)$, and the revised $\mathsf{MatMul1D}$ in a bad dimension

$$\mathtt{N}(m) = \sum_{j \in \mathrm{BabySet}(\mathtt{N})} \theta^j \left(\sum_{k \in \mathrm{GiantSet}(\mathtt{N})} \kappa'(j+gk)\theta^{gk}(m) + \kappa''(j+gk)\theta^{gk-D}(m) \right),$$

where $\kappa'(j+gk) = \theta^{-j}(\mu(j+gk)\kappa(j+gk))$ and $\kappa''(j+gk) = \theta^{-j}(\mu'(j+gk)\kappa(j+gk))$.

Swapping the roles of j and k whenever $|\mathrm{GiantSet}(\mathtt{N})| > |\mathrm{BabySet}(\mathtt{N})|$ ensures that the number of unhoisted automorphisms is minimized while the total number of automorphisms is fixed. This reduces the running time since hoisted automorphisms are cheaper than unhoisted ones.

In our example above, the sparsity of $\mathrm{BabySet}(\mathtt{N}^{(1)})$ relies on the fact that $g = \sqrt{D}$ is a power of 2. However, this is not true if $D = 2^{2k+1}$ for some integer k. Thus, in this case, we choose $g = 2^{k+1}$ and $h = 2^k$ so that the previous optimizations are still valid. Compared to the original choice of g, such choice of g will slightly increase the number of rotation keys from $2^{1.5} \cdot 2^k$ to $3 \cdot 2^k$ by about 6%, which is an acceptable cost.

Modified BSGS Method for BlockMatMul1D. The tricks for MatMul1D can be applied to the computation of BlockMatMul1D in either good or bad dimensions.

When HElib computes a BlockMatMul1D transformation, $\rho^i(m)$'s in Equation 2 are computed for all $i \in [D]$ if the dimension is good. In a bad dimension, $\theta^i(m)$'s are computed for all $i \in [D]$. Let $j = [i]_g$ and $k = \lfloor \frac{i}{g} \rfloor$, these ciphertexts are generated in two steps, (1) $\theta^{gk}(m)$ are generated from m with hoisting for $k \in [h]$, (2) $\theta^i(m) = \theta^j(\theta^{gk}(m))$ are generated from $\theta^{gk}(m)$ with hoisting for $j \in [g]$. Thus, we can still replace $[g]$ with $\mathrm{BabySet}(\mathtt{N})$ and $[h]$ with $\mathrm{GiantSet}(\mathtt{N})$ for faster computation. The role of giant and baby steps can also be swapped if $|\mathrm{BabySet}(\mathtt{N})| < |\mathrm{GiantSet}(\mathtt{N})|$, which reduces the number of hoisting precomputations from $|\mathrm{GiantSet}(\mathtt{N})| + 1$ to $|\mathrm{BabySet}(\mathtt{N})| + 1$. If they are swapped, $\theta^j(m)$ will be generated from m and $\theta^{j+gk}(m)$ will be computed from $\theta^j(m)$.

4.3 Applying the Decomposition to BGV Bootstrapping

In this subsection, we describe how the decomposition of linear transformations can be deployed into general or thin bootstrapping, including some modifications to them for better efficiency.

Recall that $\mathtt{Decode} = \mathtt{Eval} \circ \mathtt{Red}$ and $\mathtt{Red}^{-1} = \mathtt{BR} \circ \mathtt{Red}_{\mathrm{BR}}^{-1}$, where $\mathtt{BR}$ is an order-two permutation of the $L \cdot d$ slot coefficients induced by some bit-reversal map. Then the polynomial $m \in R_{p^r}$ and its slot values α are related as

$$\alpha = \mathtt{Decode}(m) = \mathtt{Eval} \circ \mathtt{Red}(m) = \mathtt{Eval} \circ \mathtt{Red}_{\mathrm{BR}} \circ \mathtt{BR}^{-1}(m).$$

Applying to General Bootstrapping. The workflow of general bootstrapping is illustrated in Figure 5. Note that the output of CoeffToSlot and the

$$
\begin{tikzcd}
 & \text{Decryption Formula Simplification} & \text{CoeffToSlot} & \text{Digit Removal} & & \text{SlotToCoeff} \\
\text{Coefficients} & m_0 \arrow[r] & m \arrow[dr] & & & m_0 \\
\text{Slots} & \alpha_0 & \alpha & \mathtt{BR}(m) \arrow[r] & \mathtt{BR}(m_0) \arrow[ur] & \alpha_0 \\
 & & & \mathtt{Red}_{\mathrm{BR}}^{-1} \circ \mathtt{Eval}^{-1}(\alpha) \arrow[u, equal] & & \mathtt{Eval} \circ \mathtt{Red}_{\mathrm{BR}}(\mathtt{BR}(m_0)) \arrow[u, equal]
\end{tikzcd}
$$

Fig. 5. Workflow of general BGV bootstrapping. The slot values in $\mathtt{BR}(m)$ after CoeffToSlot are identified with $\mathbb{Z}_{p^r}^d$ with respect to the normal basis of $\mathcal{E}$. Other slot values are represented with respect to the power basis of $\mathcal{E}$.

input of SlotToCoeff is a permutated version of m or m_0. This helps to avoid computing $\mathtt{BR}$ and its inverse homomorphically, which will be rather expensive.

The CoeffToSlot transformation (corresponding to the $\mathtt{Red}_{\mathrm{BR}}^{-1} \circ \mathtt{Eval}^{-1}$) is followed by a $\mathsf{BlockMatMul1D}$ transformation that moves the power basis coefficients of each slot into the normal basis [21]. Denoting this transformation as $\mathtt{PtoN}$, the overall transformation applied is $\mathtt{PtoN} \circ \mathtt{Red}_{\mathrm{BR}}^{-1} \circ \mathtt{Eval}^{-1}$, where $\mathtt{PtoN}$ and $\mathtt{Eval}^{-1}$ are slot-wise $\mathsf{BlockMatMul1D}$. Denote the split $\mathtt{Red}_{\mathrm{BR}}^{-1}$ as $\mathtt{Red}_{\mathrm{BR}}^{-1} = \mathtt{N}^{(k)} \dots \mathtt{N}^{(1)}$. As the first optimization, we merge $\mathtt{Eval}^{-1}$ with $\mathtt{N}^{(1)}$ to save a multiply-by-constant level, which is a tradeoff between level and time. Moreover, this is free if $\mathtt{N}^{(1)}$ is already a $\mathsf{BlockMatMul1D}$. This trick is applied to both SlotToCoeff and CoeffToSlot transformations, whether the bootstrapping is a general one or a thin one.

As the second optimization, we merge $\mathtt{PtoN}$ with the $\mathtt{N}^{(k)}$ to save a multiply-by-constant level, again increasing its running time if it is not a $\mathsf{BlockMatMul1D}$. However, we can avoid the extra cost by reordering $\mathtt{N}^{(k)}$. If $p \equiv 1 \bmod 4$, all $\mathtt{N}^{(i)}$'s are either $\mathsf{MatMul1D}$ or $\mathsf{MatMulFull}$. For $p \equiv 3 \bmod 4$, $\mathtt{N}^{(1)}$ is a $\mathsf{BlockMatMul1D}$ and other $\mathtt{N}^{(i)}$'s are either $\mathsf{MatMul1D}$ (for Bruun style decomposition) or $\mathsf{BlockMatMul1D}$ (for Radix-2 style decomposition). Each entry of a $\mathsf{MatMul1D}$ or $\mathsf{MatMulFull}$ used here is a multiple of $\mathbf{I}_d$, which is a linear transformation that multiplies the input $v \in \mathcal{E}$ by some constant in $\mathbb{Z}_{p^r}$. Note that such a multiply-by-integer map remains the same regardless of the basis we use for $\mathcal{E}$ (i.e., the power basis or the normal basis). Thus, $\mathtt{PtoN}$ commutes with all $\mathtt{N}^{(i)}$'s that are $\mathsf{MatMul1D}$ or $\mathsf{MatMulFull}$. It is easy to see that there exists some integer j such that $\mathtt{N}^{(i)}$ is a $\mathsf{BlockMatMul1D} \iff i \leq j$. Then we can rewrite the overall linear transformation as

$$\mathtt{N}^{(k)} \circ \dots \circ \mathtt{N}^{(j+1)} \circ (\mathtt{PtoN} \circ \mathtt{N}^{(j)}) \circ \mathtt{N}^{(j-1)} \circ \dots \circ \mathtt{N}^{(2)} \circ (\mathtt{N}^{(1)} \circ \mathtt{Eval}^{-1}).$$

In this way, we ensure that the number of $\mathsf{BlockMatMul1D}$ transformations during SlotToCoeff is minimized to $\max(j, 1)$. Since $\mathsf{BlockMatMul1D}$ is usually more time-consuming than $\mathsf{MatMul1D}$, running time is saved by the reordering of transformations.

$$
\begin{tikzcd}
 & & \text{SlotToCoeff} & \text{Decryption Formula Simplification} & \text{CoeffToSlot} & \text{Digit Removal} \\
\text{Coefficients} & m_0 & & \mathtt{BR}(\alpha_0) \arrow[r] & \mathtt{BR}(\alpha^*) \arrow[dr] & & \\
\text{Slots} & \alpha_0 \arrow[urr] & & \beta_0 & \beta & \alpha \arrow[r] & \alpha_0
\end{tikzcd}
$$

	β_0	α
$p \equiv 1 \bmod 4$	$= \mathtt{Red}_{\mathrm{BR}}(\alpha_0)$	$= \mathtt{Red}_{\mathrm{BR}}^{-1} \circ \mathtt{Rm}(\beta)$
$p \equiv 3 \bmod 4$, Bruun style	$\mathtt{Eval} \circ \mathtt{Red}_{\mathrm{BR}}(\alpha_0)$	$\mathtt{Red}_{\mathrm{BR}}^{-1} \circ \mathtt{Eval}^{-1} \circ \mathtt{Rm}(\beta)$
$p \equiv 3 \bmod 4$, Radix-2 style	$\mathtt{Eval} \circ \mathtt{Red}_{\mathrm{BR}}(\alpha_0)$	$\mathtt{Rm}' \circ \mathtt{Red}_{\mathrm{BR}}^{-1} \circ \mathtt{Eval}^{-1} \circ \mathtt{Rm}(\beta)$

Fig. 6. Workflow of thin BGV bootstrapping. The SlotToCoeff and CoeffToSlot transformations are compositions of different sub-transformations for different parameters. All slot values are represented with respect to the power basis of $\mathcal{E}$.

Applying to Thin Bootsgrapping. The workflow of thin bootstrapping is illustrated in Figure 6. The permutation BR is also not computed homomorphically, similar to that in general bootstrapping.

SlotToCoeff (corresponding to $\mathtt{Eval} \circ \mathtt{Red}_{\mathrm{BR}}$) is performed first on a thin ciphertext, where each slot contains an integer instead of a Galois ring element. Let the slot values of the input to SlotToCoeff be $\alpha \in \mathcal{E}^L$. If $p \equiv 1 \bmod 4$, each slot in $\mathtt{Red}_{\mathrm{BR}}(\alpha_0)$ still stores an integer because the entry in $\mathtt{Red}_{\mathrm{BR}}$ is a multiple of $\mathbf{I}_d$. This means the restriction of $\mathtt{Eval}$ on $\mathtt{Red}_{\mathrm{BR}}(\alpha_0)$ is an identity map and can be omitted. For $p \equiv 3 \bmod 4$, the value in each slot during the computation of $\mathtt{Red}_{\mathrm{BR}}$ lies in the subring $F \subset \mathcal{E}$ satisfying $[F : \mathsf{GR}(p^r)] = 2$ because each entry of $\mathbf{N}_j$ has the form of $\begin{bmatrix} a_0\mathbf{I}_{d/2} & a_1\mathbf{I}_{d/2} \\ a_2\mathbf{I}_{d/2} & a_3\mathbf{I}_{d/2} \end{bmatrix}$ for $a_i \in \mathbb{Z}_p$. This means the linearized polynomials in the $\mathsf{BlockMatMul1D}$ maps of $\mathtt{Red}_{\mathrm{BR}}$ and in $\mathtt{Eval}$ can be built on F instead of on $\mathcal{E}$, reducing the highest power of σ in the linearized polynomials from $d-1$ to 1.

Another feature of thin bootstrapping is that a trace-like map needs to be applied to the ciphertext to clear the extra coefficients introduced by the decryption formula simplification. For a power-of-2 M, Chen and Han found that such a map can be computed efficiently before CoeffToSlot [10]. As their core observation, for $m \in R_{p^r}$ and $0 \le k \le \log_2(M/2)$, it is possible to obtain $m' \in R_{p^r}$ such that $m'[i] = 0$ for $[i]_{2^k} \neq 0$ and $m'[i] = m[i]$ otherwise. In other words, $\mathtt{RM}_k$ keeps $m[i]$ if and only if the lowest k bits in the binary representation of i are zero. Denote this map as $\mathtt{RM}_k : R_{p^r} \to R_{p^r}$, its computation proceeds as follows, where $\Gamma_j(m(X)) = m(X^j)$.

In Figure 6, $\mathtt{Rm}$ and $\mathtt{Rm}'$ clear the extra coefficients in $\mathtt{BR}(\alpha^*)$ introduced by decryption formula simplification into $\mathtt{BR}(\alpha_0)$. Using our FFT-like linear transformations, the permutation $\mathtt{BR}$ satisfies

$$
\mathsf{BR} = \begin{cases} \mathsf{BR}'_{\log_2(D)+1,\log_2(d)}, & \text{if } p \equiv 1 \bmod 4 \\ \mathsf{BR}_{\log_2(D),\log_2(d)}, & \text{if } p \equiv 3 \bmod 4, \text{ Bruun style decomposition} \\ \mathsf{BR}_{\log_2(D),\log_2(d)-1}, & \text{if } p \equiv 3 \bmod 4, \text{ Radix-2 style decomposition} \end{cases} .
$$

Algorithm 1 $\mathtt{RM}_k$ map [10]

Input: m
Output: $m' = \mathrm{RM}_k(m)$
$m_0 \leftarrow m$
for $i = 1, \ldots, k$ **do**
  $m_i \leftarrow m_{i-1} + \Gamma_{2^{-i}M+1}(m_{i-1})$
end for
$m' \leftarrow 2^{-k} m_k$

For $p \equiv 1 \bmod 4$, the indices of the coefficients of $\mathtt{BR}(\alpha_0)$ in Figure 6 form the set $d \times [2D]$. I.e., $\mathtt{BR}(\alpha^*)[i]$ should be kept by $\mathtt{RM}$ if and only if the lowest $\log_2(d)$ bits of i are all zeros. Thus, we let $\mathtt{RM} = \mathtt{RM}_{\log_2(d)}$ and $\mathtt{RM}'$ be the identity map. Note that we abuse the notation of $\mathtt{RM}_k : R_{p^r} \to R_{p^r}$ here to denote its corresponding map on the slots, which is a $\mathcal{E}^L \to \mathcal{E}^L$ map.

For $p \equiv 3 \bmod 4$ and Bruun style decomposition, the indices of coefficients of $\mathtt{BR}(\alpha_0)$ form $d \times [D]$ and $\mathtt{RM} = \mathtt{RM}_{\log_2(d)}$ suffices to clear the extra coefficients. However, for Radix-2 style decomposition, the indices of the coefficients of $\mathtt{BR}(\alpha_0)$ form the set $\{\mathtt{BR}_{\log_2(D), \log_2(d)-1}(i) \mid i \in d \times [D]\} = d/2 \times [D]$. In other words, $\mathtt{BR}(\alpha^*)[i]$ should be kept by $\mathtt{RM}$ if and only if the highest bit and the lowest $\log_2(d) - 1$ bits of i are all zeros. In this case, although we can clear $\mathtt{BR}(\alpha^*)[i]$ with $[i]_{d/2} \neq 0$ using $\mathtt{RM} = \mathtt{RM}_{\log_2(d)-1}$, those undesired coefficients with indices in $d/2 \times [D/2, D-1]$ cannot be cleared. This means that the first $D/2$ slots in $\mathtt{Red}_{\mathrm{BR}}^{-1} \circ \mathtt{Eval}^{-1} \circ \mathrm{RM}(\beta)$ will have the form of $\alpha_i + bX^{d/2}$, with b being the undesired coefficient. Thus, an extra map $\mathtt{Rm}'$ needs to be applied slot-wise to clear b in these slots. We note that $\mathtt{Rm}'$ can also be represented as a linearized polynomial in the subring $F \subset \mathcal{E}$ and can be incorporated into the last $\mathsf{BlockMatMul1D}$ in $\mathtt{Red}_{\mathrm{BR}}^{-1}$ for free.

The optimizations we made to SlotToCoeff can be applied to CoeffToSlot as well. Specifically, if $p \equiv 1 \bmod 4$, $\mathtt{Eval}^{-1}$ in CoeffToSlot can also be omitted because $\mathtt{Rm}(\beta)$ stores an integer in each of its slots. For $p \equiv 3 \bmod 4$, $\mathtt{Rm}(\beta)$ and the intermediate results during the computation of $\mathtt{Red}_{\mathrm{BR}}^{-1}$ store an element in the subring F in each of their slots. Again, this means the linearized polynomials of $\mathtt{Eval}^{-1}$ and the $\mathsf{BlockMatMul1D}$ maps that $\mathtt{Red}_{\mathrm{BR}}^{-1}$ splits into can be built on F instead of on $\mathcal{E}$.

Remark. When $p \equiv 3 \bmod 4$, let $\mathtt{Red}_{\mathrm{BR}} = \mathtt{N}^{(k)} \circ \cdots \circ \mathtt{N}^{(1)}$, where $\mathtt{N}^{(i)}$'s are composition of multiple $\mathtt{N}_j'^{-1}$'s. We remark that $\mathtt{N}^{(1)}$ can be simplified from a $\mathsf{BlockMatMul1D}$ into a $\mathsf{MatMul1D}$ because each slot in its input stores only an integer in $\mathbb{Z}_{p^r}$. This is not true for $\mathtt{N}^{(i)}$ with $i \geq 2$ or the inverse matrices in $\mathtt{Red}_{\mathrm{BR}}^{-1}$ because each slot in their inputs lies in the subring F. We do not include this optimization in our implementation for simplicity.

4.4 Asymptotic Complexity Analysis

In this subsection, we discuss the asymptotic complexity of linear transformations in BGV bootstrapping for both our method and the baseline approach. The results are summarized in Table 1. For our method, we ignore the optimization of combining `Eval`, $\mathtt{N}_j$, and `PtoN` due to the maximum number of decompositions is logarithmic in L, rendering the depth consumption negligible in the asymptotic analysis. For the baseline method, we assume that the rotation keys are generated in the BSGS manner, and CoeffToSlot/SlotToCoeff is evaluated without decomposition. The complexity of both methods is estimated by counting the number of unhoisted automorphisms and hoisting precomputation, which are the most computationally expensive operations.

Table 1. Asymptotic complexity of linear transformations in BGV bootstrapping for our method and the baseline method.

Complexity	Thin Bootstrapping	General Bootstrapping
Baseline	$O(\log_2(d) + \sqrt{L})$	$O(d + \sqrt{L})$
Ours	$O(\log_2(d) + \log_2(L))$	$O(d \cdot \log_2(L))$

For the baseline method, the whole CoeffToSlot/SlotToCoeff in thin bootstrapping is a MatMul1D [19,16], requiring a complexity of $O(\sqrt{L})$. For both methods, the complexity of `RM` and $\mathtt{RM}'$ is $O(\log_2(d))$. In general bootstrapping, CoeffToSlot and SlotToCoeff become BlockMatMul1D, thus having a complexity of $O(d + \sqrt{L})$ according to [19]. Thus, the total complexity is $O(\log_2(d) + \sqrt{L})$ for thin bootstrapping and $O(d + \sqrt{L})$ for general bootstrapping.

For our method, the complexity of `PtoN` is $O(d)$ for general bootstrapping, while the complexity of `Eval` and its inverse is $O(1)$ for thin bootstrapping and $O(d)$ for general bootstrapping. Each $\mathtt{N}_j$ in our method has a computational complexity of $O(1)$ in thin bootstrapping and $O(d)$ in general bootstrapping. Thus, the total complexity of evaluating all $\mathtt{N}_j$'s is $O(\log_2(L))$ in thin bootstrapping and $O(d \cdot \log_2(L))$ in general bootstrapping, leading to a total complexity of $O(\log_2(d) + \log_2(L))$ and $O(d + d \cdot \log_2(L)) = O(d \cdot \log_2(L))$. Additionally, if we generate all the Frobenius key-switching keys of σ^i for $i \in [d]$ and exchange the order of θ and σ (as mentioned in [19]), the complexity of each $\mathtt{N}_j$ in general bootstrapping can be lowered to $O(1)$, leading to a $O(d + \log_2(L))$ complexity for general bootstrapping using our method.

5 Implementation

5.1 Experiment Setup

We implemented our approach in BGV bootstrapping based on HElib (commit id `3e337a6`) with the optimization in [25]. The security level of BGV parameter sets is estimated using the lattice estimator [2] with commit id `fd4a460`. The experiments are conducted on a machine running Fedora 33 (Workstation Edition) equipped with a 3 GHz Intel(R) Core(TM) i9-10980XE CPU and 125GB of RAM. The compiled program is executed in a single thread, as in previous works on BGV bootstrapping [21,25].

Parameter selection. We set p to be of the form $2^i \pm 1$ for friendly integer arithmetic, and choose it to correspond to a large number of slots L, ranging from 4096 to 32768. The Hamming weight h of the main secret key is set to 120, aligning with the default value used in HElib. In accordance with previous works [21,15,25], we choose the maximum ciphertext modulus Q to guarantee a security level of at least 80 bits. The Hamming weight of the encapsulated bootstrapping key is chosen to have a security level of at least 128 bits to defend against potential attacks on sparse secrets, which is consistent with the choice in [25]. The selected parameter sets are displayed in Table 2.

Table 2. The parameter sets. h and λ are the Hamming weight and the security level of the main secret key, while h' and λ' are those for the encapsulated bootstrapping key.

<table>
<tr><th>ID</th><th>p</th><th>r</th><th>M</th><th>L</th><th>D</th><th>d</th><th>$\log_2(Q)$</th><th>h</th><th>λ</th><th>h'</th><th>λ'</th></tr>
<tr><td>I</td><td>65537</td><td rowspan="3">1</td><td>65536</td><td>32768</td><td>16384</td><td>1</td><td rowspan="3">1332</td><td rowspan="3">120</td><td rowspan="3">81.13</td><td>26</td><td>134.4</td></tr>
<tr><td>II</td><td>8191</td><td>65536</td><td>4096</td><td>4096</td><td>8</td><td>24</td><td>129.8</td></tr>
<tr><td>III</td><td>131071</td><td>65536</td><td>16384</td><td>16384</td><td>2</td><td>26</td><td>133.81</td></tr>
</table>

The Decomposition of $\mathtt{Red}_{\mathbf{BR}}^{\mathbf{-1}}$. Recall that we combine consecutive NTT matrices $\mathtt{N}_j$ to reduce the number of levels consumed by homomorphic NTT. We use a list P to represent a partition of $\mathtt{N}_j$'s. The list stores $n_{mats}+1$ integers in an increasing order with $P[0] = 1$ and $\mathtt{N}^{(i)} = \prod_{P[i] \leq j < P[i+1]} \mathtt{N}_j$ for $0 \leq i < n_{mats}$. We use the same P for CoeffToSlot and SlotToCoeff, although we could use different P for more fined-grained performance tuning.

The optimal partition for a fixed n_{mats} can be obtained using the dynamic programming method of Chen et al. [9]. However, their method requires an accurate estimation of the running time, which means one may have to benchmark the running time of a series of basic operations, including hoisting precomputation, hoisted automorphism, non-hoisted automorphism, plaintext-ciphertext multiplication (with plaintext in both double-CRT and non-double-CRT form), and ciphertext summation. Thus, considering the difficulty of obtaining an accurate model of the running time, we choose to determine the partitions experimentally through trial and error, which we believe suffices to demonstrate the effectiveness of our method. The obtained partitions are listed in Table 3.

Table 3. The partitions under different parameter sets for general and thin bootstrapping.

	Bootstrapping Type	I	Style	II	III
Partition	Thin	(1,6,12,16)	Bruun	(1,6,10,13)	(1,7,12,15)
			Radix-2	(1,5,9,13)	(1,6,10,15)
	General	(1,6,12,16)	Bruun	(1,5,10,13)	(1,7,12,15)
			Radix-2	(1,5,9,13)	(1,6,10,15)

5.2 Experimental Results

The benchmark results for thin bootstrapping are shown in Table 4 while those for general bootstrapping are in Table 5. The case IDs without primes or subscripts represent the baselines with corresponding parameter sets. I′ is the case of $p \equiv 1 \bmod 4$ under parameter set I. II_{Br} and III_{Br} use Bruun-style decomposition while II_{R2} and III_{R2} use Radix-2 style decomposition.

For thin bootstrapping, the algorithm proposed in [10] and refined in [16] is chosen as the baseline of comparison. Since the method in [10] only applies to thin bootstrapping, for general bootstrapping, the single-matrix representation of $\mathtt{Red}_{\mathrm{BR}}^{-1}$ (i.e., $n_{mats} = 1$) is taken as the baseline. For general bootstrapping, the running time of CoeffToSlot and SlotToCoeff includes the unpacking/repacking procedure before/after digit removal. The capacity of a ciphertext is defined as $\log_2$(ciphertext modulus/bound of ciphertext noise). The capacity needed by the next bootstrapping is subtracted from the remaining capacity, e.g., the capacity required by SlotToCoeff in thin bootstrapping or the decryption formula simplification process. The throughput of the bootstrapping procedure is defined as the remaining capacity divided by the running time, as in [15].

HElib stores the ring constants of a linear transformation (e.g., $\kappa(i)$ in Equation 1) in two ways, either as plain R_{p^r} elements or in the double-CRT form. The former format has lower memory cost while the latter leads to faster homomorphic computation at the cost of memory overhead. Thus, we only store these constants in the double-CRT form if they fit in the memory of our machine. Note that in all baselines but the baseline of II in thin bootstrapping, these constants will cause an out-of-memory error if represented in double-CRT form.

As shown in the tables, compared to the baselines where SlotToCoeff and CoeffToSlot are represented as a whole dense matrix, our NTT-like linear transformations run 7.35x∼63x faster in thin bootstrapping and 48.9x∼143x faster in general bootstrapping. Consequently, the throughput of thin bootstrapping is improved by 4.79x∼36.0x and the throughput of general bootstrapping is improved by 28.6x∼66.4x. Although the cases using our method consume more capacity than the baseline cases, they have much shorter running times, outweighing the extra capacity consumption and leading to a higher throughput.

Our method's advantage in running times is still significant even if the $\kappa(i)$'s are not stored in the double-CRT form. Moving from double-CRT to non-double-CRT will increase the running time of our methods by no more than 19.7%, but will double the running time of the baseline of II in thin bootstrapping. In this

Table 4. Benchmark results for thin bootstrapping. Capacity refers to the capacity consumed by each stage of bootstrapping. The speedup is computed as the ratio of throughput with respect to the baseline case.

Case ID		I	I′	II	II_{Br}	II_{R2}	III	III_{Br}	III_{R2}
Capacity (bits)	Initial	941	941	947	947	947	939	939	939
	SlotToCoeff	39	79	34	70	70	40	85	85
	CoeffToSlot	62	134	58	119	118	66	144	143
	Digit removal	265	264	232	231	232	277	276	277
	Remaining	556	446	609	511	513	537	415	415
Time (sec)	SlotToCoeff	99	3.8	31	3.4	2.8	255	4.3	3.3
	CoeffToSlot	522	10.8	89	12.9	10.1	686	13.8	11.6
	Digit removal	5.4	5.1	5.2	5.3	5.2	5.2	5.0	5.0
	Total	627	20.0	126	22.1	18.6	947	23.4	20.3
Throughput (bps)		0.89	22.2	4.84	23.2	27.6	0.57	17.7	20.4
Speedup		1x	25.1x	1x	4.79x	5.71x	1x	31.2x	36.0x

case, the throughput of our methods is still 8.36x$\sim$30.2x that of baselines in thin bootstrapping, and 24.7x$\sim$55.5x that of baselines in general bootstrapping.

For $p \equiv 3 \bmod 4$, the two styles of decomposition exhibit different running times (the cases of $\mathrm{II}_{Br}, \mathrm{III}_{Br}$ versus $\mathrm{II}_{R2}, \mathrm{III}_{R2}$ in Table 4 and Table 5). For general bootstrapping with a small d or thin bootstrapping (i.e., except for the cases $\mathrm{II}_{Br}/\mathrm{II}_{R2}$ in Table 5), the Radix-2 style decomposition is faster than the Bruun style because the NTT/INTT matrices in Radix-2 style have fewer nonzero diagonals. In general bootstrapping with a larger d (i.e., the cases $\mathrm{II}_{Br}/\mathrm{II}_{R2}$ in Table 5), however, the Bruun style one is faster than the Radix-2 style. This is because the computational overhead of BlockMatMul1D over MatMul1D grows with d. Recall that only one of the split NTT/INTT matrices in Brunn style is BlockMatMul1D, while all the NTT/INTT matrices in Radix-2 style are BlockMatMul1D. Thus, the disadvantage of having more BlockMatMul1D overweights the advantage of having fewer diagonals in each matrix, making the Radix-2-style transformation slower than the Bruun-style one.

References

1. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Homomorphic Encryption Security Standard. Tech. rep., HomomorphicEncryption.org, Toronto, Canada (November 2018)
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of Learning with Errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015). https://doi.org/doi:10.1515/jmc-2015-0016, `https://doi.org/10.1515/jmc-2015-0016`
3. Badawi, A.A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Quah, I., Polyakov,

Table 5. Benchmark results for general bootstrapping. Capacity refers to the capacity consumed by each stage of bootstrapping. The speedup is computed as the ratio of throughput with respect to the baseline case.

Case ID		I	I′	II	II_{Br}	II_{R2}	III	III_{Br}	III_{R2}
Capacity (bits)	Initial	918	918	927	927	927	915	915	915
	CoeffToSlot	54	126	86	148	157	91	169	169
	SlotToCoeff	54	126	83	156	154	90	168	168
	Digit extract	281	282	245	246	245	294	293	293
	Remaining	526	382	511	375	369	439	282	282
Time (sec)	CoeffToSlot	525	10.8	1579	17.6	21.5	1688	13.8	11.9
	SlotToCoeff	528	10.8	1579	16.1	18.0	1687	13.5	11.6
	Digit extract	5.3	4.9	42	39	40	10.1	8.8	8.8
	Total	1059	26.9	3200	73	80	3386	36.5	32.3
Throughput (bps)		0.50	14.2	0.16	5.1	4.6	0.13	7.7	8.6
Speedup		1x	28.6x	1x	32.0x	28.9x	1x	59.7x	66.4x

Y., R.V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., Zucca, V.: OpenFHE: Open-Source Fully Homomorphic Encryption Library. Cryptology ePrint Archive, Paper 2022/915 (2022), https://eprint.iacr.org/2022/915, https://eprint.iacr.org/2022/915

4. Blatt, M., Gusev, A., Polyakov, Y., Rohloff, K., Vaikuntanathan, V.: Optimized homomorphic encryption solution for secure genome-wide association studies. BMC Medical Genomics **13**(7), 83 (Jul 2020). https://doi.org/10.1186/s12920-020-0719-9, https://doi.org/10.1186/s12920-020-0719-9
5. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) Fully Homomorphic Encryption without Bootstrapping. ACM Trans. Comput. Theory **6**(3) (jul 2014). https://doi.org/10.1145/2633600, https://doi.org/10.1145/2633600
6. Bruun, G.: z-transform DFT filters and FFT's. IEEE Transactions on Acoustics, Speech, and Signal Processing **26**(1), 56–63 (1978). https://doi.org/10.1109/TASSP.1978.1163036
7. Cantor, D.G., Kaltofen, E.: On fast multiplication of polynomials over arbitrary algebras. Acta Informatica **28**(7), 693–701 (Jul 1991). https://doi.org/10.1007/BF01178683, https://doi.org/10.1007/BF01178683
8. Chen, H.T., Chung, Y.H., Hwang, V., Liu, C.T., Yang, B.Y.: Algorithmic Views of Vectorized Polynomial Multipliers for NTRU and NTRU Prime (Long Paper). Cryptology ePrint Archive, Paper 2023/541 (2023), https://eprint.iacr.org/2023/541, https://eprint.iacr.org/2023/541
9. Chen, H., Chillotti, I., Song, Y.: Improved Bootstrapping for Approximate Homomorphic Encryption. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019. pp. 34–54. Springer International Publishing, Cham (2019)
10. Chen, H., Han, K.: Homomorphic Lower Digits Removal and Improved FHE Bootstrapping. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018. pp. 315–337. Springer International Publishing, Cham (2018)
11. Chen, H., Han, K.: Homomorphic Lower Digits Removal and Improved FHE Bootstrapping. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018. pp. 315–337. Springer International Publishing, Cham (2018)
12. Cong, K., Moreno, R.C., da Gama, M.B., Dai, W., Iliashenko, I., Laine, K., Rosenberg, M.: Labeled PSI from Homomorphic Encryption with Re-

duced Computation and Communication. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. p. 1135–1150. CCS '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3460120.3484760, https://doi.org/10.1145/3460120.3484760
13. Cooley, J.W., Tukey, J.W.: An Algorithm for the Machine Calculation of Complex Fourier Series. Mathematics of Computation **19**(90), 297–301 (1965), http://www.jstor.org/stable/2003354
14. Fan, J., Vercauteren, F.: Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2012/144 (2012), https://eprint.iacr.org/2012/144, https://eprint.iacr.org/2012/144
15. Geelen, R., Iliashenko, I., Kang, J., Vercauteren, F.: On Polynomial Functions Modulo p^e and Faster Bootstrapping for Homomorphic Encryption. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 257–286. Springer Nature Switzerland, Cham (2023)
16. Geelen, R., Vercauteren, F.: Bootstrapping for BGV and BFV Revisited. Journal of Cryptology **36**(2), 12 (Mar 2023). https://doi.org/10.1007/s00145-023-09454-6, https://doi.org/10.1007/s00145-023-09454-6
17. Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing. p. 169–178. STOC '09, Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1536414.1536440, https://doi.org/10.1145/1536414.1536440
18. Halevi, S., Shoup, V.: Bootstrapping for HElib. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015. pp. 641–670. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
19. Halevi, S., Shoup, V.: Faster Homomorphic Linear Transformations in HElib. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018. pp. 93–120. Springer International Publishing, Cham (2018)
20. Halevi, S., Shoup, V.: Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Paper 2020/1481 (2020), https://eprint.iacr.org/2020/1481, https://eprint.iacr.org/2020/1481
21. Halevi, S., Shoup, V.: Bootstrapping for HElib. Journal of Cryptology **34**(1), 7 (Jan 2021). https://doi.org/10.1007/s00145-020-09368-7, https://doi.org/10.1007/s00145-020-09368-7
22. Han, K., Hhan, M., Cheon, J.H.: Improved Homomorphic Discrete Fourier Transforms and FHE Bootstrapping. IEEE Access **7**, 57361–57370 (2019). https://doi.org/10.1109/ACCESS.2019.2913850
23. Lattigo v5. Online: https://github.com/tuneinsight/lattigo (Nov 2023), ePFL-LDS, Tune Insight SA
24. Lee, J.W., Kang, H., Lee, Y., Choi, W., Eom, J., Deryabin, M., Lee, E., Lee, J., Yoo, D., Kim, Y.S., No, J.S.: Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network. IEEE Access **10**, 30039–30054 (2022). https://doi.org/10.1109/ACCESS.2022.3159694
25. Ma, S., Huang, T., Wang, A., Wang, X.: 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, https://eprint.iacr.org/2024/115
26. Meyn, H.: Factorization of the Cyclotomic Polynomialx2n+ 1 over Finite Fields. Finite Fields and Their Applications **2**(4), 439–442

(1996). https://doi.org/https://doi.org/10.1006/ffta.1996.0026, https://www.sciencedirect.com/science/article/pii/S107157979690026X
27. Ng, L.K.L., Chow, S.S.M.: GForce: GPU-Friendly Oblivious and Rapid Neural Network Inference. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 2147–2164. USENIX Association (Aug 2021), https://www.usenix.org/conference/usenixsecurity21/presentation/ng
28. Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL (Jan 2023), microsoft Research, Redmond, WA.
29. Wan, Z.: Lectures on Finite Fields and Galois Rings. G - Reference,Information and Interdisciplinary Subjects Series, World Scientific (2003), https://books.google.com.hk/books?id=uCSVbYMljNIC

Supplementary Material

A Proofs of Lemma 4 and Lemma 5

Proof (Lemma 4).

Concerning $\mathbf{N}_1$, for $i \in [D/2]$ and $k \in [2]$, define

$$\begin{cases} \mathbf{l} = \boldsymbol{\alpha}_0[i][0 +: d/2] \\ \mathbf{h} = \boldsymbol{\alpha}_0[i][d/2 +: d/2] \\ \mathbf{l}' = \boldsymbol{\alpha}_0[i + D/2][0 +: d/2] \\ \mathbf{h}' = \boldsymbol{\alpha}_0[i + D/2][d/2 +: d/2] \end{cases}, \begin{cases} \mathbf{a}_{00} = \boldsymbol{\alpha}_1[i][0 +: d/2] \\ \mathbf{a}_{01} = \boldsymbol{\alpha}_1[i][d/2 +: d/2] \\ \mathbf{a}_{10} = \boldsymbol{\alpha}_1[i + D/2][d/2 +: d/2] \\ \mathbf{a}_{11} = \boldsymbol{\alpha}_1[i + D/2][d/2 +: d/2] \end{cases}.$$

By traversing i and k, $\mathbf{l}, \mathbf{h}, \mathbf{l}', \mathbf{h}'$ and $\mathbf{a}_{00}, \ldots, \mathbf{a}_{11}$ cover all the inputs and outputs of $\mathbf{N}_1$. $\mathbf{a}_{00}, \ldots, \mathbf{a}_{11}$ are $\mathbb{Z}_p$-linear combinations of $\mathbf{l}, \mathbf{h}, \mathbf{l}', \mathbf{h}'$ because they form a Bruun butterfly with respect to $f_0(X) = F_i$ and $f_1(X) = F_{i+D/2}$ as in Equation 3, which can be deduced from the definition of $\boldsymbol{\alpha}_j$ and $\boldsymbol{\alpha}_{j-1}$. The linear combinations correspond to a 2×2 submatrix in $\mathbf{N}_1$

$$\begin{bmatrix} \mathbf{N}_1[i, i] & \mathbf{N}_1[i, i + D/2] \\ \mathbf{N}_1[i + D/2, i] & \mathbf{N}_1[i + D/2, i + D/2] \end{bmatrix}.$$

Each entry is in the form of $\begin{bmatrix} a_0\mathbf{I}_{d/2} & a_1\mathbf{I}_{d/2} \\ a_2\mathbf{I}_{d/2} & a_3\mathbf{I}_{d/2} \end{bmatrix}$ for some $a_0, \ldots, a_3 \in \mathbb{Z}_p$. Traversing i will expand the submatrix into $\mathbf{N}_1$. Thus, $\mathbf{N}_1$ has three nonzero diagonals indexed as $D/2 \times \{-1, 0, 1\}$. The structure of $\mathbf{N}_1^{-1}$ can be proved similarly.

Concerning $\mathbf{N}_j$ with $j \in [2, \log_2(D)]$, for $i \in [2^{-j+1}D]$ and $k_0 \in [2^{j-2}]$,

$$\begin{aligned} \mathbf{a}_{00} &= \boldsymbol{\alpha}_j[i + \mathrm{BitRev}_{j,0}(k_0) \cdot 2^{-j}D] \\ \mathbf{a}_{01} &= \boldsymbol{\alpha}_j[i + \mathrm{BitRev}_{j,0}(k_0 + 2^{j-2}) \cdot 2^{-j}D] \\ \mathbf{a}_{10} &= \boldsymbol{\alpha}_j[i + \mathrm{BitRev}_{j,0}(k_0 + 2 \cdot 2^{j-2}) \cdot 2^{-j}D] \\ \mathbf{a}_{11} &= \boldsymbol{\alpha}_j[i + \mathrm{BitRev}_{j,0}(k_0 + 3 \cdot 2^{j-2}) \cdot 2^{-j}D] \end{aligned}$$

are $\mathbb{Z}_p$-linear combinations of

$$\begin{aligned} \mathbf{l} &= \boldsymbol{\alpha}_{j-1}[i + \mathrm{BitRev}_{j-1,0}(k_0) \cdot 2^{-j+1}D] \\ \mathbf{h} &= \boldsymbol{\alpha}_{j-1}[i + \mathrm{BitRev}_{j-1,0}(k_0 + 2^{j-2}) \cdot 2^{-j+1}D] \\ \mathbf{l}' &= \boldsymbol{\alpha}_{j-1}[i + 2^{-j}D + \mathrm{BitRev}_{j-1,0}(k_0) \cdot 2^{-j+1}D] \\ \mathbf{h}' &= \boldsymbol{\alpha}_{j-1}[i + 2^{-j}D + \mathrm{BitRev}_{j-1,0}(k_0 + 2^{j-2}) \cdot 2^{-j+1}D], \end{aligned}$$

because they form a Bruun butterfly with respect to $F_i^{(j-1)}$ and $F_{i+2^{-j}D}^{(j-1)}$ as in Equation 3, which can be deduced from the definition of $\boldsymbol{\alpha}_j$ and $\boldsymbol{\alpha}_{j-1}$. By traversing i and k_0, $\mathbf{l}, \mathbf{l}', \mathbf{h}, \mathbf{h}'$ and $\mathbf{a}_{00}, \mathbf{a}_{10}, \mathbf{a}_{01}, \mathbf{a}_{11}$ cover all the inputs and outputs of $\mathbf{N}_j$. Observe that $\mathbf{a}_{00}, \mathbf{a}_{10}, \mathbf{a}_{01}, \mathbf{a}_{11}$ and $\mathbf{l}, \mathbf{l}', \mathbf{h}, \mathbf{h}'$ share the same index

s_0, s_1, s_2, s_3 in sequence, where $s_t = i + (\mathrm{BitRev}_{j,0}(k_0) + t) \cdot 2^{-j}D$. Thus, the linear combinations between them correspond to a 4×4 submatrix in $\mathbf{N}_j$

$$\begin{bmatrix} \mathbf{N}_j[s_0, s_0] & \cdots & \mathbf{N}_j[s_0, s_3] \\ \vdots & \ddots & \vdots \\ \mathbf{N}_j[s_3, s_0] & \cdots & \mathbf{N}_j[s_3, s_3] \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ & & * & * \\ * & * & * & * \\ * & * & & \end{bmatrix},$$

where a '$*$' means a nonzero multiple of $\mathbf{I}_d$. Traversing i for a fixed value of k_0 will expand the submatrix into a $2^{-j+2}D$-sized diagonal block in $\mathbf{N}_j$, whose nonzero diagonals are indexed by $\{s_u - s_v \mid u, v \in [4]\} = 2^{-j}D \times [-3, 3]$. The structure of $\mathbf{N}_j^{-1}$ can be proved by expressing $\mathbf{l}, \mathbf{h}, \mathbf{l}', \mathbf{h}'$ as $\mathbb{Z}_p$-linear combinations of $\mathbf{a}_{00}, \ldots, \mathbf{a}_{11}$.

□

Proof (Lemma 5).

Concerning a fixed $j \in [1, \log_2(D)]$, for $i \in [2^{-j}D], k \in [2^{j-1}]$,

$$\begin{aligned} \mathbf{a}_{00} &= \boldsymbol{\alpha}'_j[2(i + 2k \cdot 2^{-j}D)] \\ \mathbf{a}_{10} &= \boldsymbol{\alpha}'_j[2(i + 2k \cdot 2^{-j}D) + 1] \\ \mathbf{a}_{01} &= \boldsymbol{\alpha}'_j[2(i + (2k+1) \cdot 2^{-j}D)] \\ \mathbf{a}_{11} &= \boldsymbol{\alpha}'_j[2(i + (2k+1) \cdot 2^{-j}D) + 1] \end{aligned}$$

are $\mathbb{Z}_p$-linear combinations of

$$\begin{aligned} \mathbf{l} &= \boldsymbol{\alpha}'_{j-1}[2(i + k \cdot 2^{-j+1}D)] \\ \mathbf{h} &= \boldsymbol{\alpha}'_{j-1}[2(i + k \cdot 2^{-j+1}D) + 1] \\ \mathbf{l}' &= \boldsymbol{\alpha}'_{j-1}[2(i + 2^{-j}D + k \cdot 2^{-j+1}D)] \\ \mathbf{h}' &= \boldsymbol{\alpha}'_{j-1}[2(i + 2^{-j}D + k \cdot 2^{-j+1}D) + 1], \end{aligned}$$

because they form a Bruun butterfly with respect to $F_i^{(j-1)}$ and $F_{i+2^{-j}D}^{(j-1)}$ as in Equation 3, which can be deduced from the definition of $\boldsymbol{\alpha}'_j$ and $\boldsymbol{\alpha}'_{j-1}$. By traversing i and k, $\mathbf{l}, \mathbf{h}, \mathbf{l}', \mathbf{h}'$ and $\mathbf{a}_{00}, \mathbf{a}_{10}, \mathbf{a}_{01}, \mathbf{a}_{11}$ cover all the inputs and outputs of $\mathbf{N}_j$. The index of $\mathbf{a}_{00}, \mathbf{a}_{10}$ in $\boldsymbol{\alpha}_j$ and the index of $\mathbf{l}, \mathbf{h}$ in $\boldsymbol{\alpha}_{j-1}$ are both $s = i + k \cdot 2^{-j+1}D$. $\mathbf{a}_{01}, \mathbf{a}_{11}$ and $\mathbf{l}, \mathbf{h}$ also share the same index $t = i + 2^{-j}D + k \cdot 2^{-j+1}D$. Thus, the linear combinations correspond to a 2×2 submatrix in $\mathbf{N}'_j$

$$\begin{bmatrix} \mathbf{N}'_j[s, s] & \mathbf{N}'_j[s, t] \\ \mathbf{N}'_j[t, s] & \mathbf{N}'_j[t, t] \end{bmatrix},$$

where each entry has the form of $\begin{bmatrix} a_0\mathbf{I}_{d/2} & a_1\mathbf{I}_{d/2} \\ a_2\mathbf{I}_{d/2} & a_3\mathbf{I}_{d/2} \end{bmatrix}$ for $a_0, \ldots, a_3 \in \mathbb{Z}_p$. Traversing i for a fixed value of k will expand the submatrix into a $2^{-j+1}D$-sized diagonal block in $\mathbf{N}'_j$, which has three nonzero diagonals indexed as $\{0, \pm(s - t)\} = 2^{-j}D \times \{-1, 0, 1\}$. The structure of $\mathbf{N}'^{-1}_j$ can be proved similarly.

□