Faster Homomorphic Discrete Fourier Transforms and Improved FHE Bootstrapping

Jung Hee Cheon[1], Kyoohyung Han[1], and Minki Hhan[1]

Seoul National University, Seoul Gwanak-ro 1 08826, Korea
{jhcheon, satanigh, hhan_}@snu.ac.kr

Abstract. In this work, we propose a faster homomorphic linear transform algorithm for structured matrices such as the discrete Fourier transform (DFT) and linear transformations in bootstrapping.
First, we proposed new method to evaluate the DFT homomorphically for a given packed ciphertext from the Cooley-Tukey fast Fourier transform algorithm. While the previous method requires $O(\sqrt{n})$ rotations and $O(n)$ constant vector multiplications, our method only needs $O(\log n)$ rotations/multiplications by consuming $O(\log n)$ depth for the length of input vector n.
Second, we apply the same method to the linear transform of bootstrapping for HEAAN. To achieve this, we construct a recursive relation of matrices in those linear transformations. Accordingly, we can highly accelerate the linear transformation part of bootstrapping: the number of homomorphic operations becomes logarithmic to the number of slots, as in homomorphic DFT.
We also implement both algorithms. Our homomorphic DFT with length 2^{14} only takes about 8 seconds which is about 150 times faster result than previous one. The bootstrapping for HEAAN with our linear transform algorithm takes about 2 minutes for $\mathbb{C}^{32768}$ plaintext space with 8 bit precision, which takes 26 hours using the previous method.

Keywords: Discrete Fourier Transformation · Fully Homomorphic Encryption · Bootstrapping.

1 Introduction

Following Gentry's blueprint [10], various schemes and techniques have been suggested for efficient fully homomorphic encryption (FHE) schemes. One of the most important techniques for FHE is to encrypt multiple messages in one ciphertext, called *packing*. Homomorphic operation between packed ciphertexts or single instruction multiple data (SIMD) allows the entry-wise operation of packed ciphertexts.

Because a large number of messages can be encrypted in single ciphertext as a vector, applying linear transformations on packed ciphertext is an important task for homomorphic encryption. For this reason, many studies have been done to improve the efficiency of general linear transformations [11, 12]. However, while the discrete Fourier transform (DFT) and linear transformations in

bootstrapping have in common a special structure that has been overlooked. No previous study has exploited this structure for linear transformation on packed ciphertext.

The DFT is a widely used tool in various fields; digital data processing, data compression, partial differential equations, etc. For example, it is often used to remove noise sound with small frequency. However, in many applications, the DFT is applied to the private data such as face, voice, and bioinformation data. Therefore, homomorphic evaluation of DFT is necessary for data processing with privacy preserving.

Bootstrapping, the only known way to refresh the noise in the ciphertext without decryption, is crucial in evaluating large depth circuit or unlimited number of operations. Linear transformations which convert between coefficient and slot representations serve as a central role in the bootstrapping procedure. When the number of slots is large, homomorphic evaluation of the linear transformation becomes a bottleneck for the performance of bootstrapping. For this reason, our goal is to build an improved method for homomorphic evaluation of these two linear transformations.

Our Results. In this paper, we study the fast linear transformations for special structured matrices. First, we propose a new way to evaluate discrete Fourier transformation for a given packed ciphertext. Our method only needs $O(\log n)$ number of homomorphic operations while the previous method requires $O(\sqrt{n})$ rotations and $O(n)$ constant vector multiplications for n the length of input vector.

We factorize the DFT matrix into $\log_2 n$ sparse block diagonal matrices using the Cooley-Tukey factorization with radix 2. We observe that each factor has only three diagonal vectors, and each $\log_2 k$ consecutive multiplication of those factors has $(2k-1)$ diagonal vectors. Therefore, homomorphic DFT evaluation is converted to $\log_k n$ number of homomorphic matrix multiplications for matrix with $(2k-1)$ diagonal vectors for an arbitrary integer k dividing n.

From SIMD operation of HE schemes, evaluating matrices with d diagonal vector in encrypted state can be done with $O(\sqrt{d})$ homomorphic rotations and d homomorphic constant vector multiplications using the baby-step algorithm. So, we obtain a homomorphic DFT algorithm which needs $O(\sqrt{k}\log n)$ number of homomorphic rotations and $O(k\log n)$ number of homomorphic constant vector multiplications with $O(\log_k n)$ constant vector multiplication depth. In addition, we can obtain a trade-off between depth and complexity by adjusting k.

Second, we apply the same matrix decomposition strategy into sparse diagonal matrices to improve the linear transformations in bootstrapping for HEAAN. We decompose corresponding matrices recursively, similarly to the Cooley-Tukey algorithm. As a result we obtain the same improvement in the linear transformations in bootstrapping: $O(\sqrt{k}\log n)$ homomorphic rotations and $O(k\log n)$ homomorphic constant vector multiplications with $O(\log_k n)$ constant vector multiplication depth for plaintext vector length n.

We also implement our method using the approximate homomorphic encryption library [13] to show the improvements. Our implementation shows that the

homomorphic DFT with length 2^{13} only takes about 8 seconds when $k = 2$. This results shows a more than 150× performance improvement compared to previous works on homomorphic DFT (or FFT) [6,8,9]. On the other hand, the bootstrapping procedure for HEAAN using our linear transformation algorithm only takes 2 minutes for $\mathbb{C}^{32768}$ plaintext space with 8-bit precision. This result yields an amortized rate per bits of 0.45ms, less than one millisecond. The previous algorithm takes 26 hours in the same setting, which is only realistic for a small number of slots.

Related Works. Due to the importance of DFT (or FFT), there are several works that perform DFT on encrypted domains. In [2, 3], the authors present works on homomorphic FFT using the Paillier encryption algorithm. In [8, 9], the authors used homomorphic encryption library named HElib with different encoding method for real (or complex) messages. In [6], the authors implemented homomorphic evaluation of multiple FFTs using approximate homomorphic encryption scheme.

All of those previous works encrypt each element of input vector in different ciphertext. For this reason, the previous algorithms require as least $O(n \log n)$ ciphertext and homomorphic operations, which worsens performance on the DFT and on bootstrapping. If they put several messages in one ciphertext using packing, these works can achieve $O(\log n)$ homomorphic operation complexity only in amortized sense [6].

Homomorphic encryption schemes with packed message use the linear transformations in bootstrapping, which convert the slots of messages and coefficients of polynomial. For this linear transformation part, a general linear transformation method which is called the baby-step giant-step is usually used [4, 5, 11]. This method needs $O(\sqrt{n})$ key-switchings for the length of plaintext vector n. Especially, when the underlying ring is the (tensor) product of two rings, the linear transformation in bootstrapping can be decomposed into transformations in each factor ring. In this case, the number of key-switching is $O(\sqrt{n_1} + \sqrt{n_2})$ where the factor rings are of dimension n_1, n_2.

Road Map. In Section 2, we define the standard notations and briefly introduce the approximate homomorphic encryption scheme. We then introduce homomorphic DFT algorithm including previous approach in Section 3. We apply our new method in bootstrapping for approximate homomorphic encryption in Section 4. Each sections includes implementation results of homomorphic DFT and bootstrapping.

2 Preliminary

Notations. Column vectors are written by bold and lower case letters and matrices are written by bold and upper case letters. The entries of bold face is denoted as $\boldsymbol{v} = (v_0, v_1, \cdots, v_{n-1})^T$ and $\boldsymbol{M} = (M_{i,j})_{1 \le i,j \le n}$. We sometimes take modular n for indices of vector or matrices, and omit the transpose operator T. The

entry-wise multiplication of two vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ is denoted by $\boldsymbol{v}_1 \odot \boldsymbol{v}_2$ which is called *Hadamard* multiplication. For the given vector $\boldsymbol{v}$ with length n, $\mathsf{diag}_i(\boldsymbol{v})$ is n by n matrix $\boldsymbol{M}$ such that $M_{j,j+i} = v_j$ for $0 \leq j < n$ and all other entries are zero. We will omit the index i of diag when $i = 0$. On the other hand, for n by n matrix $\boldsymbol{M}$, $\mathsf{diag}_i(\boldsymbol{M})$ denotes a length n vector $(M_{0,i}, M_{1,1+i}, \cdots, M_{n-1,n-1+i})$. $\mathsf{rot}_i(\boldsymbol{v})$ is left shifted vector with index i, this means that the result vector $\boldsymbol{w}$ is $(v_i, v_{i+1}, \ldots, v_{i-1})$. When the index i is negative, it means right shifting with index $-i$.

We sometimes use the special order of indices called *bit-reversal* order. It is defined by ordering the indices in increasing order of the reverse of binary representations that are padded so that each of these binary representation has the same length. For example, bit-reversal order of the given array (a_0, a_1, a_2, a_3) is (a_0, a_2, a_1, a_3) (because bit-reversed index is follows: $(00_{(2)}, 10_{(2)}, 01_{(2)}, 11_{(2)}) = (0, 2, 1, 3)$).

2.1 Discrete Fourier Transforms

The discrete Fourier transform (DFT) is a linear transformation $\mathrm{DFT}_n : \mathbb{C}^n \to \mathbb{C}^n$ which maps a vector $\boldsymbol{x} = (x_1, \cdots, x_n)$ into another vector $\boldsymbol{y} = (y_1, \cdots, y_n)$ where

$$y_m = \sum_{k=0}^{n-1} x_k \cdot w_n^{km}$$

for $w_n = e^{2\pi i/n}$. The inverse of discrete Fourier transform iDFT_n has a similar form as $x_m = \frac{1}{n}\sum_{k=0}^{n-1} y_k \cdot w_n^{-km}$, which also can be expressed by DFT_n as $\mathrm{iDFT}_n = \overline{\mathrm{DFT}_n(\overline{\boldsymbol{x}})}/n$ (here division and $\overline{\boldsymbol{x}}$ means element-wise division and conjugation). This algorithm is known to be computed in $O(n \log n)$ operations using so-called *fast Fourier transform* (FFT).

2.2 Approximate Homomorphic Encryption

In our paper, we will focus on the DFT on complex field. For this reason, we need homomorphic encryption for complex arithmetic. At 2017, homomorphic encryption scheme for approximate number arithmetic is proposed by Cheon et al. [6] which is called HEAAN. The plaintext structure of this scheme is $\mathbb{C}^{N/2}$ for polynomial ring dimension N, and it is suitable for our purpose. This subsection gives scheme description and function definition for HEAAN scheme (for more information about this scheme refer to [5,6]).

Let $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$ for ciphertext modulus q and power of two N. For the given $\sigma > 0$, $\mathcal{DG}(\sigma^2)$ denotes distribution on $\mathcal{R}$ that each coefficient follows discrete Gaussian distribution over $\mathbb{Z}$ with standard deviation σ. For this given $h > 0$, $\mathcal{HWT}(h)$ denotes a uniform distribution on a set of $\mathcal{R}$ with signed binary $\{\pm 1\}$ coefficients and hamming weight exactly h. For a real $0 \leq \rho \leq 1$, $\mathcal{ZO}(\rho)$ denotes a distribution on $\mathcal{R}$ such that each coefficients is $+1$ with probability $\rho/2$, -1 with probability $\rho/2$, and 0 with probability $1-\rho$.

- $\mathsf{KeyGen}(1^\lambda)$
  - Let $q_i = p^i$ for $i = 1, \ldots, L$. Using the given the security parameter λ, we choose a power-of-two integer N, an integer h, an integer $P > q_L$, and a real number $\sigma > 0$ to achieve λ-bit security level.
  - Sample $s(x) \leftarrow \mathcal{HWT}(h)$, $a(x) \leftarrow U(\mathcal{R}_{q_L})$ and $e(x) \leftarrow \mathcal{DG}(\sigma^2)$. Set the secret key $\mathsf{sk} = (1, s(x))$ and the public key for encryption as $\mathsf{pk}_{\mathsf{enc}} \leftarrow (b(x), a(x)) \in \mathcal{R}_{q_L}^2$ where $b(x) \leftarrow -a(x) \cdot s(x) + e(x) \in \mathcal{R}_{q_L}$
- $\mathsf{KeySwitchGen}(s'(x), \mathsf{sk})$
  - Sample $a'(x) \leftarrow U(R_{P \cdot q_L})$ and $e' \leftarrow \mathcal{DG}(\sigma^2)$. Set the public key for key switching as $(b'(x), a'(x)) \in \mathcal{R}_{P \cdot q_L}$ where $b'(x) = -a'(x) \cdot s(x) + e'(x) + P \cdot s'(x) \in \mathcal{R}_{P \cdot q_L}$.
  - The method for encoding can be understood as negacyclic DFT and the group $\mathbb{Z}_M^\times$ is generated by 5 and 2 for $M = 2N$ (see [5]). For this reason, three public keys for homomorphic multiplication and rotation and conjugation are generated as follows:

$$\begin{aligned}
\mathsf{pk}_{\mathsf{mult}} &= \mathsf{KeySwitchGen}(s^2(x), \mathsf{sk}), \\
\mathsf{pk}_{\mathsf{rot}}^{\mathsf{idx}} &= \mathsf{KeySwitchGen}(s(x^{5^{\mathsf{idx}}}), \mathsf{sk}), \\
\mathsf{pk}_{\mathsf{conj}} &= \mathsf{KeySwitchGen}(s(x^2), \mathsf{sk}).
\end{aligned}$$

- $\mathsf{Encode}(\boldsymbol{m})$
  - Let

$$\boldsymbol{U} = \begin{bmatrix} 1 & w_0 & w_0^2 & \cdots & w_0^{N-1} \\ 1 & w_1 & w_1^2 & \cdots & w_1^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_{N/2-1} & w_{N/2-1}^2 & \cdots & w_{N/2-1}^{N-1} \end{bmatrix}$$

    for $w_i = w^{5^i}$ and $w = \exp(2\pi i/M)$ for $M = 2N$.
  - Output $f(x) = \sum_{i=0}^{N-1} f_i X^i$ such that $\boldsymbol{f} = (f_i)_{0 \le i < N} = \frac{1}{N}(\overline{U}^T \cdot \boldsymbol{m} + U^T \cdot \overline{\boldsymbol{m}})$.
- $\mathsf{Decode}(f(x))$
  - Output $\boldsymbol{m} = U \cdot \boldsymbol{f}$ such that $\boldsymbol{f} = (f_i)_{0 \le i < N}$ for $f(x) = \sum_{i=0}^{N-1} f_i X^i$.
- $\mathsf{Encrypt}(\boldsymbol{m} \in \mathbb{C}^{N/2}, p^k, \mathsf{pk}_{\mathsf{enc}})$
  - Let $m(x) = \mathsf{Encode}(\boldsymbol{m})$.
  - For $\mathsf{pk}_{\mathsf{enc}} = (b(x), a(x))$, output $c = (\lfloor p^k \cdot m(x) \rceil + v(x) \cdot b(x) + e_1(x), v(x) \cdot a(x) + e_2(x))$ for $v(x) \leftarrow \mathcal{ZO}(\rho)$ and $e_1(x), e_2(x) \leftarrow DG(\sigma^2)$.
- $\mathsf{Decrypt}(c, p^k, \mathsf{sk})$
  - For $c = (b(x), a(x)) \in \mathcal{R}_{q_i}^2$, compute $\langle (b(x), a(x), \mathsf{sk} \rangle = M(x) \in \mathcal{R}_{q_i}$.
  - Output $\boldsymbol{m} = \mathsf{Decode}(M(x)/p^k \in \mathbb{R}[X]/(X^N + 1))$.
- $\mathsf{Add}(c_1, c_2)$
  - For $c_1 = (b_1(x), a_1(x)) \in \mathcal{R}_{q_i}^2$ and $c_2 = (b_2(x), a_2(x)) \in \mathcal{R}_{q_i}^2$, output $c_3 = (b_1(x) + b_2(x), a_1(x) + a_2(x)) \in \mathcal{R}_{q_i}^2$.

- $\mathsf{CMult}(\boldsymbol{m} \in \mathbb{C}^{N/2}, c, p^k)$
  - Let $m(x) = \mathsf{Encode}(\boldsymbol{m})$.
  - For $c = (b(x), a(x)) \in \mathcal{R}_{q_i}^2$, output $c' = (m'(x) \cdot b(x), m'(x) \cdot a(x)) \in \mathcal{R}_{q_i}^2$ for $m'(x) = \lfloor p^k \cdot m(x) \rceil$.
- $\mathsf{Mult}(c_1, c_2, \mathsf{pk}_{\mathsf{mult}})$
  - For $c_1 = (b_1(x), a_1(x)) \in \mathcal{R}_{q_i}^2$ and $c_2 = (b_2(x), a_2(x)) \in \mathcal{R}_{q_i}^2$, compute $(d_0(x), d_1(x), d_3(x)) = (b_1(x) \cdot b_2(x), b_1(x) \cdot a_2(x) + b_2(x) \cdot a_1(x), a_1(x) \cdot a_2(x)) \in \mathcal{R}_{q_i}^3$.
  - For $\mathsf{pk}_{\mathsf{mult}} = (B(x), A(x)) \in \mathcal{R}_{P \cdot q_i}$, output $c_3 = (d_0(x) + \lfloor P^{-1} \cdot d_2(x) \cdot B(x) \rceil, d_1(x) + \lfloor P^{-1} \cdot d_2(x) \cdot A(x) \rceil) \in \mathcal{R}_{q_i}^2$ (here $d_2(x) \cdot A(x)$ and $d_2(x) \cdot B(x)$ are operations in $\mathcal{R}_{P \cdot q_i}$).
- $\mathsf{LeftRotate}(c, \mathsf{idx}, \mathsf{pk}_{\mathsf{rot}})$
  - For $c = (b(x), a(x)) \in \mathcal{R}_{q_i}$, compute $c' = (b'(x), a'(x)) = (b(x^{5^{\mathsf{idx}}}), a(x^{5^{\mathsf{idx}}}))$.
  - For $\mathsf{pk}_{\mathsf{rot}}^{\mathsf{idx}} = (B(x), A(x)) \in \mathcal{R}_{P \cdot q_i}$, output $c'' = (\lfloor P^{-1} \cdot b'(x) \cdot B(x) \rceil, a'(x) + \lfloor P^{-1} \cdot b'(x) \cdot A(x) \rceil) \in \mathcal{R}_{q_i}^2$ (here $b'(x) \cdot A(x)$ and $b'(x) \cdot B(x)$ are operations in $\mathcal{R}_{P \cdot q_i}$).
- $\mathsf{Conj}(c, \mathsf{pk}_{\mathsf{conj}})$
  - For $c = (b(x), a(x)) \in \mathcal{R}_{q_i}$, compute $c' = (b'(x), a'(x)) = (b(x^2), a(x^2))$.
  - For $\mathsf{pk}_{\mathsf{conj}} = (B(x), A(x)) \in \mathcal{R}_{P \cdot q_i}$, output $c'' = (\lfloor P^{-1} \cdot b'(x) \cdot B(x) \rceil, a'(x) + \lfloor P^{-1} \cdot b'(x) \cdot A(x) \rceil) \in \mathcal{R}_{q_i}^2$ (here $b'(x) \cdot A(x)$ and $b'(x) \cdot B(x)$ are operations in $\mathcal{R}_{P \cdot q_i}$).
- $\mathsf{Rescale}(c, p^k)$
  - For $c = (b(x), a(x)) \in \mathcal{R}_{q_i}^2$, output $(\lfloor p^{-k} \cdot b(x) \rceil, \lfloor p^{-k} \cdot a(x) \rceil) \in \mathcal{R}_{q_{i-k}}^2$.

Note that we need to multiply p^k to convert real polynomial to integer polynomial. For this reason, we also need to do Rescale after homomorphic multiplication between constant vector and encrypted vector. Each Rescale consumes one level, so we will regard that CMult as depth 1 computation in our paper. Furthermore, right rotation in encrypted state is obtained from left rotation, we just need to use index $(N/2 - \mathsf{idx})$.

3 Homomorphic Discrete Fourier Transforms

In this section, we briefly review the previous approach to evaluate DFT with homomorphic encryption (HE) and describe our new homomorphic DFT algorithm. We propose new homomorphic DFT algorithm and also hybrid algorithm that combines our new method with previous approach.

3.1 Previous Approach

In [11], they proposed faster linear transformation ($\simeq$ NTT) for bootstrapping when the input size of $\phi(m)$ (here m is product of co-prime integers m_i). They understand one variable polynomial ring as multivariate with special basis which

is called *powerful basis*. This approach shows that DFT with dimension m can be split to several number of DFT with m_i for co-prime m_is.

On the other hand, in the case of power of prime dimension, there is no specialized algorithm for homomorphic DFT. Previously known approaches apply a general homomorphic linear transform with DFT matrix to the ciphertext [4, 6]. We review the key ideas of these approaches. HE schemes support Hadamard multiplication and rotation for the plaintext vector. The following equation shows a representation of matrix-vector multiplication via Hadamard multiplications and rotations.

$$\begin{aligned}
\boldsymbol{M} \cdot \boldsymbol{v} &= \sum_{i=0}^{n} \mathsf{diag}_i(\boldsymbol{M}) \odot \mathsf{rot}_i(\boldsymbol{v}) \\
&= \sum_{i=0}^{\ell} \sum_{j=0}^{k} \mathsf{diag}_{ki+j}(\boldsymbol{M}) \odot \mathsf{rot}_{ki+j}(\boldsymbol{v}) \\
&= \sum_{i=0}^{\ell} \mathsf{rot}_{ki} \left(\sum_{j=0}^{k} \mathsf{rot}_{-ki}(\mathsf{diag}_{ki+j}(\boldsymbol{M})) \odot \mathsf{rot}_j(\boldsymbol{v}) \right)
\end{aligned}$$

The first line gives a simple way to compute homomorphic matrix-vector multiplication which requires $O(n)$ rotations and Hadamard multiplications. Based on the third line of the equation, we can achieve an algorithm so-called baby-step giant-step (BSGS) to matrix-vector multiplication with $O(\sqrt{n})$ rotations of ciphertexts.

3.2 Our method

Now we will introduce our method for fast homomorphic DFT. In this section, we will mainly consider DFT with bit-reversed output $\mathrm{DFT}_n^{\mathsf{NR}}$ and its inverse (the letter NR stands for *normal to reversal*). In addition, we will describe the method to extend our method to input bit-reversed case and its inverse in the last part of this section. We focus on the power-of-two dimension case while our method can be generalized to other power of prime dimensions, because power-of-two cases are appropriate to our applications and, moreover, easy to describe.

The starting point of our method is to observe that the multiplication between matrix and encrypted vector can be much faster when the matrix only has the small number of non-zero $\{\mathsf{diag}_i(\boldsymbol{M})\}_{0 \le i < n}$. Bit-reversed order DFT matrix can be decomposed to sparse matrices, and this property is used to fasten the discrete Fourier transform. Our observation is that those sparse matrices have small number of non-zero $\{\mathsf{diag}_i(\cdot)\}_{0 \le i < n}$ (exactly two or three non-zero vectors).

The DFT matrix factorization. Let $\mathbf{DFT}_n^{\mathsf{NR}}$ be a matrix coressponding to the DFT algorithm with input length n with bit-reversed output. The following

equation shows that the matrix representation of recursive FFT Cooley-Tukey algorithm [7].

$$\mathbf{DFT}_n^{\mathsf{NR}} = \begin{bmatrix} \mathbf{DFT}_{n/2}^{\mathsf{NR}} & \mathbf{DFT}_{n/2}^{\mathsf{NR}} \\ \mathbf{DFT}_{n/2}^{\mathsf{NR}} \cdot \boldsymbol{W}_{n/2} & -\mathbf{DFT}_{n/2}^{\mathsf{NR}} \cdot \boldsymbol{W}_{n/2} \end{bmatrix} = \begin{bmatrix} \mathbf{DFT}_{n/2}^{\mathsf{NR}} & \mathbf{0} \\ \mathbf{0} & \mathbf{DFT}_{n/2}^{\mathsf{NR}} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{I}_{n/2} & \boldsymbol{I}_{n/2} \\ \boldsymbol{W}_{n/2} & -\boldsymbol{W}_{n/2} \end{bmatrix}$$

where the matrix $\boldsymbol{W}_{n/2} = \mathsf{diag}(1, \omega_n, \omega_n^2, \cdots, \omega_n^{n/2-1})$ and $\omega_n = e^{2\pi i/n}$. If we adapt this equation repeatedly, we can decompose the DFT matrix $\mathbf{DFT}_n^{\mathsf{NR}}$ to $\log_2 n$ number of matrices. The following matrix illustrates the specific form of matrices in the recursive formula:

$$\boldsymbol{D}_k^{(n)} = \begin{bmatrix} \begin{bmatrix} \boldsymbol{I}_{n/k} & \boldsymbol{I}_{n/k} \\ \boldsymbol{W}_{n/k} & -\boldsymbol{W}_{n/k} \end{bmatrix} & 0 & \cdots & 0 \\ 0 & \begin{bmatrix} \boldsymbol{I}_{n/k} & \boldsymbol{I}_{n/k} \\ \boldsymbol{W}_{n/k} & -\boldsymbol{W}_{n/k} \end{bmatrix} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \begin{bmatrix} \boldsymbol{I}_{n/k} & \boldsymbol{I}_{n/k} \\ \boldsymbol{W}_{n/k} & -\boldsymbol{W}_{n/k} \end{bmatrix} \end{bmatrix} \in \mathbb{C}^{n \times n}. \quad (1)$$

which has $k/2$ number diagonal blocks. The recursive equation above implies

$$\mathbf{DFT}_n^{\mathsf{NR}} = \boldsymbol{D}_n^{(n)} \cdot \boldsymbol{D}_{n/2}^{(n)} \cdot \cdots \cdot \boldsymbol{D}_2^{(n)}. \quad (2)$$

Remark 1. As noted above, decomposing DFT matrices into sparse diagonal matrices is possible for other power-of-prime cases and this induces a fast homomorphic DFT algorithm for power-of-prime dimension. This fact can be obtained by using general Cooley-Tukey algorithm.

Homomorphic DFT. We recall the representation of matrix and vector multiplication via Hadamard multiplication and vector shifting:

$$\boldsymbol{M} \cdot \boldsymbol{v} = \sum_{i=0}^{n} \mathsf{diag}_i(\boldsymbol{M}) \odot \mathsf{rot}_i(\boldsymbol{v}).$$

The matrix-vector multiplication algorithm based on this form is especially efficient for the matrix $\boldsymbol{M}$ with only small number of non-zero diagonal vector. Namely, $\mathsf{diag}_i(\boldsymbol{M})$ is a non-zero vector only for small number of i's. We call this matrix by *sparse-diagonal* matrix. For sparse diagonal matrix $\boldsymbol{M}$, we don't need to compute $\mathsf{rot}_i(\boldsymbol{v})$ for those i's satisfying $\mathsf{diag}_i(\boldsymbol{M}) = \mathbf{0}$. Therefore, the required number of shifting in naive approach is at most the number of non-zero diagonal vectors that the matrix M has.

Lemma 1. *$\mathsf{diag}_k(\boldsymbol{D}_{2^i}^{(n)})$ is nonzero only for $k = 0, \pm n/2^i$.*

Proof. See Equantion (1). □

From Lemma 1, multiplication between matrix $\boldsymbol{D}_{2^i}^{(n)}$ and vector $\boldsymbol{v}$ can be represented as follows:

$$\boldsymbol{D}_{2^i}^{(n)} \cdot \boldsymbol{v} = \mathsf{diag}_0(\boldsymbol{D}_{2^i}^{(n)}) \odot \boldsymbol{v} + \mathsf{diag}_{n/2^i}(\boldsymbol{D}_{2^i}^{(n)}) \odot \mathsf{rot}_{n/2^i}(\boldsymbol{v}) + \mathsf{diag}_{n-n/2^i}(\boldsymbol{D}_{2^i}^{(n)}) \odot \mathsf{rot}_{-n/2^i}(\boldsymbol{v}).$$

Therefore, $\mathrm{DFT}_n^{\mathsf{NR}} \cdot \boldsymbol{v}$ can be computed recursively as $\prod_{i=1}^{\log_2 n} \boldsymbol{D}_{2^i}^{(n)} \cdot \boldsymbol{v}$, where each multiplication can be done with $O(1)$ number of Hadamard multiplication and shifting. The overall number of operations is $O(\log n)$ homomorphic shiftings and Hadamard multiplications with constant plaintext vectors. The Algorithm 1 shows our homomorphic DFT algorithm in detail with notations in Section 2.2.

Algorithm 1 Homomorphic $\mathrm{DFT}_n^{\mathsf{NR}}$ algorithm

Require: Ciphertext ctxt such that $\mathsf{Dec}(\mathsf{ctxt}, \mathsf{sk}) = \boldsymbol{m} \in \mathbb{C}^n$
  for all $1 \le i \le \log_2 n$ **do**
    $\mathsf{ctxt}_0 \leftarrow \mathsf{CMult}(\mathsf{diag}_0(\boldsymbol{D}_{2^i}^{(n)}), \mathsf{ctxt})$
    $\mathsf{ctxt}_1 \leftarrow \mathsf{LeftRotate}(\mathsf{ctxt}, n/2^i)$
    $\mathsf{ctxt}_2 \leftarrow \mathsf{RightRotate}(\mathsf{ctxt}, n/2^i)$
    $\mathsf{ctxt}_1 \leftarrow \mathsf{CMult}(\mathsf{diag}_{n/2^i}(\boldsymbol{D}_{2^i}^{(n)}), \mathsf{ctxt}_1)$
    $\mathsf{ctxt}_2 \leftarrow \mathsf{CMult}(\mathsf{diag}_{n-n/2^i}(\boldsymbol{D}_{2^i}^{(n)}), \mathsf{ctxt}_2)$
    $\mathsf{ctxt} \leftarrow \mathsf{Add}(\mathsf{ctxt}_0, \mathsf{ctxt}_1)$
    $\mathsf{ctxt} \leftarrow \mathsf{Add}(\mathsf{ctxt}, \mathsf{ctxt}_2)$
  end for

In each loop of the Algorithm 1, there are two homomorphic rotations and three homomorphic constant vector multiplications. Furthermore, left rotation by $n/2$ and right rotation by $n/2$ is same. For this reason, we do not need to compute right and left rotations for $i = 1$ case. This will reduce one homomorphic rotation. As a result, our algorithm needs $(2\log_2 n - 1)$ number of homomorphic rotation and $(3\log_2 n)$ number of homomorphic constant vector multiplications.

Trade-off between depth and complexity. While our method is fairly efficient with respect to the number of operations, the required depth with respect to constant multiplication is also increased by $O(\log_2 n)$. In this respect, we adapt additional parameter r which is called *radix* to generalize our method. Our generalized method gives trade-off between the number of steps and complexity of homomorphic DFT.

Assume that $\log_2 n$ is even and recall the matrix decomposition of DFT matrix:

$$\begin{aligned}\mathbf{DFT}_n^{\mathsf{NR}} &= \boldsymbol{D}_n^{(n)} \cdot \boldsymbol{D}_{n/2}^{(n)} \cdot \boldsymbol{D}_{n/4}^{(n)} \cdot \boldsymbol{D}_{n/8}^{(n)} \cdot \cdots \cdot \boldsymbol{D}_{16}^{(n)} \cdot \boldsymbol{D}_8^{(n)} \cdot \boldsymbol{D}_4^{(n)} \cdot \boldsymbol{D}_2^{(n)} \\ &= (\boldsymbol{D}_n^{(n)} \cdot \boldsymbol{D}_{n/2}^{(n)}) \cdot (\boldsymbol{D}_{n/4}^{(n)} \cdot \boldsymbol{D}_{n/8}^{(n)}) \cdots (\boldsymbol{D}_4^{(n)} \cdot \boldsymbol{D}_2^{(n)}).\end{aligned}$$

This equation give a factorization of DFT matrix into $\log_4 n$ number of matrices of the form

$$\boldsymbol{D}_i^{(n;4)} = (\boldsymbol{D}_{2^{2i}}^{(n)} \cdot \boldsymbol{D}_{2^{2i-1}}^{(n)}) \text{ for } 1 \leq i \leq \log_4 n.$$

We also can define similar term for $r = 2^k$ by

$$\boldsymbol{D}_j^{(n;r)} = \boldsymbol{D}_{r^j}^{(n)} \cdot \boldsymbol{D}_{r^j/2}^{(n)} \cdot \cdots \cdot \boldsymbol{D}_{r^{j-1}\cdot 2}^{(n)}$$

for $1 \leq j \leq \log_r n$ and $k | \log_2 n$. This factorization allows us to compute DFT in a new way. To analyze the efficiency, we observe some properties of these matrices.

Lemma 2. *The multiplication of i-th diagonal matrix and j-th diagonal matrix is $i + j$-th diagonal matrix. More precisely, the following equation holds*

$$\mathsf{diag}_i(\boldsymbol{a}) \cdot \mathsf{diag}_j(\boldsymbol{b}) = \mathsf{diag}_{i+j}(\boldsymbol{a} \odot \mathsf{rot}_i(\boldsymbol{b})).$$

Proof. Trivial.

Lemma 3. *Let D_k be a multiplication of k consecutive matrices in Equation 2:*

$$\boldsymbol{D}_k = \boldsymbol{D}_{2^{s+k}}^{(n)} \cdot \boldsymbol{D}_{2^{s+k-1}}^{(n)} \cdot \cdots \cdot \boldsymbol{D}_{2^{s+1}}^{(n)}.$$

Then at most $2^{k+1} - 1$ diagonals of D is nonzero vector. Further, the indices of nonzero diagonals form arithmetic progression.

Proof. Lemma 2 clearly holds. To show Lemma 3, we decompose $D_{2^t}^{(n)}$ into $\mathsf{diag}_{-n/2^t}(\boldsymbol{D}_{2^t}^{(n)}) + \mathsf{diag}_0(\boldsymbol{D}_{2^t}^{(n)}) + \mathsf{diag}_{n/2^t}(\boldsymbol{D}_{2^t}^{(n)})$ as in Lemma 1. By Lemma 2, the index of $\boldsymbol{D}_k$ that is non-zero is of the form

$$e_{s+1} \cdot \frac{n}{2^{s+1}} + e_{s+2} \cdot \frac{n}{2^{s+2}} + \cdots + e_{s+t} \cdot \frac{n}{2^{s+t}},$$

where $e_i \in \{-1, 0, 1\}$ for $s + 1 \leq i \leq s + t$. These indices are multiple of $n/2^{s+t}$, and the absolute value of it is bounded by $\sum_{j=s+1}^{s+t} n/2^j = (2^t - 1)n/2^{s+t}$. □

According to Lemma 3, the number of nonzero diagonal of $\boldsymbol{D}_j^{(n;r)}$ is $2r - 1$ for $j > 1$ and r for $j = 1$. Thus the required number of homomorphic multiplication and slot shifting to compute multiplication of encryption of $\boldsymbol{v}$ and $\boldsymbol{D}_j^{(n;r)}$ is less than $2r - 1 = O(r)$ for radix r, respectively. By recursively multiplying $\boldsymbol{D}_j^{(n;r)}$ to $\boldsymbol{v}$, we obtain a new algorithm to compute homomorphic DFT which requires $O(r \log_r n)$ homomorphic rotations and constant vector multiplications while has $O(\log_r n)$ depth. Overall, we obtain depth-efficiency trade-off using larger radix. We note that we assumed that the used radix is a divisor of $\log_2 n$, but this condition can be removed by considering dynamic radices for each recursive step.

3.3 Hybrid method

An interesting observation in Lemma 3 is that the indices of $\boldsymbol{D}_j^{(n;r)}$ forms an arithmetic progression. We call this property *regular*. Here we show that this property yield a hybrid method of our homomorphic DFT algorithm and baby-step giant-step (BSGS) algorithm. To do this, we apply a BSGS matrix-vector multiplication method for sparse diagonal matrix $\boldsymbol{M}$ with arithmetic progression indices as follows:

$$\begin{aligned}\sum_{i=1}^{t} \boldsymbol{m}_i \odot \mathsf{rot}_{\ell i}(v) &= \sum_{i=0}^{k_1-1}\sum_{j=1}^{k_2} \boldsymbol{m}_{ik_2+j} \odot \mathsf{rot}_{\ell\cdot(ik_2+j)}(\boldsymbol{v}) \\ &= \sum_{i=0}^{k_1-1} \mathsf{rot}_{lk_2i}\left(\sum_{j=1}^{k_2} \mathsf{rot}_{-lb_2i}(\boldsymbol{m}_{ik_2+j}) \odot \mathsf{rot}_{\ell j}(\boldsymbol{v})\right)\end{aligned}$$

where $\boldsymbol{m}_i = \mathsf{diag}_{\ell i}(\boldsymbol{M})$ and $k_1k_2 = t$.

In this BSGS method we can obtain a matrix multiplication $\boldsymbol{M}\cdot\boldsymbol{v}$ by $O(k_1 + k_2)$ rotations and $O(t)$ constant multiplications. We remark that we can vary the choice of k_1 and k_2 by increasing t and add zero diagonals. For this reason, we can say that the hybrid method needs $O(\sqrt{t})$ homomorphic rotations and $O(t)$ number of homomorphic constant vector multiplications. The Table 1 shows comparison our methods with other techniques.

Remark 2. Another advantage of our method is that it highly reduces the size of public key for operations. While the previous BSGS method requires $O(\sqrt{n})$ rotation key, our method only needs $O(r\log_r n)$ number of rotation key.

	Naive	BSGS	Ours with radix r	Hybrid with radix r
# Hadamard Mult	$O(n)$	$O(n)$	$O(r\log_r n)$	$O(r\log_r n)$
# Slot Shifting	$O(n)$	$O(\sqrt{n})$	$O(r\log_r n)$	$O(\sqrt{r}\log_r n)$
Depth	1	1	$O(\log_r n)$	$O(\log_r n)$

Table 1. Comparison: homomorphic operation number and depth consume for homomorphic DFT

3.4 Extension to Inverse and Different Orders

Homomorphic Inverse DFT. Now we describe the computation of inverse DFT (iDFT) in homomorphic way. Note that our homomorphic DFT algorithm computes the bit-reversal DFT values, so we should compute iDFT from the bit-reversal order input to regular order output. The matrix representation of iDFT with bit-reversal order is

$$\begin{aligned}\mathbf{iDFT}_n^{\mathsf{RN}} &:= \left(\mathbf{DFT}_r^{\mathsf{NR}}\right)^{-1} = \left(\boldsymbol{D}_n^{(n)}\cdot\boldsymbol{D}_{n/2}^{(n)}\cdot\cdots\cdot\boldsymbol{D}_2^{(n)}\right)^{-1} \\ &= \left(\boldsymbol{D}_2^{(n)}\right)^{-1}\cdot\left(\boldsymbol{D}_4^{(n)}\right)^{-1}\cdot\cdots\cdot\left(\boldsymbol{D}_n^{(n)}\right)^{-1}.\end{aligned}$$

These matrices are decomposed into the certain form of matrices with useful property as follows:

$$\begin{bmatrix} \boldsymbol{I}_{n/k} & \boldsymbol{I}_{n/k} \\ \boldsymbol{W}_{n/k} & -\boldsymbol{W}_{n/k} \end{bmatrix}^{-1} = \frac{1}{2} \begin{bmatrix} \boldsymbol{I}_{n/k} & \overline{\boldsymbol{W}_{n/k}} \\ \boldsymbol{I}_{n/k} & -\overline{\boldsymbol{W}_{n/k}} \end{bmatrix}.$$

In other words, for the divisor 2^i of n, the equation

$$\left(\boldsymbol{D}_{2^i}^{(n)}\right)^{-1} = \frac{1}{2}\overline{\boldsymbol{D}_{2^i}^{(n)}}^T$$

holds. This equation implies that $\mathsf{diag}_k((\boldsymbol{D}_{2^k}^{(n)})^{-1})$ is nonzero only for two or three k's as in Lemma 1. Therefore $\mathbf{iDFT}_n^{\mathsf{RN}}$ matrix can also be decomposed into sparse diagonal matrices as $\mathbf{DFT}_n^{\mathsf{NR}}$, and it induces the fast homomorphic iDFT algorithm. We remark that we also obtain a useful equation

$$\mathbf{iDFT}_n^{\mathsf{RN}} = \frac{1}{n}\overline{\mathbf{DFT}_n^{\mathsf{NR}}}^T.$$

$\mathbf{DFT}_n^{\mathsf{RN}}$ and $\mathbf{iDFT}_n^{\mathsf{NR}}$. Let $\boldsymbol{R}$ is the bit-reversal permutation matrix. We know that $\mathbf{DFT}_n^{\mathsf{RN}} = \boldsymbol{R}\cdot\mathbf{DFT}_n^{\mathsf{NR}}\cdot\boldsymbol{R}$ and $\boldsymbol{R}^2 = I$. If we use $\tilde{\boldsymbol{D}}_{2^i}^{(n)} = \boldsymbol{R}\cdot\boldsymbol{D}_{2^i}^{(n)}\cdot\boldsymbol{R}$ instead of $\boldsymbol{D}_{2^i}^{(n)}$, we can get an matrix factorization of $\mathbf{DFT}_n^{\mathsf{RN}}$:

$$\mathbf{DFT}_n^{\mathsf{RN}} = \tilde{\boldsymbol{D}}_n^{(n)}\cdot\tilde{\boldsymbol{D}}_{n/2}^{(n)}\cdot\tilde{\boldsymbol{D}}_{n/4}^{(n)}\cdot\cdots\cdot\tilde{\boldsymbol{D}}_{16}^{(n)}\cdot\tilde{\boldsymbol{D}}_8^{(n)}\cdot\tilde{\boldsymbol{D}}_4^{(n)}\cdot\tilde{\boldsymbol{D}}_2^{(n)}.$$

In addition, we can use same technique as Section 3.4 to obtain matrix factorization of $\mathbf{iDFT}_n^{\mathsf{RN}}$. For that, we prove that this matrices $\tilde{\boldsymbol{D}}_{2^i}^{(n)}$ has same structure with $\boldsymbol{D}_{n/2^i}^{(n)}$.

Lemma 4. $\mathsf{diag}_k\left(\boldsymbol{R}\cdot\boldsymbol{D}_{2^i}^{(n)}\cdot\boldsymbol{R}\right)$ *is nonzero only for* $k = 0, \pm 2^i$.

Proof. First, we know that (i,j)-th element of $\boldsymbol{R}\cdot\boldsymbol{M}\cdot\boldsymbol{R}$ is a n by n matrix is $\left(m_{\mathsf{rev}(i),\mathsf{rev}(j)}\right)$. And, Following equation hold:

$$\mathsf{rev}(i+2^k) = \mathsf{rev}(i) + \mathsf{rev}(2^k) \;\text{ if } k\text{-th bit of } i \text{ is } 0.$$

The equation 1 shows that the nonzero term of $\boldsymbol{D}_{2^i}^{(n)}$ is in (k,k) or $(k, k\pm n/2^i)$,and elements with index $(k, k\pm n/2^i)$ for $\lfloor k/(n/2^i)\rfloor = 0 \bmod 2$ are zero. This means that we only need to consider elements with index (k,k) and $(k', k'\pm n/2^i)$ for $(\log_2 n - i)$-th bit of k' is 0. For all k' that satisfies the condition, $R(k'\pm n/2^i) = R(k') \pm R(n/2^i) = R(k') \pm 2^i$.

Above Lemma shows that $\tilde{\boldsymbol{D}}_{2^i}^{(n)}$ also has three diagonals $\mathsf{diag}_k(\tilde{\boldsymbol{D}}_{2^i}^{(n)})$ for $k = 0, \pm 2^i$. For this reason, our method can be applied to those linear transformations.

3.5 Implementation

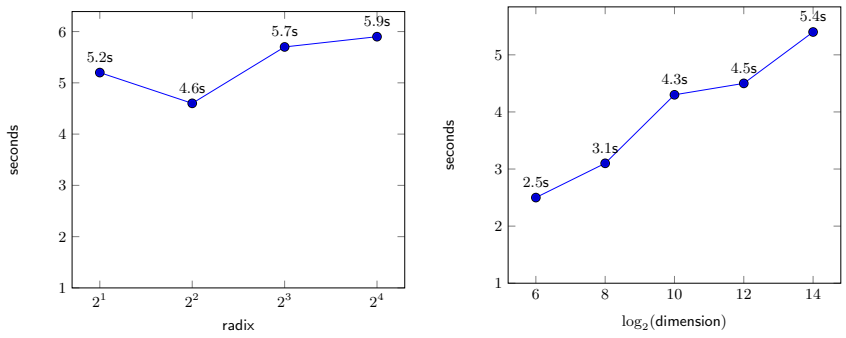
We implemented our DFT algorithm using HEAAN library [13]. HEAAN library supports batch encodings, or encoding for vectors, for complex plaintext space thus it is suitable for our target; discret Fourier transform. All of experiments in this paper are done at the PC having 32 number of Intel(R) Xeon(R) CPU E5-2620 v4 2.10 GHz CPU (each CPU has 8 cores) and 64GB RAM. We used multi-threding with 8 number of threads.

The following HEAAN parameter setting is what we used in the experiment for our homomorphic DFT algorithm.

- $q_L = 2^{440}$: the largest ciphertext modulus.
- $N = 2^{15}$: the dimension of polynomial ring $\mathcal{R}$.
- $\Delta = p^k = 2^{30}$: scaling factor which is used to make integer polynomial in encryption and constant vector multiplication both.
- $\sigma = 3.2$, $\rho = 0.5$, and $h = 64$: distribution related parameters.

Note that the expected security of this parameter setting is about 128 bit following the `LWEestimator` [1].

The Figure 1 shows timing results for various setting. In case of the first one, radix varies from 2 to 16 with the fixed dimension of input vector 2^{12}. In case of the second one, dimension varies from 2^6 to 2^{12} with the fixed radix 4.



(a) Timming results for various radix setting with dimension $n = 2^{12}$

(b) Timming results for various dimension setting with radix $r = 4$

Fig. 1. Implementation results for our homomorphic DFT algorithm

By the effect of baby-step giant-step method, the left one of the Figure 1 shows that timing does not increase a lot when we increase the radix. And, the right figure shows that timing increase linearly to logarithm of the dimension n. Therefore, we can get a homomorphic DFT algorithm which is significantly faster and similar depth consume. In our experiment, we compare the result with DFT on un-encrypted vector. We use average of $|a_i - b_i|$ for all $0 \leq i < n$ as

difference between two length n vector $\boldsymbol{a}$ and $\boldsymbol{b}$. The difference between DFT on encrypted and un-encrypted state in our experiment is 2^{-9} to 2^{-10}. We can reduce this difference by using larger $\Delta = p^k$.

There are a few previous implementation results about homomorphic DFT. In [8], there homomorphic DFT takes about 22 minutes for $n = 2^{13}$ with 8-bit precision. In [6], it takes about 22 minutes with same length. But these works focus on amortized time by put each element of the input vector in different ciphertext. We note that our results shows about 200 times faster than previous one.

4 Improved Bootstrapping for approximate HE

In this section, we explain about linear transformations in bootstrapping for approximate homomorphic encryption scheme. And, we give an improved transformation algorithms for such linear transform using our homomorphic DFT which provides an improved bootstrapping procedure for approximate homomorphic encryption.

4.1 Linear Transformation in Bootstrapping

The bootstrapping procedure for approximate homomorphic encryption in [5] can be divided as following steps:

1. Put polynomial coefficients in plaintext slots,
2. Evaluate exponent function,
3. Extract Imaginary part,
4, Switch back to the coefficient representation.

The transformations in the first and the last step are called CoeffToSlot and SlotToCoeff respectively. In [5], the authors use the index i of slots corresponding to $5^k \pmod{2N}$ for $0 \leq k \leq N/2$ by considering $w_{2N}^{5^k}$ as in Encode map. To transform coefficients of polynomial representation of plaintext into slots, we should construct two encodings since there are only $N/2$ slots while the number of coefficients is N.

Let $t(x) = t_0 + t_1 x + \cdots t_{N-1} x^{N-1}$ be a polynomial representation of encoding with messages $\boldsymbol{z} = (z_0, \cdots, z_{N/2-1})$ in slots, and let $\boldsymbol{v} = (t_0, \cdots, t_{N-1}) = (\boldsymbol{v}_0, \boldsymbol{v}_1)$ be its vector representation. Suppose that $\boldsymbol{U}$ be the encoding matrix defined in Section 2.2 and parsed into $[\boldsymbol{U}_0 | \boldsymbol{U}_1]$ for $N/2$ by $N/2$ matrices $\boldsymbol{U}_k$'s. Then the following equation holds by definition of encoding map, which yields the SlotToCoeff map,

$$\boldsymbol{z} = \boldsymbol{U} \cdot \begin{bmatrix} \boldsymbol{v}_0 \\ \boldsymbol{v}_1 \end{bmatrix} = \boldsymbol{U}_0 \cdot \boldsymbol{v}_0 + \boldsymbol{U}_1 \cdot \boldsymbol{v}_1.$$

Note that $i \cdot \boldsymbol{U}_0 = \boldsymbol{U}_1$ and $\boldsymbol{U}_0^{-1} = \frac{2}{N} \cdot \overline{\boldsymbol{U}_0}^T$ hold. Using this, we can obtain that

$$\boldsymbol{v}_k = \frac{1}{N} \left(\overline{\boldsymbol{U}_k}^T \cdot \boldsymbol{z} + \boldsymbol{U}_k^T \cdot \overline{\boldsymbol{z}} \right) \text{ for } k = 0, 1.$$

This equation corresponds to CoeffToSlot map.

4.2 Improved Linear Transformation in Bootstrapping

We now describe a modified linear transforms for bootstrapping. We mainly focus on how to decompose the matrix $\boldsymbol{U}$ into sparse diagonal matrices. To obtain this, the bit-reversal permutation matrix $\boldsymbol{R}$ works a central role in this method. Note that the order of the slots after $\mathsf{CoeffToSlot}$ does not play any role in the bootstrapping. For this reason, we replace $\boldsymbol{U}_k$ to $\boldsymbol{V}_k$ which is row permuted by $\boldsymbol{R}$:

$$\boldsymbol{V}_k = \boldsymbol{U}_k \cdot \boldsymbol{R} \;\; \textit{for } k = 0, 1.$$

As in $\boldsymbol{U}$, the relation $\boldsymbol{V}_1 = i \cdot \boldsymbol{V}_0$ holds. For this reason, we focus on the matrix decomposition of $\boldsymbol{V}_0$ using recursive relation; this induces the decomposition of $\boldsymbol{V}_1$. Let $\mathsf{rev}_n(i)$ denotes bit-reversal permutation of i with size n.

Lemma 5. *Let* $\boldsymbol{S}_n = \left(\omega_{4n}^{5^i \cdot \mathsf{rev}_n(j)}\right)_{0 \le i,j < n}$. *Then,* $\boldsymbol{V}_0 = \boldsymbol{S}_{N/2}$ *and following equation holds:*

$$\boldsymbol{S}_n = \begin{bmatrix} I & \boldsymbol{W}_n \\ I & -\boldsymbol{W}_n \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{S}_{n/2} & 0 \\ 0 & \boldsymbol{S}_{n/2} \end{bmatrix}$$

for $\boldsymbol{W}_n = \mathsf{diag}(\omega_{4n}^{5^i})_{0 \le i < n}$.

Proof. $\boldsymbol{V}_0 = \boldsymbol{S}_{N/2}$ is clear by definition. Let's start the proof with the following claim. Here $v_2(a)$ is the maximal integer k such that 2^k is a divisor of integer a.

Claim. $v_2(5^e - 1) = v_2(e) + 2$ holds for a positive integer e.

Proof: This claim can be proven using the mathematical induction on $v_2(e)$. ■

To prove the recursive formula, it suffices to show the following equation:

$$\boldsymbol{S}_n = \begin{bmatrix} \boldsymbol{S}_{n/2} & \boldsymbol{W}_n \cdot \boldsymbol{S}_{n/2} \\ \boldsymbol{S}_{n/2} & -\boldsymbol{W}_n \cdot \boldsymbol{S}_{n/2}. \end{bmatrix}$$

Let $\boldsymbol{S}_n = (s_{i,j})_{0 \le i,j < n}$, i.e. $s_{i,j} = \omega_{4n}^{5^i \cdot \mathsf{rev}_n(j)}$. The following equations show the above equation. Note that $4n$ is a power of two integer.

1. $s_{i,j} = s_{i+n/2,j}$ for all i and for $0 \le j < n/2$: this is equivalent to $4n | (5^{(i+n/2)} \cdot \mathsf{rev}_n(j) - 5^i \cdot \mathsf{rev}_n(j))$. By the claim $v_2(5^{n/2} - 1) = v_2(n/2) + 2 = v_2(2n)$ holds and $\mathsf{rev}_n(j)$ is even for $j < n/2$. Combining this we obtain the desired result is induced.
2. $s_{i,j} = -s_{i+N/2,j}$ for all i and for $n/2 \le j < n$: as in the above, it is equivalent to $v_2(5^{(i+n/2)} \cdot \mathsf{rev}_n(j) - 5^i \cdot \mathsf{rev}_n(j)) = v_2(2n)$. It is showed by $v_2(5^{n/2} - 1) = v_2(n/2) + 2 = v_2(2n)$ and $\mathsf{rev}_n(j)$ is odd for $j \ge n/2$.
3. $s_{i,j+N/2} = s_{i,j} \cdot \omega_{4n}^{5^i}$ for all i and $0 \le j < n/2$: this is clear by definition of rev_n.

If we combine these cases, we can easily show that the recursive relation of $\boldsymbol{S}$ holds. □

By adapting Lemma 5 repeatedly, we can decompose $\boldsymbol{V}_0$ to $\log_2 n$ number of matrices as in Equation 2. The following matrix illustrates the specific form of matrices in the recursive formula:

$$\boldsymbol{E}_k^{(n)} = \begin{bmatrix} \begin{bmatrix} \boldsymbol{I}_{n/k} & \boldsymbol{W}_{n/k} \\ \boldsymbol{I}_{n/k} & -\boldsymbol{W}_{n/k} \end{bmatrix} & 0 & \cdots & 0 \\ 0 & \begin{bmatrix} \boldsymbol{I}_{n/k} & \boldsymbol{W}_{n/k} \\ \boldsymbol{I}_{n/k} & -\boldsymbol{W}_{n/k} \end{bmatrix} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \begin{bmatrix} \boldsymbol{I}_{n/k} & \boldsymbol{W}_{n/k} \\ \boldsymbol{I}_{n/k} & -\boldsymbol{W}_{n/k} \end{bmatrix} \end{bmatrix} \in \mathbb{C}^{n \times n}. \quad (3)$$

which has $k/2$ number diagonal blocks. Lemma 5 implies

$$\boldsymbol{V}_0 = \boldsymbol{E}_2^{(N/2)} \cdot \boldsymbol{E}_4^{(N/2)} \cdot \boldsymbol{E}_8^{(N/2)} \cdots \boldsymbol{E}_{N/2}^{(N/2)}.$$

These factor matrices have exactly the same structure with $\boldsymbol{D}_k^{(n)}$, so we can apply our method in previous section (from radix to hybrid method). Furthermore, we can also multiply the inverse of $\boldsymbol{V}_0$ in encrypted state, as in the same way to the inverse DFT matrix case.

Now we will describe two linear transformations, CoeffToSlot and SlotToCoeff, using $\boldsymbol{V}_0$, $\boldsymbol{V}_0^{-1}$ and its conjugations. As we noted above, $\boldsymbol{V}_1 = i \cdot \boldsymbol{V}_0$ and further $\boldsymbol{V}_k^{-1} = \frac{2}{N}\overline{\boldsymbol{V}_k}^T$ hold as in the case of $\boldsymbol{U}$ for $k = 0, 1$. Therefore, CoeffToSlot with bit-reversed result and SlotToCoeff with bit-reversed input are computed as follows for $\boldsymbol{t}_k = \boldsymbol{R} \cdot v_k$ for $k = 0, 1$:

$$\boldsymbol{t}_0 = \frac{1}{2}\left(\boldsymbol{V}_0^{-1} \cdot \boldsymbol{z} + \overline{\boldsymbol{V}_0^{-1} \cdot \boldsymbol{z}}\right), \quad \boldsymbol{t}_1 = -\frac{1}{2} i \left(\boldsymbol{V}_0^{-1} \cdot \boldsymbol{z} - \overline{\boldsymbol{V}_0^{-1} \cdot \boldsymbol{z}}\right),$$

$$\boldsymbol{z} = \boldsymbol{V}_0 \cdot (\boldsymbol{t}_0 + i \cdot \boldsymbol{t}_1).$$

Optimization. We can further improve the efficiency of the bootstrapping in light of *hoisting*, i.e. by computing the common part first or last. More precisely, for CoeffToSlot, compute $\boldsymbol{V}_0^{-1} \cdot \boldsymbol{z}$ first and compute other parts using conjugation. Therefore, $\boldsymbol{t}_0$ and $\boldsymbol{t}_1$ can be computed from $\boldsymbol{z}$ in $2\sqrt{r}\log_r(N/2)$ homomorphic operations for the radix r. For SlotToCoeff, we compute $(\boldsymbol{t}_0 + i \cdot \boldsymbol{t}_1)$ first and multiply $\boldsymbol{V}_0$. This also needs only $2\sqrt{r}\log_r(N/2)$ number of homomorphic operations.

Remark 3. Our technique can be applied for bootstrapping of $(n/2)$-sparsely packed ciphertext in [5]. The plaintext space of sparse packed ciphertext is $\mathbb{Z}[Y]/(Y^n + 1)$ for $Y = X^{N/n}$. So, we just need to replace ω_{2N} to ω_{2n}.

4.3 Implementation

Use one of the parameter sets which is in the previous work [5] for easier comparison. And, we run the previous method which is implemented in HEAAN library [13] in the same machine for fare comparison (with recently release version v2.1). The PC information is same as the previous implementation in Section 3.5.

- $q_0 = 2^{41}$: the smallest ciphertext modulus (before bootstrapping).
- $q_L = 2^{1240}$: the largest ciphertext modulus.
- $N = 2^{16}$: the dimension of polynomial ring $\mathcal{R}$.
- $\Delta = p^k = 2^{31}$: scaling factor which is used to make integer polynomial in encryption and constant vector multiplication both.
- $\sigma = 3.2$, $\rho = 0.5$, and $h = 64$: distribution related parameters.
- $r = 7$ which is the number of iteration in sin evaluation.

The Table 2 shows implementation result of bootstrapping using our linear transformation and previous method. To maximize the effect of our method, we used number of slots as the largest one ($= N/2$).

	Key Gen	Linear Trans	Eval sin	**Total**	Amortized Time
Previous	25 hours	26 hours	30 sec	26 hours	5.71 sec
Ours	**44 sec**	**97 sec**	30 sec	127 sec	3.8 ms

Table 2. Timing of Bootstrapping with comparison for $\mathbb{C}^{32768}$ plaintext space. Here amortized time means that bootstrapping time per one complex element. Both works gives about 2^{-7} additive error while bootstrapping.

The timing results for linear transformation time shows about 700 times faster result than previous one. We use radix 32 which means each linear transformation consumes 3 ($= \log_{32} 2^{15}$) constant vector multiplication depth. As a result, the modulus of the return ciphertext is 468 bits which means 14 depth computation can be done after bootstrapping. In the previous method, the modulus of the return ciphertext is 632 bits which means 19 depth computation can be done after bootstrapping.

Another advantage of our method is key generation time. Key generation includes public key generation for various rotations and pre-encodings for diagonal vectors. In the previous method, they need to encode for $N/2(= 32768)$ number of constant vectors for each linear transformation. The number of rotation key is $2\sqrt{N/2}$ which is quite large compare to $2\sqrt{k}\log_k N/2$ in our case. In the experiment, this problem makes their key generation time slower and the size of pre-encoded vector and public keys to be huge. Previous method need 800GB to save them and 7GB for ours.

References

1. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.
2. Tiziano Bianchi, Alessandro Piva, and Mauro Barni. Comparison of different fft implementations in the encrypted domain. In *Signal Processing Conference, 2008 16th European*, pages 1–5. IEEE, 2008.
3. Tiziano Bianchi, Alessandro Piva, and Mauro Barni. On the implementation of the discrete fourier transform in the encrypted domain. *IEEE Transactions on Information Forensics and Security*, 4(1):86–97, 2009.

4. Hao Chen and Kyoohyung Han. Homomorphic lower digits removal and improved fhe bootstrapping. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 315–337. Springer, 2018.
5. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 360–384. Springer, 2018.
6. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.
7. James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.
8. Anamaria Costache, Nigel P Smart, and Srinivas Vivek. Faster homomorphic evaluation of discrete fourier transforms. In *International Conference on Financial Cryptography and Data Security*, pages 517–529. Springer, 2017.
9. Anamaria Costache, Nigel P Smart, Srinivas Vivek, and Adrian Waller. Fixed-point arithmetic in she schemes. In *International Conference on Selected Areas in Cryptography*, pages 401–422. Springer, 2016.
10. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.
11. Shai Halevi and Victor Shoup. Bootstrapping for HElib. In *Advances in Cryptology–EUROCRYPT 2015*, pages 641–670. Springer, 2015.
12. Shai Halevi and Victor Shoup. Faster homomorphic linear transformations in helib. Technical report, Cryptology ePrint Archive, Report 2018/244, 2018.
13. Andrey Kim. HEAAN. `https://github.com/kimandrik/HEAAN`, 2018.