Better Bootstrapping for Approximate Homomorphic Encryption

Kyoohyung Han[1]* and Dohyeong Ki[2]

[1] Coinplug Inc, Republic of Korea
kyoohyunghan@coinplug.com
https://kyoohyunghan.github.io
[2] Seoul National University, Republic of Korea
wooki7098@snu.ac.kr

Abstract. After Cheon et al. (Asiacrypt' 17) proposed an approximate homomorphic encryption scheme, HEAAN, for operations between encrypted real (or complex) numbers, the scheme is widely used in a variety of fields with needs on privacy-preserving in data analysis. After that, a bootstrapping method for HEAAN is proposed by Cheon et al. (Eurocrypt' 18) with modulus reduction being replaced by a sine function. In this paper, we generalize the Full-RNS variant of HEAAN proposed by Cheon et al. (SAC, 19) to reduce the number of temporary moduli used in key-switching. As a result, our scheme can support more depth computations without bootstrapping while ensuring the same level of security.
We also propose a new polynomial approximation method to evaluate a sine function in an encrypted state, which is specialized for the bootstrapping for HEAAN. Our method considers a ratio between the size of a plaintext and the size of a ciphertext modulus. Consequently, it requires a smaller number of non-scalar multiplications, which is about half of the Chebyshev method.
With our variant of the Full-RNS scheme and a new sine evaluation method, we firstly implement bootstrapping for a Full-RNS variant of approximate homomorphic encryption scheme. Our method enables bootstrapping for a plaintext in the space $\mathbb{C}^{16384}$ to be completed in 52 seconds while preserving 11 bit precision of each slot.

Keywords: Homomorphic Encryption · Bootstrapping.

1 Introduction

After the Gentry's first blueprint for a fully homomorphic encryption scheme [9], homomorphic encryption is regarded as one of the most important tools for privacy-preserving. In various applications that need privacy-protection, homomorphic operations between encrypted real number data are necessary. In 2017, Cheon et al. proposed a new homomorphic encryption scheme for efficient

* This work was done when the first author was in Seoul National University (SNU).

operations between real number data, which is called HEAAN [8]. Through homomorphic operations on encrypted real numbers, a lot of methodologies, such as logistic regression and Genome-side association study (GWAS), can be done in encrypted states [4, 14, 11, 13, 16, 18, 17].

Recently, a lot of data analysis tools and methods come out into the world, and they become more and more complicated. For examples, machine-learning algorithms such as convolutional neural network (CNN) and deep neural network (DNN) are extremely complicated. Therefore, modern data analysis algorithms require faster homomorphic operations and huge depth, which makes them hard to display their all ability in encrypted states without bootstrapping, since only a limited number of levels is provided by homomorphic encryption schemes. For example, since *nGraph-HE* [3], a deep-learning prediction on encrypted data, does not use bootstrapping, it only supports a limited number of layers. Also, in the case of logistic regression, solutions without bootstrapping [16, 18] support only a small number of iterations. For these reasons, importance of efficient homomorphic operations and bootstrapping become greater nowadays.

To solve these problems, a full-RNS variant of HEAAN (HEAAN-RNS) is proposed by Cheon et al. [6]. The key idea of their work comes from the fast base conversion in the full-RNS variant of the FV scheme [2]. By using the conversion, they can expand and reduce basis without going through Chinese Remainder Theorem (CRT) composition process. Since it does not need to use big integer arithmetic, overall speed of its homomorphic operations becomes 4 to 10 times faster than that of the original HEAAN scheme. After that, SEAL includes an implementation of this scheme in version 3.0. To avoid using temporary moduli, they combined the bit-decomposition technique and the RNS-decomposition technique[3]. Recently, an improved method for the fast base conversion is introduced in [11], which requires floating point operations to predict quotient parts.

In the case of the bootstrapping for HEAAN, the first method is proposed by Cheon et al. [7] with modulus reduction being replaced by a sine function. More precisely, they approximate the modulo q operation by the function $\frac{q}{2\pi}\sin(\frac{2\pi x}{q})$. Since HEAAN is an approximate homomorphic encryption scheme, additive noise in bootstrapping, which is not that big, is acceptable. To evaluate the function $\frac{q}{2\pi}\sin(\frac{2\pi x}{q})$ efficiently, they apply the Taylor approximation method to the function in a small range and use double-angle formula. This method requires a small number of homomorphic multiplications, but it needs the degree of an approximate polynomial to be large ($= O(\log Kq)$), when $x/q \in (-K, K)$. After that, improved methods for linear transformation in the bootstrapping are suggested in [5, 12], and another method for sine evaluation via the Chebyshev polynomial approximation is proposed in [5]. By using the Chebyshev polynomial approximation, they reduce the degree of an approximate polynomial a lot without increasing the number of homomorphic multiplications much.

In this paper, to make the bootstrapping more practical, we generalize the Full-RNS variant of HEAAN and improve a sine evaluation method for the boot-

[3] After version 3.2, they use one temporary modulus instead of bit-decomposition as in [17].

strapping. Moreover, we implement our generalized `HEAAN-RNS` scheme and it's bootstrapping using a newly computed approximate polynomial which is optimized for the bootstrapping.

1.1 Our Contribution

- We suggest a generalized key-switching method for the Full-RNS variant of `HEAAN`. We combine the RNS-decomposition method in [2] and the temporary modulus technique in [10]. Compared to the `HEAAN-RNS` scheme, we use a smaller number of temporary moduli while consuming lower complexity. As a result, our scheme requires about half complexity for homomorphic multiplication even with a larger security parameter.
- We propose a method which considers the size of a message in sine evaluation. More precisely, we evaluate a sine function by considering a ratio between the size of a message and the size of a ciphertext modulus. As a result, our method only requires $\mathsf{Max}(\log K + 3 + \frac{1}{K}(\log \epsilon - 1), \log\log q)$ levels, where ϵ is a ratio between the size of a message and the size of a ciphertext modulus. Furthermore, by using cosine instead of sine, we combine double-angle formula for cosine with our approximation method. As a result, the number of non-scalar multiplications is almost reduced by half compared to the previous work [5].
- We put every technique together and implement the bootstrapping for our Full-RNS variant of `HEAAN`. As a result, our bootstrapping only takes 52 seconds for a plaintext in the space $\mathbb{C}^{16384}$ while preserving 11 bit precision of each slot.

1.2 Road Map

In Section 2, we briefly introduce the Chebyshev approximation, the `HEAAN-RNS` scheme with its fast base conversion, and the bootstrapping for `HEAAN`. In Section 3, we discuss a generalized key-switching method for the Full-RNS variant of `HEAAN`. In Section 4, we propose a better way of approximating a sine function for the bootstrapping. In Section 5, we implement our Full-RNS variant of `HEAAN` and its bootstrapping, and analyze results of our experiments. We complete the paper with a suggestion of future works for improving the bootstrapping.

2 Preliminary

2.1 Chebyshev Approximation

For the range $[a, b]$ and $n > 0$, choose $n + 1$ Chebyshev points $\{t_i\}_{1 \le i \le n+1}$ as

$$t_i = \frac{b+a}{2} + \frac{b-a}{2} \cdot \cos\left(\frac{2i-1}{2n+2}\pi\right).$$

For these points, the goal of the Chebyshev approximation of $f(x)$ is to find the degree n polynomial $p_n(t)$ satisfying $p_n(t_i) = f(t_i)$ for all $1 \leq i \leq n+1$. Due to the property of the Chebyshev points, there exists some $\psi_t \in [a,b]$ which makes the following inequality hold for $t \in [a,b]$.

$$||f(t) - p_n(t)|| \leq \frac{|f^{(n+1)}(\psi_t)|}{(n+1)!} \cdot \frac{1}{2^n} \cdot \left(\frac{b-a}{2}\right)^{n+1} \tag{1}$$

Note that the additional term $\frac{1}{2^n}$ is the reason why the Chebyshev method is much better than the Taylor approximation method when the degree n is large.

2.2 Full-RNS HEAAN

In this section, we introduce the fast base conversion in [2] and the `HEAAN-RNS` scheme in [6]. By using the fast base conversion which does not require CRT decomposition, we can keep ciphertexts of the `HEAAN-RNS` scheme in residue number systems (RNS) throughout homomorphic operations.

Fast Base Conversion. In the `HEAAN-RNS` scheme, a RNS representation

$$[a]_{\mathcal{C}} = (a^{(0)}, \ldots, a^{(\ell-1)}) \in \mathbb{Z}_{q_0} \times \cdots \times \mathbb{Z}_{q_{\ell-1}}$$

of an integer a with respect to $\mathbb{Z}_Q$ can be easily converted into its RNS representation with respect to $\mathbb{Z}_P$ by the equation

$$\texttt{Conv}_{\mathcal{C}\to\mathcal{B}}([a]_{\mathcal{C}}) = \left(\sum_{j=0}^{\ell-1} [a^{(j)} \cdot \hat{q}_j^{-1}]_{q_j} \cdot \hat{q}_j \pmod{p_i}\right)_{0 \leq i < k},$$

where $\hat{q}_j = \prod_{j' \neq j} q_{j'} \in \mathbb{Z}$, $P = \prod_{i=0}^{k} p_i$ and $Q = \prod_{j=0}^{l-1} q_j$. Since $a + Qe \in \mathbb{Z}_P$ for some small e is given as a result of this conversion, it includes the noise which can be ignored in the case of the `HEAAN-RNS` scheme. Even though the effect of this noise is negligible, we can reduce its size further by adapting the algorithms introduced in [11].

In [6], authors introduce two algorithms called `ModUp` and `ModDown` to expand and to reduce a modulus space, respectively (Algorithm 1 and 2 in [6]):

$$\begin{aligned}
\texttt{ModUp}_{\mathcal{C}\to\mathcal{D}}(\cdot) :& \prod_{j=0}^{\ell-1} R_{q_j} \to \prod_{i=0}^{k-1} R_{p_i} \times \prod_{j=0}^{\ell-1} R_{q_j} \\
:& [a]_{\mathcal{C}} \to (\texttt{Conv}_{\mathcal{C}\to\mathcal{B}}([a]_{\mathcal{C}}), [a]_{\mathcal{C}}), \\
\texttt{ModDown}_{\mathcal{D}\to\mathcal{C}}(\cdot) :& \prod_{i=0}^{k-1} R_{p_i} \times \prod_{j=0}^{\ell-1} R_{q_j} \to \prod_{j=0}^{\ell-1} R_{q_j} \\
:& ([a]_{\mathcal{B}}, [b]_{\mathcal{C}}) \to ([b]_{\mathcal{C}} - \texttt{Conv}_{\mathcal{B}\to\mathcal{C}}([a]_{\mathcal{B}})) \cdot [P^{-1}]_{\mathcal{C}},
\end{aligned}$$

where $\mathcal{D} = \{p_0, \ldots, p_{k-1}, q_0, \ldots, q_{\ell-1}\}$, $\mathcal{B} = \{p_0, \ldots, p_{k-1}\}$, $\mathcal{C} = \{q_0, \ldots, q_{\ell-1}\}$, and $P = \prod_{i=0}^{k-1} p_i$. Note that `ModUp` expands the modulus space of a from $\mathcal{C}$ to $\mathcal{D}$, and `ModDown` reverts the modulus space of a to the original and divide its value further by $P = \prod p_i$. These algorithms are used for modulus switching before and after key-switching, respectively.

Scheme Description. In this section, K is a $(2N)$-th cyclotomic field $\mathbb{Q}[X]/(X^N+1)$ and R is its ring of integers ($= \mathbb{Z}[X]/(X^N+1)$) for a power-of-two integer N. The residue ring modulo an integer q is denoted by $R_q = R/qR$.

$\underline{\texttt{Setup}(q, L, \eta; 1^\lambda)}$. As a base integer q, a number of levels L, a bit precision η, and a security parameter λ are given as inputs, we choose the followings according to them.

- A power-of-two integer N.
- A secret key distribution χ_{key}, an encryption key distribution χ_{enc}, and an error distribution χ_{err} over R.
- A basis $\mathcal{D} = \{p_0, \ldots, p_{k-1}, q_0, q_1, \ldots, q_L\}$ for which $q_j/q \in (1-2^{-\eta}, 1+2^{-\eta})$ for $1 \leq j \leq L$.

Next, we let $\mathcal{B} = \{p_0, \ldots, p_{k-1}\}$, $\mathcal{C}_\ell = \{q_0, \ldots, q_\ell\}$, and $\mathcal{D}_\ell = \mathcal{B} \cup \mathcal{C}_\ell$ for $0 \leq \ell \leq L$. Also, we let $P = \prod_{i=0}^{k-1} p_i$, $Q = \prod_{j=0}^{L} q_j$, $\hat{p}_i = \prod_{0 \leq i' < k, i' \neq i} p_{i'}$ for $0 \leq i < k$, and $\hat{q}_{\ell,j} = \prod_{0 \leq j' \leq \ell, j' \neq j} q_{j'}$ for $0 \leq j \leq \ell \leq L$. As the last step of Setup, we compute the followings.

- $[\hat{p}_i]_{q_j}$ and $[\hat{p}_i^{-1}]_{p_i}$ for $0 \leq i < k$, $0 \leq j \leq L$.
- $[P^{-1}]_{q_j} = \left(\prod_{i=0}^{k-1} p_i\right)^{-1} \pmod{q_j}$ for $0 \leq j \leq L$.
- $[\hat{q}_{\ell,j}]_{p_i}$ and $[\hat{q}_{\ell,j}^{-1}]_{q_j}$ for $0 \leq i < k$, $0 \leq j \leq \ell \leq L$.

$\underline{\texttt{KSGen}(s_1, s_2)}$. Sample $(a'^{(0)}, \ldots, a'^{(k+L)}) \leftarrow U\left(\prod_{i=0}^{k-1} R_{p_i} \times \prod_{j=0}^{L} R_{q_j}\right)$ and an error $e' \leftarrow \chi_{\mathsf{err}}$ first. With secret polynomials $s_1, s_2 \in R$ given as inputs, compute the switching key swk by

$$\left(\mathsf{swk}^{(0)} = (b'^{(0)}, a'^{(0)}), \ldots, \mathsf{swk}^{(k+L)} = (b'^{(k+L)}, a'^{(k+L)})\right) \in \prod_{i=0}^{k-1} R_{p_i}^2 \times \prod_{j=0}^{L} R_{q_j}^2,$$

where $b'^{(i)} \leftarrow -a'^{(i)} \cdot s_2 + e' \pmod{p_i}$ for $0 \leq i < k$ and $b'^{(k+j)} \leftarrow -a'^{(k+j)} \cdot s_2 + [P]_{q_j} \cdot s_1 + e' \pmod{q_j}$ for $0 \leq j \leq L$.

$\underline{\texttt{KeyGen}}$. First, sample $s \leftarrow \chi_{\mathsf{key}}$ and set the secret key as $\mathsf{sk} \leftarrow (1, s)$ and the evaluation key as $\mathsf{evk} \leftarrow \texttt{KSGen}(s^2, s)$. Next, sample $(a^{(0)}, \ldots, a^{(L)}) \leftarrow U\left(\prod_{j=0}^{L} R_{q_j}\right)$ and $e \leftarrow \chi_{\mathsf{err}}$ and set the public key as

$$\mathsf{pk} \leftarrow \left(\mathsf{pk}^{(j)} = (b^{(j)}, a^{(j)}) \in R_{q_j}^2\right)_{0 \leq j \leq L},$$

where $b^{(j)} \leftarrow -a^{(j)} \cdot s + e \pmod{q_j}$ for $0 \leq j \leq L$.

$\underline{\mathtt{Enc}_{\mathsf{pk}}(m)}$. First, sample $v \leftarrow \chi_{\mathsf{enc}}$ and $e_0, e_1 \leftarrow \chi_{\mathsf{err}}$. With a plaintext $m \in R$ given as a input, obtain the ciphertext $\mathsf{ct} = \left(\mathsf{ct}^{(j)}\right)_{0 \le j \le L} \in \prod_{j=0}^{L} R_{q_j}^2$, where $\mathsf{ct}^{(j)} \leftarrow v \cdot \mathsf{pk}^{(j)} + (m + e_0, e_1) \pmod{q_j}$ for $0 \le j \le L$.

$\underline{\mathtt{Dec}_{\mathsf{sk}}(\mathsf{ct})}$. Given a ciphertext $\mathsf{ct} = \left(\mathsf{ct}^{(j)}\right)_{0 \le j \le \ell} \in \prod_{j=0}^{l} R_{q_j}^2$, compute $\langle \mathsf{ct}^{(0)}, \mathsf{sk} \rangle \pmod{q_0}$.

$\underline{\mathtt{Add}(\mathsf{ct}, \mathsf{ct}')}$. Given two ciphertexts $\mathsf{ct} = \left(\mathsf{ct}^{(0)}, \dots, \mathsf{ct}^{(\ell)}\right), \mathsf{ct}' = \left(\mathsf{ct}'^{(0)}, \dots, \mathsf{ct}'^{(\ell)}\right) \in \prod_{j=0}^{\ell} R_{q_j}^2$, obtain the ciphertext $\mathsf{ct}_{\mathsf{add}} = \left(\mathsf{ct}_{\mathsf{add}}^{(j)}\right)_{0 \le j \le \ell}$, where $\mathsf{ct}_{\mathsf{add}}^{(j)} \leftarrow \mathsf{ct}^{(j)} + \mathsf{ct}'^{(j)} \pmod{q_j}$ for $0 \le j \le \ell$.

$\underline{\mathtt{Mult}_{\mathsf{evk}}(\mathsf{ct}, \mathsf{ct}')}$. Given two ciphertexts $\mathsf{ct} = \left(\mathsf{ct}^{(j)} = (c_0^{(j)}, c_1^{(j)})\right)_{0 \le j \le \ell}$ and $\mathsf{ct}' = \left(\mathsf{ct}'^{(j)} = (c_0'^{(j)}, c_1'^{(j)})\right)_{0 \le j \le \ell}$, perform the following computations in turn and obtain the ciphertext $\mathsf{ct}_{\mathsf{mult}} \in \prod_{j=0}^{\ell} R_{q_j}^2$.

- $d_0^{(j)} \leftarrow c_0^{(j)} c_0'^{(j)} \pmod{q_j}$, $d_1^{(j)} \leftarrow c_0^{(j)} c_1'^{(j)} + c_1^{(j)} c_0'^{(j)} \pmod{q_j}$, and $d_2^{(j)} \leftarrow c_1^{(j)} c_1'^{(j)} \pmod{q_j}$ for $0 \le j \le \ell$.
- $\mathtt{ModUp}_{\mathcal{C}_\ell \to \mathcal{D}_\ell}(d_2^{(0)}, \dots, d_2^{(\ell)}) = (\tilde{d}_2^{(0)}, \dots, \tilde{d}_2^{(k-1)}, d_2^{(0)}, \dots, d_2^{(\ell)})$.
- $\tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}^{(0)} = (\tilde{c}_0^{(0)}, \tilde{c}_1^{(0)}), \dots, \tilde{\mathsf{ct}}^{(k+\ell)} = (\tilde{c}_0^{(k+\ell)}, \tilde{c}_1^{(k+\ell)})) \in \prod_{i=0}^{k-1} R_{p_i}^2 \times \prod_{j=0}^{\ell} R_{q_j}^2$, where $\tilde{\mathsf{ct}}^{(i)} = \tilde{d}_2^{(i)} \cdot \mathsf{evk}^{(i)} \pmod{p_i}$ and $\tilde{\mathsf{ct}}^{(k+j)} = d_2^{(j)} \cdot \mathsf{evk}^{(k+j)} \pmod{q_j}$ for $0 \le i < k$, $0 \le j \le \ell$.
- $\left(\hat{c}_0^{(0)}, \dots, \hat{c}_0^{(\ell)}\right) \leftarrow \mathtt{ModDown}_{\mathcal{D}_\ell \to \mathcal{C}_\ell}\left(\tilde{c}_0^{(0)}, \dots, \tilde{c}_0^{(k+\ell)}\right)$ and $\left(\hat{c}_1^{(0)}, \dots, \hat{c}_1^{(\ell)}\right) \leftarrow \mathtt{ModDown}_{\mathcal{D}_\ell \to \mathcal{C}_\ell}\left(\tilde{c}_1^{(0)}, \dots, \tilde{c}_1^{(k+\ell)}\right)$.
- $\mathsf{ct}_{\mathsf{mult}} = (\mathsf{ct}_{\mathsf{mult}}^{(j)})_{0 \le j \le \ell}$, where $\mathsf{ct}_{\mathsf{mult}}^{(j)} \leftarrow (\hat{c}_0^{(j)} + d_0^{(j)}, \hat{c}_1^{(j)} + d_1^{(j)}) \pmod{q_j}$ for $0 \le j \le \ell$.

$\underline{\mathtt{RS}(\mathsf{ct})}$. Given a ciphertext $\mathsf{ct} = \left(\mathsf{ct}^{(j)} = (c_0^{(j)}, c_1^{(j)})\right)_{0 \le j \le \ell} \in \prod_{j=0}^{\ell} R_{q_j}^2$, compute the ciphertext $\mathsf{ct}' \leftarrow \left(\mathsf{ct}'^{(j)} = (c_0'^{(j)}, c_1'^{(j)})\right)_{0 \le j \le \ell-1} \in \prod_{j=0}^{\ell-1} R_{q_j}^2$, where $c_i'^{(j)} \leftarrow q_\ell^{-1} \cdot \left(c_i^{(j)} - c_i^{(\ell)}\right) \pmod{q_j}$ for $i = 0, 1$ and $0 \le j < \ell$.

2.3 Bootstrapping for HEAAN

In this section, we briefly describe the overall process of the bootstrapping for `HEAAN` suggested in [7], and the improvements introduced in [5, 12]. As in the previous section, we let $R = \mathbb{Z}[X]/(X^N + 1)$ for a power-of-two integer N and $R_q = R/qR$. For $a \in \mathbb{Z}_q$, assign a the unique integer, which is equivalent to $a \pmod{q}$ and is contained in $\mathbb{Z} \cap (-q/2, q/2]$, and denote it by $[a]_q$. Extend this definition to R_q by applying it component-wisely.

Let ct be a ciphertext of $m(X)$ relative to the secret key sk and the ciphertext modulus q. Note that $m(X) = [\langle \mathsf{ct}, \mathsf{sk} \rangle]_q$. The goal of the bootstrapping is to find an encryption of $m(X)$ with a bigger ciphertext modulus. In other words, we hope to find ct' and a modulus $Q > q$ satisfying $m(X) = [\langle \mathsf{ct}', \mathsf{sk}\rangle]_Q$. The overall process can be divided into four steps; Modulus Raising, Coefficients to Slots, Sine Evaluation and Slots to Coefficients.

Modulus Raising. Consider a polynomial $t(X) = \langle \mathsf{ct}, \mathsf{sk}\rangle$ of deg $< N$. Under the assumption that the message m is much smaller than the ciphertext modulus q, $t(X)$ can be represented as $t(X) = qI(X) + m(X)$, where $I(X) \in R$ and all the coefficients of I are bounded by a constant K which is determined by the secret key distribution of the scheme. If we choose $Q_0 \gg q$, then it follows that $t(X) = [\langle \mathsf{ct}, \mathsf{sk}\rangle]_{Q_0}$. Therefore, ct can be regarded as an encryption of $t(X)$ with respect to the modulus Q_0.

Coefficients to Slots. Before introducing about this step, we need to recall encoding and decoding procedures of HEAAN. Let ξ be a primitive $2N$-th root of unity and $\xi_i = \xi^{5i}$ for $0 \leq i < N/2$. Since 5 has the order $N/2$ modulo $2N$ and spans $\mathbb{Z}_{2N}^*$ with -1, $\{\xi_i, \bar{\xi_i} : 0 \leq i < N/2\}$ is the set of all primitive $2N$-th roots of unity. Now, we can define a decoding map $\tau : \mathbb{R}[X]/(X^N + 1) \to \mathbb{C}^{N/2}$ by $\tau(m(X)) = (m(\xi_j))_{0 \leq j < N/2}$. We say $m(X)$ has values $m(\xi_0), \cdots, m(\xi_{\frac{N}{2}-1})$ in its slots or $m(X)$ is the plaintext of $(m(\xi_j))_{0 \leq j < N/2}$ in this case. If we identifying each element $m(X) = m_0 + m_1X + \cdots + m_{N-1}X^{N-1}$ of $\mathbb{R}[X]/(X^N + 1)$ with $\mathbf{m} = (m_0, \cdots, m_{N-1}) \in \mathbb{R}^N$, the decoding map can be considered as a linear transformation from $\mathbb{R}^N$ to $\mathbb{C}^{N/2}$, which is characterized by the matrix

$$\mathbf{U} = \begin{pmatrix} 1 & \xi_0 & \xi_0^2 & \cdots & \xi_0^{N-1} \\ 1 & \xi_1 & \xi_1^2 & \cdots & \xi_1^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \xi_{\frac{N}{2}-1} & \xi_{\frac{N}{2}-1}^2 & \cdots & \xi_{\frac{N}{2}-1}^{N-1} \end{pmatrix}.$$

With this identification, the decoding process can be simply written as $\mathbf{m} \mapsto \mathbf{U} \cdot \mathbf{m}$ for $\mathbf{m} \in \mathbb{R}^N$. Moreover, an encoding map is just an inverse map of the decoding map, and it can be checked that the encoding process can be written as $\mathbf{z} \mapsto \frac{1}{N}(\bar{\mathbf{U}}^T\mathbf{z} + \mathbf{U}^T\bar{\mathbf{z}})$ for $\mathbf{z} \in \mathbb{C}^{N/2}$.

Given a polynomial $t(X) = t_0 + t_1X + \cdots + t_{N-1}X^{N-1}$ from the previous step, this step aims to get the ciphertext whose corresponding plaintext has values $t_0, \cdots, t_{N-1}$ in its slots. Since each plaintext can have at most $N/2$ values, it is impossible that just one plaintext has those values. Thus, we will find two ciphertexts that correspond to plaintexts of the vectors $\mathbf{z_1} = (t_0, \cdots, t_{\frac{N}{2}-1})$ and $\mathbf{z_2} = (t_{\frac{N}{2}}, \cdots, t_{N-1})$, respectively.

Let $\mathbf{z} = \tau(t) \in \mathbb{C}^{N/2}$ be the vector that corresponds to the ciphertext ct. If we divide the matrix $\mathbf{U}$ into following two square matrices

$$\mathbf{V} = \begin{pmatrix} 1 & \xi_0 & \cdots & \xi_0^{\frac{N}{2}-1} \\ 1 & \xi_1 & \cdots & \xi_1^{\frac{N}{2}-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \xi_{\frac{N}{2}-1} & \cdots & \xi_{\frac{N}{2}-1}^{\frac{N}{2}-1} \end{pmatrix} \quad \text{and} \quad \mathbf{W} = \begin{pmatrix} \xi_0^{\frac{N}{2}} & \xi_0^{\frac{N}{2}+1} & \cdots & \xi_0^{N-1} \\ \xi_1^{\frac{N}{2}} & \xi_1^{\frac{N}{2}+1} & \cdots & \xi_1^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \xi_{\frac{N}{2}-1}^{\frac{N}{2}} & \xi_{\frac{N}{2}-1}^{\frac{N}{2}+1} & \cdots & \xi_{\frac{N}{2}-1}^{N-1} \end{pmatrix},$$

it follows that $\mathbf{z_1} = \frac{1}{N}(\bar{\mathbf{V}}^T\mathbf{z}+\mathbf{V}^T\bar{\mathbf{z}})$ and $\mathbf{z_2} = \frac{1}{N}(\bar{\mathbf{W}}^T\mathbf{z}+\mathbf{W}^T\bar{\mathbf{z}})$. These equations mean that $\mathbf{z_1}$ and $\mathbf{z_2}$ can be obtained by applying linear transformations to $\mathbf{z}$, and thus we can also get corresponding ciphertexts by applying the same linear transformations homomorphically to ct.

Sine Evaluation. This step is the hardest part of the bootstrapping for `HEAAN`. This step aims to perform the function $f(t) = [t]_q$ homomorphically. More precisely, given two ciphertexts ct_1 and ct_2 corresponding to $\mathbf{z_1}$ and $\mathbf{z_2}$, respectively, we hope to obtain ciphertexts corresponding to the plaintexts of $\mathbf{z'_1} = (m_0, \cdots, m_{\frac{N}{2}-1})$ and $\mathbf{z'_2} = (m_{\frac{N}{2}}, \cdots, m_{N-1})$, where $t_j = qI_j + m_j$ for $0 \le j < N$, by applying the function f to ct_1 and ct_2. However, the problem is that the function f is hard to approximate by a polynomial. Therefore, the function $g(t) = \frac{q}{2\pi}\sin(\frac{2\pi t}{q})$ is used as an approximation of f. Since t can be represented as $t = qI + m$, where $|I| < K$ and $m \ll q$, the difference between $f(t)$ and $g(t)$ is given as

$$|f(t) - g(t)| = \frac{q}{2\pi}\left|\frac{2\pi m}{q} - \sin(\frac{2\pi m}{q})\right| \le \frac{q}{2\pi}\cdot\frac{1}{6}\left|\frac{2\pi m}{q}\right|^3,$$

which is small enough due to $m \ll q$. Also, since g is a smooth function, it is easier to approximate it by a polynomial. For these reasons, the sine function g is a good approximation of the modulus reduction function f, and the main interest of this step becomes how to approximate g by a polynomial.

The ways to approximate g by a polynomial differ in previous works [7, 5]. In [7], to evaluate $\sin(t)$, they first scale down t by a power of two to make it locate close enough to the origin, and they use the Taylor polynomial approximation for the evaluation of sine in a small interval around the origin. After that, they compute the original sine value at t by using double angle formula. On the other hand, in [5], they use the Chebyshev polynomial approximation instead of the Taylor approximation.

Slots to Coefficients. This is the final step, and it is just a reverse process of CoefficientstoSlots. Given two ciphertexts ct'_1 and ct'_2 corresponding to $\mathbf{z'_1} = (m_0, \cdots, m_{\frac{N}{2}-1})$ and $\mathbf{z'_2} = (m_{\frac{N}{2}}, \cdots, m_{N-1})$, respectively, the goal of this step is to find the ciphertext whose plaintext is $m(X) = m_0 + m_1X + \cdots + m_{N-1}X^{N-1}$. If we let $\mathbf{z}' = \tau(m)$, it follows that $\mathbf{z}' = \mathbf{Um} = \mathbf{V}\mathbf{z'_1} + \mathbf{W}\mathbf{z'_2}$. Therefore, we can get the ciphertext that we want by taking the same linear combination homomorphically to ct'_1 and ct'_2. As a result, through the bootstrapping process, we can get a ciphertext ct' which is an encryption of $m(X)$ relative to a ciphertext modulus Q, which is smaller than Q_0, but much bigger enough than q.

Remark 1. Improved linear transformation. Applying linear transformation to a ciphertext with n slots originally requires $O(n)$ homomorphic operations, and it can be improved a lot by using special structure of the matrix $\mathbf{U}$ [5, 12]. In [5, 12], they decompose the linear transformation part (including "Coefficients to Slots" and "Slots to Coefficients") and reduce the complexity to $O(r \log_r n)$ while consuming $O(\log_r n)$ depth.

3 Full-RNS variant of HEAAN

Before starting this section, we note that we use the same notation as in Section 2. In the `HEAAN-RNS` scheme, since the decomposition methods are not used for key-switching, it only requires one evaluation key for key-switching. Also, since $P = \prod_{i=0}^{k} p_i$ should be much bigger than $Q = \prod_{j=0}^{L-1} q_j$ to effectively reduce the size of noise added through key-switching, k, which indicates the number of temporary moduli, should be $\simeq L$ [4]. On the other hand, `SEAL` (v 3.3) uses one temporary modulus and the RNS-decomposition technique, which will be introduced later, to perform key-switching. In other words, they set the number of temporary moduli k to 1.

Using large k ($k \simeq L$) and small k ($k = 1$) has pros and cons, respectively. By using large $k \simeq L$, we only need one evaluation key for key-switching, and thus smaller complexity for key-switching is required. However, since the security of the scheme depends on the largest ciphertext modulus $\prod_{i=0}^{k-1} p_i \cdot \prod_{i=0}^{L} q_i$, the bit size of $\prod_{i=0}^{k-1} p_i \cdot \prod_{i=0}^{L} q_i$ should be fixed when we assume the same security level. Therefore, using large $k \simeq L$ forces us to use a smaller number of q_j's, and it follows that the less depth computations are supported by the scheme.

As noted above, there is a trade-off between the complexity of key-switching and the number of levels supported by the scheme. Therefore, in many cases, it is better to choose an appropriate value of k between 1 and L rather than using extremely large k ($k \simeq L$) and extremely small k ($k = 1$). However, the `HEAAN-RNS` scheme and `SEAL` (v 3.3) only support $k \simeq L$ and $k = 1$ case, respectively. Hence, the main goal of our scheme is to propose a generalized `HEAAN-RNS` scheme, which also includes the scheme of `SEAL` (v.3.3), to make it possible to use optimal k for each situation.

Full-RNS decomposition. First, we will introduce the RNS-decomposition technique in [2]. The RNS-decomposition method can be represented by the following equations:

$$\mathsf{RNS\text{-}Decomp}_{\mathcal{C}}(a(x)) = ([a(x) \cdot \hat{q_0}^{-1}]_{q_0}, \ldots, [a(x) \cdot \hat{q_L}^{-1}]_{q_L}) \in R^{L+1}$$
$$\mathsf{RNS\text{-}Power}_{\mathcal{C}}(b(x)) = (b(x) \cdot \hat{q_0}, b(x) \cdot \hat{q_1}, \ldots, b(x) \cdot \hat{q_L}) \in R^{L+1},$$

where $\mathcal{C} = \{q_0, q_1, \ldots, q_L\}$, $\hat{q_i} = \prod_{j \neq i} q_j$ and $a(x), b(x) \in R$. Here, $f(x) \cdot c$ indicates the multiplication between each coefficient of polynomial $f(x)$ and c. Those functions work similarly as the bit-decomposition and power of two technique:

$$a(x) \cdot b(x) = \langle \mathsf{RNS\text{-}Decomp}_{\mathcal{C}}(a(x)), \mathsf{RNS\text{-}Power}_{\mathcal{C}}(b(x)) \rangle \in R_Q,$$

where $Q = \prod_{i=0}^{L} q_i$. Since the sizes of coefficients of $\mathsf{RNS\text{-}Decomp}_{\mathcal{C}}(a(x))$ are less than $\mathsf{max}_{0 \le i \le L}(q_i) \ll Q$, the functions can be used for key-switching. More

[4] In practice, p_i's are chosen to have maximum sizes within the word size (< 64 bits). On the other hand, sizes of q_j's are depend on the precision of applications, and usually they are $40 \sim 45$ bits.

precisely, we can replace the multiplication of $a(x)$ and $s(x)^2$ in key-switching with

$$\langle \textsf{RNS-Decomp}_{\mathcal{C}}(a(x)), \textsf{RNS-Power}_{\mathcal{C}}(s(x)^2) + \texttt{Enc}_{s(x)}(0)\rangle \bmod Q.$$

Here, we add the term $\texttt{Enc}_{s(x)}(0)$ not to reveal information on $s(x)^2$. In this method, since both vectors have length $L+1$, the complexity of the inner product is quadratic to L, which is not favorable. Furthermore, the noise growth through key-switching is about $||e_{\textsf{fresh}}|| \cdot \textsf{max}(||q_i||)$, which is quite large compared to the size of a plaintext.

Overview of Idea. To solve those problems stated above, we will reduce the length of the vectors and control the noise growth using temporary moduli [5]. First, we use the partial products $\{Q_j\}_{0\le j<\texttt{dnum}} = \{\prod_{i=j\alpha}^{(j+1)\alpha-1} q_i\}_{0\le j<\texttt{dnum}}$, where $\alpha = (L+1)/\texttt{dnum}$ for a pre-fixed parameter $\texttt{dnum}$, instead of using $\{q_i\}_{0\le i\le L}$ in the RNS-decomposition. Then, it follows that $P = \prod_{i=0}^{k-1} p_i$ only needs to be bigger than $\max_{0\le j<\texttt{dnum}} Q_j$ to effectively reduce the size of noise added through key-switching. Therefore, we can set the number of temporary moduli k to α. In addition, we apply the fast base conversion to avoid CRT composition in re-linearization (key-switching in multipication). The brief sketch about our re-linearization method can be represented as follows:

0. For $k = (L+1)/\texttt{dnum}$, set an evaluation key as $\textsf{evk} = \textsf{RNS-Power}_{\mathcal{C}'}(s(x)^2) + \texttt{Enc}_{s(x)}(0) \bmod PQ$, where $P = \prod_{i=0}^{k-1} p_i$, $Q = \prod_{i=0}^{L} q_i$ and $\mathcal{C}' = \{Q_j\}_{0\le j<\texttt{dnum}}$.
1. For a given ciphertext $(c(x), b(x), a(x))$ such that $c(x) + b(x)\cdot s(x) + a(x)\cdot s(x)^2 = m + e \mod Q$, we compute

$$(b'(x), a'(x)) = \langle \textsf{RNS-Decomp}_{\mathcal{C}'}(a(x)), \textsf{evk}\rangle \bmod PQ.$$

   In $\textsf{RNS-Decomp}_{\mathcal{C}'}(a(x)) \bmod PQ$ computation, we avoid CRT composition using the fast base conversion.
2. We apply modulus-switching using `ModDown` to reduce the size of noise:

$$(b''(x), a''(x)) = \lfloor (b'(x), a'(x))/P \rceil \bmod Q$$

3. Return $(c(x) + b''(x), b(x) + a''(x))$.

Since we just make differences in key-switching, we only need to revise $\underline{\texttt{KSGen}(s_1, s_2)}$, $\underline{\texttt{KeyGen}}$, and $\underline{\texttt{Mult}_{\textsf{evk}}(\textsf{ct}, \textsf{ct}')}$ in Section 2. We remark that the case of $\underline{\texttt{dnum} = 1}$ is same as the $\underline{\texttt{HEAAN-RNS}}$ scheme. Using larger `dnum` increases the number of evaluation keys, but it reduces the dimension of ring or increases the parameter L when we assume the same level of security. Detailed comparisons with `HEAAN-RNS` and `SEAL` (v.3.3) are contained in Section 3.2.

[5] In the case of `SEAL` v3.2, they use the bit-decomposition technique with the RNS-decomposition to reduce the noise growth. But, this method also has a drawback. It increases the length of the public key vector for key-switching further, which is directly related to the complexity of the process.

3.1 Scheme description

We will focus on differences between the HEAAN-RNS scheme and ours. The other parts which are not mentioned in this section are same as the scheme in Section 2. In this section, let

$$\mathcal{C}' = \{Q_j\}_{0 \le j < \mathtt{dnum}} = \left[\prod_{i=j\alpha}^{(j+1)\alpha-1} q_i \right]_{0 \le j < \mathtt{dnum}}$$

for a given integer $\mathtt{dnum} > 0$ and $\alpha = (L+1)/\mathtt{dnum}$. Also, let $\hat{Q}_j = \prod_{i \ne j} Q_i$ and $P = \prod_{i=0}^{k-1} p_i$, and assume $|P| \ge \max_{0 \le j < \mathtt{dnum}}(Q_j)$.

$\underline{\mathtt{KSGen}(s_1, s_2, \mathtt{dnum})}$. For given secret polynomials $s_1, s_2 \in R$, sample $(a'^{(0)}, \ldots, a'^{(k+L)}) \leftarrow U\left(\prod_{i=0}^{k-1} R_{p_i} \times \prod_{j=0}^{L} R_{q_j}\right)$ and sample an error $e' \leftarrow \chi_{\mathsf{err}}$. Output switching keys $\{\mathsf{swk}_j\}_{0 \le j < \mathtt{dnum}}$ as

$$\left(\mathsf{swk}_j^{(0)} = (b'^{(0)}_j, a'^{(0)}_j), \ldots, \mathsf{swk}_j^{(k+L)} = (b'^{(k+L)}_j, a'^{(k+L)}_j)\right) \in \prod_{i=0}^{k-1} R_{p_i}^2 \times \prod_{i=0}^{L} R_{q_i}^2,$$

where $b'^{(i)}_j \leftarrow -a'^{(i)}_j \cdot s_2 + e' \pmod{p_i}$ for $0 \le i < k$ and $b'^{(k+i)}_j \leftarrow -a'^{(k+i)}_j \cdot s_2 + [P]_{q_i} \cdot [\hat{Q}_j]_{q_i} \cdot s_1 + e' \pmod{q_i}$ for $0 \le i \le L$.

$\underline{\mathtt{KeyGen}}$. First, sample $s \leftarrow \chi_{\mathsf{key}}$ and set a secret key as $\mathsf{sk} \leftarrow (1, s)$ and evaluation keys as $\{\mathsf{evk}_i\}_{0 \le j < \mathtt{dnum}} \leftarrow \mathtt{KSGen}(s^2, s)$.

For convenience, let $\mathcal{C}_i = \{q_0, \ldots, q_i\}$, $\mathcal{C}'_i = \{q_{i\alpha}, \ldots, q_{((i+1)\alpha-1}\}$ and let $\mathcal{D}_i = (\cup_{0 \le j < i} \mathcal{C}'_j) \cup \{p_0, \ldots, p_{k-1}\}$.

$\underline{\mathtt{Mult}_{\mathsf{evk}}(\mathsf{ct}, \mathsf{ct}')}$. Given two ciphertexts $\mathsf{ct} = \left(\mathsf{ct}^{(j)} = (c_0^{(j)}, c_1^{(j)})\right)_{0 \le j \le \ell}$ and $\mathsf{ct}' = \left(\mathsf{ct}'^{(j)} = (c_0'^{(j)}, c_1'^{(j)})\right)_{0 \le j \le \ell}$, perform the followings and return the ciphertext $\mathsf{ct}_{\mathsf{mult}} \in \prod_{j=0}^{\ell} R_{q_j}^2$.

1. For $0 \le j \le \ell$, compute

$$\begin{aligned} d_0^{(j)} &\leftarrow c_0^{(j)} c_0'^{(j)} \pmod{q_j}, \\ d_1^{(j)} &\leftarrow c_0^{(j)} c_1'^{(j)} + c_1^{(j)} c_0'^{(j)} \pmod{q_j}, \\ d_2^{(j)} &\leftarrow c_1^{(j)} c_1'^{(j)} \pmod{q_j}. \end{aligned}$$

2. RNS-Decompose:
   2-1. Zero-padding and Split: Let $\beta = \lceil (\ell+1)/\alpha \rceil$,

$$d'^{(i)}_{2,j} = \begin{cases} d_2^{(j\alpha+i)} \cdot [Q']_{q_{j\alpha+i}} & \text{if } j\alpha + i \le \ell \\ 0 & \text{otherwise} \end{cases}$$

   for $0 \le i < \alpha$, $0 \le j < \beta$ and $Q' = \prod_{i=\ell+1}^{\alpha\beta-1} q_i$.

2-2. RNS-Decompose:

$$d'^{(i)}_{2,j} \leftarrow d'^{(i)}_{2,j} \cdot [\hat{Q}_j^{-1}]_{q_{j\alpha+i}}$$

for $0 \le i < \alpha$ and $0 \le j < \beta$ with $j\alpha + i \le \ell$.

3. Modulus-Raise: compute $\tilde{d}_{2,j} = \mathtt{ModUp}_{\mathcal{C}'_j \to \mathcal{D}_\beta}(d'_{2,j})$.
4. Inner Product: compute

$$\tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}^{(0)} = (\tilde{c}_0^{(0)}, \tilde{c}_1^{(0)}), \dots, \tilde{\mathsf{ct}}^{(k+\ell)} = (\tilde{c}_0^{(k+\ell)}, \tilde{c}_1^{(k+\ell)})) \in \prod_{i=0}^{k-1} R_{p_i}^2 \times \prod_{j=0}^{\ell} R_{q_j}^2,$$

where $\tilde{\mathsf{ct}}^{(i)} = \sum_{j=0}^{\beta-1} \tilde{d}_{2,j}^{(i)} \cdot \mathsf{evk}_j^{(i)} \pmod{p_i}$ for $0 \le i < k$ and $\tilde{\mathsf{ct}}^{(k+i)} = \sum_{j=0}^{\beta-1} \tilde{d}_{2,j}^{(k+i)} \cdot \mathsf{evk}_j^{(k+i)} \pmod{q_i}$ for $0 \le i < \alpha\beta$.

5. Modulus-Down: compute

$$\left(\hat{c}_0^{(0)}, \dots, \hat{c}_0^{(\ell)}\right) \leftarrow \mathtt{ModDown}_{\mathcal{D}_\beta \to \mathcal{C}_\ell}\left(\tilde{c}_0^{(0)}, \dots, \tilde{c}_0^{(k+\alpha\beta-1)}\right),$$
$$\left(\hat{c}_1^{(0)}, \dots, \hat{c}_1^{(\ell)}\right) \leftarrow \mathtt{ModDown}_{\mathcal{D}_\beta \to \mathcal{C}_\ell}\left(\tilde{c}_1^{(0)}, \dots, \tilde{c}_1^{(k+\alpha\beta-1)}\right).$$

6. Output the ciphertext $\mathsf{ct}_{\mathsf{mult}} = (\mathsf{ct}_{\mathsf{mult}}^{(j)})_{0 \le j \le \ell}$, where $\mathsf{ct}_{\mathsf{mult}}^{(j)} \leftarrow (\hat{c}_0^{(j)} + d_0^{(j)}, \hat{c}_1^{(j)} + d_1^{(j)}) \pmod{q_j}$ for $0 \le j \le \ell$.

Figure 1 shows the overall process of our multiplication algorithm from Step 2 to Step 4, which are the key parts of our algorithm. The gray area in Figure 1 indicates temporary moduli $\{p_0, p_1, \dots, p_{k-1}\}$.

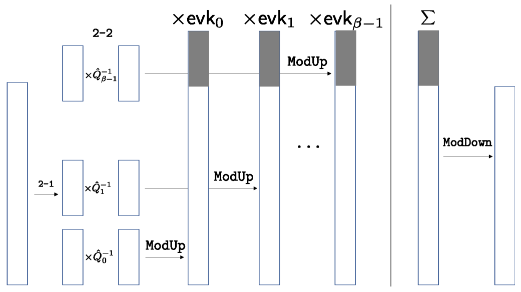


Fig. 1. Overview of our algorithm from Step 2 to Step 5.

Remark 2 (Correctness). The correctness of our scheme is directly followed from the correctness of ModUp, ModDown, RNS-Decomp, RNS-Power. Detailed proof for the correctness and noise growth is contained in Appendix.

Remark 3 (Security). In our scheme, we change the key generation and the multiplication algorithm. A way of generating evaluation key is changed, but our evaluation key is also an addition of information about secret key and encryption of zero. Therefore, the security of our scheme is also based on the Ring-LWE problem (same as HEAAN-RNS).

Remark 4 (Quantization and Batching). The homomorphic encryption scheme in Section 2 is described for a plaintext $m(x) \in R = \mathbb{Z}[X]/(X^N + 1)$. To encrypt a vector of complex numbers, we use an isomorphism between $\mathbb{C}^{N/2}$ and $\mathbb{R}[X]/(X^N + 1)$. Choose an isomorphism $\rho : \mathbb{C}^{N/2} \to \mathbb{R}[X]/(X^N + 1)$ and define $\mathsf{Encode}(\boldsymbol{m}, \Delta) = \lceil \Delta \cdot \rho^{-1}(\boldsymbol{m}) \rfloor = m(x) \in R$. The HEAAN-RNS scheme uses the same scaling factor Δ for all levels by letting $q_i \simeq \Delta$ for all i. However, this method yields additional noise in Rescale process. Therefore, as in SEAL, we use different scaling factors for each level, which means that we just regard Rescale process as dividing scaling factor by q_i.

3.2 Comparison

In this section, we compare our scheme with HEAAN-RNS and SEAL v3.3. We note that the HEAAN-RNS scheme and SEAL v3.3 can be regarded as ($\mathtt{dnum} = 1$)-case and ($\mathtt{dnum} = L+1$)-case of our scheme, respectively. Before comparison, we check the complexity of the homomorphic multiplication of our scheme.

Complexity of Homomorphic Multiplication. We assume that ciphertexts and evaluation keys are NTT (Number-theoretic transform) transformed in advance. Also, we only count the the number of multiplications in $\mathbb{Z}_{p_i}$ or $\mathbb{Z}_{q_i}$ for complexity. The followings are complexity for each step of the multiplication. Here, we ignore the last step which only needs some additions (no multiplication).

Step 1. This step computes tensor product of two vectors (with length 2). By performing in the sense of Karatsuba multiplication, it only requires 3 polynomial multiplications. Therefore, this step requires 3 Hadamard multiplications and 3 inverse NTT transformations for each ring R_{q_i}[6]: $3(\ell + 1)N + 3(\ell + 1)N \log N$

Step 2-2. Step 2-1 just rearranges the vector and zero-padding, which requires no complexity, and the multiplications in Step 2-1 can be merged into the multiplications in Step 2-2. Hence, we can ignore this part. In Step 2-2, ℓ modulus multiplications are needed: $\ell \cdot N$.

Step 3. ModUp requires $n(m - n)$ multiplications for input vector size n and output vector size m. Hence, the complexity for $\mathtt{ModUp}_{\mathcal{C}_j \to \mathcal{D}_\beta}$ is $\alpha(\alpha\beta + k - \alpha)$. Since we have to perform it β times for each coefficient, the total complexity is $\alpha\beta(\alpha\beta + k - \alpha)N$.

Step 4. This step can be divided into the following 4 sub-steps:

1. NTT transform: we need to apply the NTT algorithm for each $\{\tilde{d}_{2,j}^{(i)}\}_{0 \le i < k+\alpha\beta}$:
$$\beta(k + \alpha\beta)N \log N$$
2. Hadamard Mult.: $2\beta(k + \alpha\beta)N$.
3. Summation: there is no multiplication.
4. Inverse NTT transform: $(k + \alpha\beta)N \log N$.

[6] In Step 1, inverse NTT transform is needed for the next step (modulus raising).

N	$\ell+1$	k	$\log_2 PQ$	λ	$\log_2$(Total Complexity)	
65536	24	1	1136	147.6	29.67	SEAL v3.3
		2	1181	144.2	28.94	
		3	1227	141.3	28.60	
		4	1272	138.3	28.39	
		6	1363	132.4	28.16	
		8	1454	128.3	28.05	$\lambda > 128$
		12	1635	118.2	27.96	
		24	2180	94.3	27.97	HEAAN-RNS

Table 1. Complexity of homomorphic multiplication for fixed ℓ.

N	$\log_2 PQ$	λ	$\ell+1$	k	$\log_2$(Total Complexity)	
65536	1450	133.7	15	15	27.13	HEAAN-RNS
			20	10	27.64	
			24	6	28.16	
			27	3	28.88	
			29	1	30.17	SEAL v3.3

Table 2. Complexity of homomorphic multiplication for fixed $\log_2 PQ$ and λ.

Step 5. ModDown requires $m(n-m)$ multiplications for input vector size n and output vector size m. Hence, the complexity for $\texttt{ModDown}_{\mathcal{D}_\beta \to \mathcal{C}_\ell}$ is $\ell(k+\alpha\beta-\ell)$. Since we have to perform it for each coefficient, the total complexity of this step is $\ell \cdot (k+\alpha\beta-\ell) \cdot N$.

If we set k to α and regard all parameters except k as constants, the total complexity of the multiplication is approximately

$$N\left\{(l+\log N)\cdot k + (2+\log N)l^2 \cdot (1/k)\right\} + (\text{constant})$$

since $\alpha \cdot \beta \simeq l$. Hence, the total complexity is minimized when $k = \sqrt{\frac{2+\log N}{l+\log N}} \cdot l$. As a result, from the point of view of complexity, it is better to use proper k between 1 and l.

Comparison. Now, we compare various parameter sets for our scheme, which have different k values. First, Table 1 shows parameter sets with various k for fixed ℓ, and corresponding complexity of homomorphic multiplication. Note that the first and the last row correspond to SEAL v3.3 and HEAAN-RNS scheme[7], respectively. Suppose that we need an optimal parameter set which has $\ell+1 = 24$ and ensures $\lambda > 128$. Then, it is better to set k to 8, which requires the lowest complexity among the values that ensure enough security.

Table 2 shows parameter sets with various ℓ and k for fixed $\log_2 PQ$, and corresponding complexity of homorphic multiplication. As seen from the table, HEAAN-RNS can only support depth 14 computation without bootstrapping. On

[7] Here, SEAL v.3.3 and HEAAN-RNS indicate the scheme corresponding to each paper and library.

the other hand, SEAL v3.3 supports depth 28 computation, but it requires the largest complexity and public key size for re-linearization. Since there is a trade-off between supported depth computation and complexity, it is important to choose proper k depending on the situation that we are in.

4 Better Homomorphic Sine Evaluation

As mentioned in Section 2.3, the key part of the bootstrapping for HEAAN is a homomorphic evaluation of a sine function. In other words, the way to approximate a sine function by a polynomial is important in the bootstrapping for HEAAN. Also, all the previous works [5, 7] can be simply represented by

$$[t]_q \;\simeq\; \frac{q}{2\pi}\sin(\tfrac{2\pi}{q}t) \;\simeq\; p(t)$$

for some suitable polynomial $p(t) \in \mathbb{R}[X]$, and the difference between those works [5, 7] occurs in the step of approximating a sine function by a polynomial.

Recall that an input value t can be represented as $t = qI + m$ for some $|I| < K$ and $m \ll q$. Hence, $\frac{t}{q}$ locates close enough to some integer, and it is the reason why the first approximation of the modulus operation with the sine function is reasonable. However, all the previous works [5, 7] do not use this property in the second approximation. In other words, they just find a polynomial that approximates a sine function well in a global sense. Therefore, there is a room for finding a better approximate polynomial $p(t)$ based on the property.

From now on, by scaling and shifting t, we approximate $\cos(2\pi t)$ instead of $\sin(\frac{2\pi t}{q})$. This enables us to use the hybrid method that combines polynomial approximation and double angle formula. Now, the condition for an input t changes to $t \in \cup_{i=-K+1}^{K-1} I_i$, where $I_i = [i - \frac{1}{4} - \epsilon, i - \frac{1}{4} + \epsilon]$.

4.1 Our Method

When a sufficiently smooth function is estimated by an interpolation polynomial, an error, difference between a real value and an estimated value, can be simply represented due to the following theorem.

Theorem 1 (polynomial interpolation). *Let f be a function in $C^{n+1}[a, b]$ and p_n be a polynomial of degree $\le n$ that interpolates the function f at $n + 1$ distinct points $t_0, t_1, \cdots, t_n \in [a, b]$, i.e. $p_n(t_i) = f(t_i)$ for all $0 \le i \le n$. Then, for each $t \in [a, b]$, there exists a point $\psi_t \in [a, b]$ such that*

$$f(t) - p_n(t) = \frac{f^{(n+1)}(\psi_t)}{(n+1)!} \cdot \prod_{i=0}^{n}(t - t_i). \tag{2}$$

Let $p_n(t)$ be the interpolation polynomial of degree $\le n$ that interpolates $\cos(2\pi t)$ at $n + 1$ distinct points. Then, the error bound between $\cos(2\pi t)$ and

$p_n(t)$ can be computed through Equation 2. Even though the term $\frac{f^{(n+1)}(\psi_t)}{(n+1)!}$ in Equation 2 is hard to be estimated exactly, it is bounded by the constant when f is $\cos(2\pi t)$. Thus, the error bound of polynomial approximation mainly depends on the other term

$$w(t) = \prod_{i=0}^{n}(t - t_i)$$

for pre-determined $t_0, t_1, \ldots, t_n \in [a, b]$ which are called *nodes*. Therefore, we need to choose $\{t_i\}_{1\le i\le n}$ appropriately to minimize the maximum value of $w(t)$ in a specified domain of t. In the case of the Chebyshev method, the nodes are chosen by $t_i = \frac{b+a}{2} + \frac{b-a}{2} \cdot \cos(\frac{2i-1}{2n+2}\pi)$ for $1 \le i \le n+1$ in the range $[a, b]$, and these nodes make the upper bound of $w(t)$ in the whole interval $\frac{1}{2^n} \cdot (\frac{b-a}{2})^{n+1}$ (which is $(\frac{b-a}{2})^{n+1}$ in the case of the Taylor approximation).

Although the Chebyshev method gives fairly good error bound in a global sense, it is not appropriate for our purpose because it does not consider the condition that t is near one of the points. Therefore, we focus on the bound of $w(t)$ for $t \in \cup_{i=-K+1}^{K-1} I_i$, where $I_i = [i - \frac{1}{4} - \epsilon, i - \frac{1}{4} + \epsilon]$, and propose a better method for this setting.

Our Optimized Nodes. We choose nodes as the Chebyshev method in each interval $I_i = [i - \frac{1}{4} - \epsilon, i - \frac{1}{4} + \epsilon]$ for all $-K < i < K$. More precisely, in the interval I_i, we choose d_i nodes $t_{i,j} = i - \frac{1}{4} + \epsilon \cdot \cos\left(\frac{2j-1}{2d_i}\pi\right)$ for $1 \le j \le d_i$. Let $n = \sum d_i - 1$ and p_n be the polynomial of degree $\le n$ that interpolates the function $\cos(2\pi t)$ at $n+1$ distinct points $t_{i,j}$ $(-K < i < K, 1 \le j \le d_i)$. In other words, p_n satisfies the following equation:

$$p_n(t_{i,j}) = \cos(2\pi t_{i,j}) \ \text{ for } \ -K < i < K, 1 \le j \le d_i$$

Then, as in Equation 1, we can deduce the following upper bound of $||w(t)||$:

$$||w(t)|| \le \frac{1}{2^{d_i-1}} \cdot \epsilon^{d_i} \cdot \prod_{j=1}^{K-1-i}(j+\epsilon)^{d_{i+j}} \cdot \prod_{j=1}^{K-1+i}(j+\epsilon)^{d_{i-j}},$$

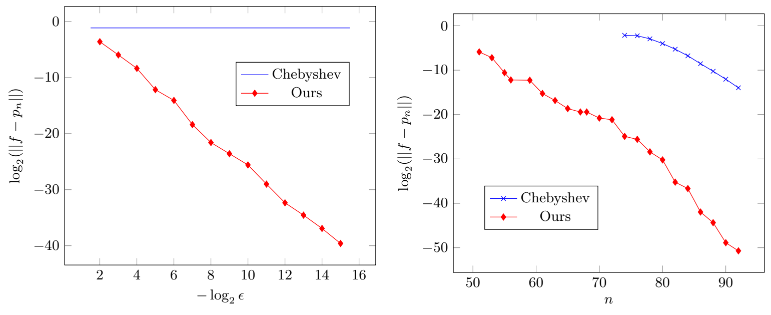
when $t \in I_i = [i - \frac{1}{4} - \epsilon, i - \frac{1}{4} + \epsilon]$. Therefore, $||\cos(2\pi t) - p_n(t)|| = \mathrm{O}(\epsilon^{d_i})$ on I_i, which means that the error bound decreases as ϵ, representing the ratio between the size of a message and the size of a cipertext modulus, gets smaller. In contrast, the error between $\cos(2\pi t)$ and $p_n(t)$ obtained from the Chebyshev method is bounded by $\frac{(2\pi)^{n+1}}{(n+1)!} \cdot \frac{K^{n+1}}{2^n}$, which is not affected by ϵ. In sum, let $M_i = \mathsf{Max}_{t\in I_i}||w(t)||$, then we obtain

$$||\cos(2\pi t) - p_n(t)|| \le \frac{(2\pi)^{n+1}}{(n+1)!} \cdot \mathsf{Max}\{M_{-K+1}, M_{-K+2}, \ldots, M_{K-1}\} \qquad (3)$$

for $t \in \cup_{i=-K+1}^{K-1} I_i$, where $I_i = [i - \frac{1}{4} - \epsilon, i - \frac{1}{4} + \epsilon]$.

How to choose $d_{-K+1}, \ldots, d_{K-1}$. For each integer i, we have to decide d_i, the number of nodes in the interval $I_i = [i - \frac{1}{4} - \epsilon, i - \frac{1}{4} + \epsilon]$, and it is done by the following algorithmical way. We first initialize $d_i = 1$ for all i. With these d_i's, we compute each M_i and find the index that has a maximum M_i value. Then, if $i_0 = \text{argmax} M_i$, we increase d_{i_0} by 1. We repeat this process until the total degree ($= \sum d_i$ - 1) becomes target degree.

Comparison. We conduct an experiment to compare our method and the Chebyshev method. We compare the experimental error bound of our method with that of the other method. Figure 2 shows experimental error bounds between the function $\cos(2\pi t)$ and its approximate polynomials obtained from each method. Figure 2(a) is obtained by fixing $n = 76$ and varying ϵ and Figure 2(b) is obtained by fixing $\log_2 \epsilon = -10$ and varying n. Since we bound the term $\frac{f^{(n+1)}(\psi_t)}{(n+1)!}$ in Equation 2 by a constant, an exact error bound is smaller than the theoretical error bound explained above.



(a) Fix $n = 76$ and vary ϵ (b) Fix $\log_2 \epsilon = -10$ and vary n

Fig. 2. Error bounds $\log_2(||f - p_n||)$ for our optimized interpolation ($K = 12$).

As seen in Figure 2, our method far outperforms the Chebyshev method. For fixed $n = 76$, for example, $\log_2$ value of the experimental error bound of ours is about -25.6 when $\log_2 \epsilon = -10$, but that of the Chebyshev method is about -1.1 independent of ϵ. Also, for fixed $\log_2 \epsilon = -10$, the degree of an interpolation polynomial of the Chebyshev method needs to be greater than 103 to yield the same level of the experimental error bound as the interpolation polynomial of degree 76 of ours.

4.2 Homomorphic evaluation of $p_n(t)$

After we get an approximate polynomial of degree $\leq n$ using our optimized method, we need to evaluate a value of the function at t in a homomorphic way. Naive approach is to compute t^i for all $i \in \{0, 1, \ldots, n\}$ first and then evaluate $p_n(t) = \sum_{i=0}^{n} p_i \cdot t^i$. Unfortunately, due to the un-stability of the coefficients, this way of computation not only can yield a lot of numerical errors but also make homomorphic evaluation difficult. Especially, extremely small p_i values make homomorphic evaluation inept since we need to multiply a huge modulus to encrypt these values. To avoid this problem, we represent the polynomial with the Chebyshev basis instead of t^i's.

The Chebyshev polynomials T_i's on $[-1, 1]$ are defined recursively by

$$T_0(t) = 1, T_1(t) = t,$$
$$T_i(t) = 2tT_{i-1}(t) - T_{i-2}(t) \text{ for } i \geq 2$$

Then, T_i satisfies the equation $T_i(\cos\theta) = \cos(i\theta)$ for all θ, thus $|T_i(t)| \leq 1$ for all $|t| \leq 1$. Since the domain of our approximate polynomial is $[-K, K]$, we use $\tilde{T}_i(t) = T_i(\frac{t}{K})$ instead of T_i for each i. Note that $|\tilde{T}_i(t)| \leq 1$ for all $|t| \leq K$.

Since each $\tilde{T}_i$ is a polynomial of degree i, $\{\tilde{T}_i\}_{i=0}^n$ forms a basis for the vector space of polynomials of degree $\leq n$. Hence, $p_n(t)$ can be represented by a linear combination of $\{\tilde{T}_i\}_{i=1}^n$ as $p_n(t) = \sum_{i=0}^n c_i \cdot \tilde{T}_i(t)$ for some $c_0, \cdots, c_n \in \mathbb{R}$. Well, these c_i values also can be un-stable as in the case of the original t^i basis representation. However, since $|\tilde{T}_i(t)| \leq 1$ for all $|t| \leq K$, the term $c_i \cdot \tilde{T}_i(t)$ with extremely small c_i has little effect on the value of $p_n(t)$. Therefore, we can simply ignore the term having extremely small c_i, which not only causes numerical errors but also makes the homomorphic evaluation inefficient.

Algorithm 1 Baby-step Giant-step algorithm

```
1: Input : A polynomial of degree $n$, $p = \sum_{i=0}^n c_i T_i$.
2: Let $m$ be the smallest integer satisfying $2^m > n$ and $l \approx m/2$.
3: Evaluate $T_2(t), T_3(t) \cdots, T_{2^l}(t)$ inductively.
4: Evaluate $T_{2^{l+1}}(t) \cdots, T_{2^{m-1}}(t)$ using the equation $T_{2i}(t) = 2T_i(t)^2 - 1$.
5: Find polynomials $q$ and $r$ of degree $< 2^{m-1}$ which satisfy $p = q \cdot T_{2^{m-1}} + r$ in forms
   of a linear combination of Chebyshev basis.
6: Evaluate $q(t)$ and $r(t)$ recursively. (Repeat 5 with $p$ being replaced by $q$ and $r$ until
   the degree of a quotient and a remainder become smaller than $2^l$)
7: Evaluate $p(t)$ with $q(t)$, $r(t)$ and $T_{2^{m-1}}(t)$.
8: Output : $p(t)$
```

Next, we can use the Baby-step Giant-step algorithm (Algorithm 1) to evaluate the polynomial $p_n(t)$. This algorithm enables us to evaluate $p_n(t)$ in $2\sqrt{2n} + \frac{1}{2}\log_2 n + \mathrm{O}(1)$ non-scalar multiplications and $\lceil \log_2 n \rceil$ depth consumption. More precisely, with m the smallest integer satisfying $2^m > n$ and $l \approx m/2$, we can evaluate $p_n(x)$ with $2^l + 2^{m-l} + m - l - 3$ non-scalar multiplications while consuming m depth.

Also, we can use the Paterson-Stockmeyer algorithm for Chebyshev polynomials suggested in [5]. In [5], authors modify the original Paterson-Stockmeyer algorithm [19] to evaluate polynomials represented in the Chebyshev basis. They propose an algorithm that enables evaluating a polynomial of degree n represented in the Chebyshev basis with $\sqrt{2n} + \log_2 n + \mathrm{O}(1)$ non-scalar multiplications, see [5] for more details. By using this algorithm, we can evaluate $p_n(t)$

with $\sqrt{2n} + \log_2 n + \mathrm{O}(1)$ non-scalar multiplications while consuming $\lceil \log_2 n \rceil$ depth. More precisely, with $k \approx \sqrt{n/2}$ and the smallest integer m satisfying $(2^m - 1)k > n$, we can evaluate $p_n(t)$ with $2^{m-1} + 2m + k - 4$ non-scalar multiplications while consuming $\lceil \log_2 k \rceil + m$ depth.

Even though the Paterson-Stockmeyer algorithm is asymptotically better than the Baby-step Giant-step algorithm, it does not mean that the Paterson-Stockmeyer algorithm outperforms the Baby-step Giant-step algorithm in a practical sense. The degree of an approximate polynomial that we use is not so big in practical, and we can reduce it further by using the hybrid method which will be introduced in the next section. Moreover, when the degree n is small, the effect of the term $\log_2 n$ becomes greater, especially in the Paterson-Stockmeyer algorithm. In fact, the Baby-step Giant-step algorithm shows the better performance based on our experiment when degree n is small. For these reasons, we use the Baby-step Giant-step algorithm instead of the Paterson-Stockmeyer algorithm.

4.3 Hybrid Method

Recall that, in [7], authors scale down an input t by a power of two and use double-angle formula to make it locate close enough to the origin before they use the Taylor approximation. We can also apply this idea to our method simply by using double angle formula of cosine[8].

Suppose we scale down t by 2^r before using our method and let $t' = t/2^r$. We say the number of scaling is r in this case. Then, we need to approximate $\cos(2\pi t')$ based on the fact that t' is contained in one of the intervals $\tilde{I}_i = [\frac{1}{2^r}(i - \frac{1}{4} - \epsilon), \frac{1}{2^r}(i - \frac{1}{4} + \epsilon)]$. Naturally, we choose d_i nodes in each interval $\tilde{I}_i$ by $\tilde{t}_{i,j} = \frac{1}{2^r}\left(i - \frac{1}{4} + \epsilon \cdot \cos\left(\frac{2j-1}{2d_i}\pi\right)\right)$ for $1 \leq j \leq d_i$ and $|i| < K$ to apply our method. Then, we can see from Equation 2 that all the terms in $w(t) = \prod_{i=0}^{n}(t - t_i)$ decrease by a factor of 2^r compared to before, and thus degree n can be smaller while ensuring the same level of error bounds as before. However, since the other term $1/(n+1)!$ is difficult to predict how it changes as n varies, it is hard to predict an exact value of degree that yields the same level of error bounds as the method without scaling.

We conduct an experiment to find the degree that gives the same level of error bound as the original method (the method without scaling) for each number of scaling. The result of the experiment is given in Table 3. The first column indicates the degree of approximate polynomials from the original method and the other columns represents the minimum degree of approximate polynomials that ensure the same level of error bounds for each number of scaling and corresponding depth consumption. Note that we need to consume $\lceil \log_2 n \rceil$ detph to evaluate $p_n(t)$ in a homomorphic way and require r more depth for double angle formula for a cosine function when the number of scaling is r. As our expectation, the degree of an approximate polynomial gradually decreases as

[8] Previous method uses a sine function and double angle formula for a sine function needs both $\cos(t)$ and $\sin(t)$ to compute $\sin(2t)$.

Degree	Depth	# of scaling					
		1		2		3	
		Degree	Depth	Degree	Depth	Degree	Depth
76	7	49	6+1	**31**	**5+2**	24	5+3
86	7	**57**	**6+1**	40	6+2	28	5+3
96	7	65	7+1	45	6+2	34	6+3
106	7	72	7+1	51	6+2	38	6+3
116	7	80	7+1	57	6+2	43	6+3
126	7	88	7+1	63	6+2	49	6+3
136	8	**94**	**7+1**	70	7+2	55	6+3

Table 3. Minimum degree of an approximate polynomials to ensure the same level of error bound for each number of scaling and corresponding depth consumption. ($K = 12$ and $\log_2 \epsilon = -10$)

the number of scaling increases. However, even the coefficients of the Chebyshev basis representation become more stabilized and the number of non-scalar multiplications decreases, it does not necessarily mean that the scaling is always favorable because depth consumption can increase.

In the case of degree 76, if we scale by a factor of 4, we can compute the approximate polynomial with depth consumption 7 as the original while the number of non-scalar multiplications based on Algorithm 1 decreases from 24 to 13. Therefore, the scaling is unconditionally favorable in this case. However, in the case of degree 116, if we scale by a factor of 2, even the number of non-scalar multiplications decreases, we need to consume one more depth compared to the original method. Therefore, in this case, we need to consider the trade-off between the number of non-scalar multiplications and the depth consumption. In conclusion, we need to conduct an experiment to decide whether we use the hybrid method or not, and decision will depend on the trade-off between the number of non-scalar multiplications and the depth consumption[9].

4.4 Overall Comparison

In this section, we compare our method with the previous work [5]. In [5], authors use the approximate polynomial of degree 119 obtained from the Chebyshev method.[10] In our case, we choose degree 74 because the approximate polynomial of degree 74 obtained from our method gives the same level of error as the inevitable error occurs between t and $\frac{1}{2\pi}\sin(2\pi t)$ under the parameter sets that used by the previous work [5]. The comparison between our method and the previous work [5] is given in Table 4. As seen from the table, by using the hybrid method with the number of scaling 2 and evaluating the obtained approximate polynomial of degree 30 with Algorithm 1, we can decrease the number of non-scalar multiplications almost by half while consuming the same depth.

[9] The code for finding an approximate polynomial for the cosine function can be found at [15].

[10] In fact, they use the nodes $t_i = K\cos(i\pi/n)$ for $0 \le i \le n$ instead of nodes $t_i = K\cos((2i-1)\pi/(2n+2))$ for $1 \le i \le n+1$. But, there is no big difference.

Method	Degree	# of Scaling	Degree (After scaling)	Non-scalar Multiplication	Depth
Ours	74	0	74	24 (Alg 1)	7
		1	49	16+1 (Alg 1)	6+1
		2	**30**	**11+2 (Alg 1)**	**5+2**
[5]	119	-	-	20 (PS alg)	7

Table 4. Comparison between our method and the previous work. [5] ($K = 12$ and $\log_2 \epsilon = -10$)

4.5 Put Everything Together

Using the scheme in Section 3 and the above sine evaluation method, we can do a better bootstrapping for Full-RNS variant of `HEAAN`. Recall that the bootstrapping can be divided into four steps: modular raising, linear transformation, sine evaluation, and inverse linear transformation. For linear transformation and the inverse part, we used the techniques proposed in [12, 5].

Sine Evaluation. To approximate a sine function, we use the polynomial interpolation method with our optimized nodes, which performs better than the other methods because an input for a sine function is restricted to some small intervals. Also, we further improve it by using the hybrid method, which combines our method with double angle formula of cosine, and it can decrease the number of non-scalar multiplications a lot while consuming the same depth.

With the approximate polynomial obtained from our method, we evaluate it with the Baby-step Giant-step algorithm, which shows better performance when the degree of an approximate polynomial is small. In our implementation, we fix $\log_2 \epsilon = -10$ and use the approximate polynomial of degree 30 obtained from the hybrid method with the number of scaling 2.

Scaling Factor Control. After each homomorphic multiplication and rescaling, the scaling factor changes. Therefore, the bootstrapping process can also change the scaling factor, and this can be a problem when we want to do some operations with an output of the bootstrapping and fresh ciphertexts. Even though they are in the same level, their scaling factor can be different, and thus, for example, homomorphic addition between these two ciphertexts can yield an un-expected result.

To solve this problem, at the last step of the bootstrapping, evaluation, we multiply Δ' and perform $\mathtt{RS}(\cdot)$ with a constant Δ' for which $(\Delta' \cdot \Delta)/q_{L'} = \Delta_{L'-1}$, where $\Delta_{L'-1}$ is the scaling factor at level $L' - 1$ and Δ is the current scaling factor. This requires one level consumption, but we can optimize it by merging this process with the last step of linear transformation in the bootstrapping.

5 Implementation

We implement our full-RNS variant of HEAAN in Section 3 and its bootstrapping. The experiments are conducted in PC with Intel(R) Core(TM) i9-9820X CPU @ 3.30GHz using single-thread.

5.1 Performance of basic homomorphic operations.

First, the performance of the basic homomorphic operations is given in Table 5. The results again show why it is better to use a proper `dnum`. By using $1 < \texttt{dnum} < L+1$, we can reduce the complexity of the homomorphic multiplication. For example, in Table 5, the best timing result is almost two times faster than the $\texttt{dnum} = 24$ case which is used in the `SEAL` library. In addition, the first row ($\texttt{dnum} = 1$), which corresponds to the `HEAAN-RNS` scheme, not only does not satisfy the 128-bit security condition based on the Martin's LWE estimator [1], but also gives a 1.8 times slower result compared to the second row[11].

	$\log q_i$	L	`dnum`	`Enc`	`Dec`	`Mult`	`Rescale`	
$N = 2^{16}$	45	23	1	103 ms	5 ms	773 ms	60 ms	`HEAAN-RNS`
			4			**436 ms**		
			6			487 ms		
			12			660 ms		
			24			958 ms		`SEAL` v3.3

Table 5. Performance of our Full-RNS variant of `HEAAN` with 2^{15} slots.

5.2 Bootstrapping Performance.

For an experiment on the bootstrapping, we set two parameter sets as in Table 6 using the Martin's LWE estimator. The bit size of each prime in modulus chain is set to 40 for `Param 1` and 45 for `Param 2`, and $\log q_0$ is 50 and 55, respectively. In the case of `Param 1`, we used large $\texttt{dnum} = 10$ to make the security parameter > 100. This enables us to use $\log_2 N = 15$ which should be 16 if we use $\texttt{dnum} = 1$. In our experiment, we use the approximate polynomial of degree 30 with 2 times of scaling for sine evaluation (See 3rd parameters of our method in Table 4).

	L	`dnum`	N	$\log Q$	$\log Q + \log P$	Security
`Param 1`	19	10	32768	810	910	110.4
`Param 2`	27	7	65536	1270	1452	127.2

Table 6. Parameter sets

Using those two parameter sets, we ran our bootstrapping with various number of slots `ns`. Here, `Amortized Time` indicates bootstrapping time per each slot. Because of computational errors generated in the linear transformation part, large `ns` implies lower precision. Here, the precision means $-\log_2 e$, where e is average noise generated through the bootstrapping. For example, precision 15.5 in the first row in Table 7 means that noise with average size $2^{-15.5}$ is added through the bootstrapping.

[11] Here, `SEAL` v.3.3 and `HEAAN-RNS` indicate the schemes corresponding to each library and paper

	`ns`	`Boot Time`	`Precision`	`After Level`	`Amortized Time`
`Param 1`	2^0	6.8 s	15.5	5	7.1 s
	2^1	7.0 s	16.8	3	3.5 s
	2^2	7.5 s	15.0	3	1.87 s
`Param 2`	2^5	28 s	18.5	9	0.87 s
	2^{10}	37.6 s	15.3	7	0.036 s
	2^{14}	52.8 s	10.8	7	0.0032 s

Table 7. Performance of the bootstrapping in our scheme

In our experiment, we used fixed $\epsilon = 2^{-10}$. Because of the difference between t and $\sin t$ [12], the maximum precision for the bootstrapping is limited to $\simeq \epsilon^2 = 2^{-20}$. From the `ns` $= 2^5$ case with `Param 2`, which ensures 18.5 precision, we can see that sine evaluation with our method yields an accurate enough result.

Comparison. The last row of Table 4 in [5] and Table 2 in [12] use similar parameter sets to the last row of Table 7. Timing results were 158 seconds in [5] and 127 seconds in [12], which is just 52.8 seconds in our experiment. In other words, using the Full-RNS variant with proper `dnum` and the improved method for sine approximation gives a 3 and 2.5 times faster result than the previous works, respectively.

6 Conclusion

In this work, we suggest the generalized key-switching method for Full-RNS variant of `HEAAN` and propose a better method for approximating a sine function. With these improvements, we increase the efficiency of the bootstrapping for Full-RNS variant of `HEAAN`. Our method of approximating a sine function is specialized in the setting when inputs for a function are restricted to union of small intervals. Hence, we can also apply our method effectively to another functions which has a restricted domain as in the case of the bootstrapping for `HEAAN`.

So far, the research on approximating a modulus function is based on a sine function. Therefore, it has the limitation because of the inevitable error generated from the approximation of $[t]_q$ with $\frac{1}{2\pi}\sin(2\pi t)$. We expect that we can overcome this limitation by finding another approximation of $[\cdot]_q$ operation. We think it can be a new breakthrough of improving the bootstrapping.

References

1. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
2. Bajard, J.C., Eynard, J., Hasan, M.A., Zucca, V.: A full RNS variant of FV like somewhat homomorphic encryption schemes. In: International Conference on Selected Areas in Cryptography. pp. 423–442. Springer (2016)

[12] $|t - \sin t| < O(t^3)$ for t near the origin.

3. Boemer, F., Lao, Y., Wierzynski, C.: nGraph-HE: A Graph Compiler for Deep Learning on Homomorphically Encrypted Data. arXiv preprint arXiv:1810.10121 (2018)
4. Carpov, S., Gama, N., Georgieva, M., Troncoso-Pastoriza, J.R.: Privacy-preserving semi-parallel logistic regression training with Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2019/101 (2019), https://eprint.iacr.org/2019/101
5. Chen, H., Chillotti, I., Song, Y.: Improved bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 34–54. Springer (2019)
6. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: A full RNS variant of approximate homomorphic encryption. In: International Conference on Selected Areas in Cryptography. pp. 347–368. Springer (2018)
7. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 360–384. Springer (2018)
8. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 409–437. Springer (2017)
9. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC. vol. 9, pp. 169–178 (2009)
10. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: CRYPTO 2012, pp. 850–867. Springer (2012)
11. Halevi, S., Polyakov, Y., Shoup, V.: An improved RNS variant of the BFV homomorphic encryption scheme. In: Cryptographers Track at the RSA Conference. pp. 83–105. Springer (2019)
12. Han, K., Hhan, M., Cheon, J.H.: Improved Homomorphic Discrete Fourier Transforms and FHE Bootstrapping. IEEE Access (2019)
13. Han, K., Hong, S., Cheon, J.H., Park, D.: Efficient logistic regression on large encrypted data. Cryptology ePrint Archive, Report 2018/662 (2018)
14. Jiang, Y., Wang, C., Wu, Z., Du, X., Wang, S.: Privacy-preserving biomedical data dissemination via a hybrid approach. In: AMIA Annual Symposium Proceedings. vol. 2018, p. 1176. American Medical Informatics Association (2018)
15. Ki, D.: https://github.com/DohyeongKi/better-homomorphic-sine-evaluation (2019)
16. Kim, A., Song, Y., Kim, M., Lee, K., Cheon, J.H.: Logistic regression model training based on the approximate homomorphic encryption. BMC medical genomics **11**(4), 83 (2018)
17. Kim, M., Song, Y., Li, B., Micciancio, D.: Semi-parallel Logistic Regression for GWAS on Encrypted Data. Cryptology ePrint Archive, Report 2019/294 (2019), https://eprint.iacr.org/2019/294
18. Kim, M., Song, Y., Wang, S., Xia, Y., Jiang, X.: Secure logistic regression based on homomorphic encryption: Design and evaluation. JMIR medical informatics **6**(2), e19 (2018)
19. Paterson, M.S., Stockmeyer, L.J.: On the number of nonscalar multiplications necessary to evaluate polynomials. SIAM Journal on Computing **2**(1), 60–66 (1973)

A Correctness and Noise Growth of Homomorphic Multiplication

Before proving the correctness of the homomorphic multiplication, first remind the properties of `ModUp` and `ModDown` with the following three equations:

$$\|\mathtt{CRT}_{\mathcal{C}\cup\mathcal{B}}(\mathtt{ModUp}([a(x)]_{q_0},[a(x)]_{q_1},\ldots,[a(x)]_{q_\ell}))\|_\infty \le (\ell+1)\cdot Q, \tag{A.1}$$

$$\mathtt{CRT}_{\mathcal{C}\cup\mathcal{B}}(\mathtt{ModUp}([a(x)]_{q_0},[a(x)]_{q_1},\ldots,[a(x)]_{q_\ell})) \equiv a(x) \bmod Q, \tag{A.2}$$

$$\left\|\mathtt{CRT}_{\mathcal{C}}(\mathtt{ModDown}([a(x)]_{q_0},\ldots,[a(x)]_{q_\ell},[a(x)]_{p_0},\ldots,[a(x)]_{p_{k-1}})) - \left\lceil\frac{a(x)}{P}\right\rfloor\right\|_\infty < k, \tag{A.3}$$

where $\mathcal{B}=\{p_0,\ldots,p_{k-1}\}$ and $\mathcal{C}=\{q_0,\ldots,q_\ell\}$. With the above three equations and properties of `RNS-Decompose` and `RNS-Power`, we can prove the correctness of the homomorphic multiplication in our scheme.

Theorem 2. *The algorithm* $\underline{\mathtt{Mult}_{\mathsf{evk}}(\mathsf{ct}_0,\mathsf{ct}_1)}$ *returns* $(b_3(x),a_3(x))$ *such that*

$$b_3(x)+a_3(x)\cdot s(x) = M_1(x)\cdot M_2(x) + M_0(x)\cdot e_1(x) + M_1(x)\cdot e_0(x) + \epsilon(x),$$

where $\|\epsilon(x)\|_\infty < \|s(x)\|_1$*, when* $P > 2\beta N e_{\mathit{fresh}}\cdot(\max_{0\le i<\beta} Q_i)$*. Here,* $\mathsf{ct}_i = (b_i(x),a_i(x)) \in R_Q^2$*, and* $b_i(x)+a_i(x)\cdot s(x) = M_i(x)+e_i(x)$ *for* $i=0,1$.

Proof. *For simplicity, we assume that* $\ell = L$ *and* $(\ell+1)$ *is a multiple of* α*. First, a vector* $(d_0(x),d_1(x),d_2(x))$ *which satisfies*

$$\begin{aligned} &d_0(x)+d_1(x)\cdot s(x)+d_2(x)\cdot s(x)^2 = (M_0(x)+e_0(x))\cdot(M_1(x)+e_0(x)) \\ &= M_0(x)\cdot M_1(x)+M_0(x)\cdot e_1(x)+M_1(x)\cdot e_0(x)+e_0(x)\cdot e_1(x) \\ &= M_0(x)\cdot M_1(x)+e_2(x)\in R_Q. \end{aligned}$$

is obtained after Step 1.

In Step 2, since $\ell = L$ *and* $(L+1)$ *is a multiple of* α*,* β *equals to* $\mathtt{dnum}$ *and the zero-padding part can be omitted. Then,*

$$([d_2(x)\cdot\hat{Q}_0^{-1}]_{Q_0},\ldots,[d_2(x)\cdot\hat{Q}_{\mathtt{dnum}-1}^{-1}]_{Q_{\mathtt{dnum}-1}}) = \mathtt{RNS\text{-}Decomp}_{\mathcal{C}'}(d_2(x))$$

is returned after RNS-Decompose step.

Also, Modulus-Raise step returns vectors of length $k+\ell+1$*,*

$$\begin{aligned} &([\tilde{d}_2^{(i)}(x)]_{q_0},\ldots,[\tilde{d}_2^{(i)}(x)]_{q_\ell},[\tilde{d}_2^{(i)}(x)]_{p_0},\ldots,[\tilde{d}_2^{(i)}(x)]_{p_{k-1}}) \\ &= \mathtt{ModUp}_{\mathcal{C}_i\to\mathcal{C}\cup\mathcal{B}}([d_2(x)\cdot\hat{Q}_i^{-1}]_{q_{i\alpha}},\ldots,[d_2(x)\cdot\hat{Q}_i^{-1}]_{q_{(i+1)\alpha-1}}), \end{aligned}$$

where $\tilde{d}_2^{(i)}(x)\in R_{PQ}$*, for* $0\le i<\mathtt{dnum}$*. From Equation A.1 - A.2, we can check that* $\tilde{d}_2(x)$ *satisfies the following equations:*

$$\tilde{d}_2^{(i)}(x) \equiv d_2(x)\cdot\hat{Q}_i^{-1} \bmod Q_i \text{ and } \left\|\tilde{d}_2^{(i)}(x)\right\|_\infty \le (\alpha+1)\cdot Q_i. \tag{A.4}$$

Note that the norm of $\tilde{d_2}^{(i)}(x)$ is still much smaller than PQ, and for this reason, `ModUp` *does not harm the functionality of* ***RNS-Decompose*** *and* ***RNS-Power****.*

Next, we suppose that evaluation keys $\mathsf{evk}_i = (B_i(x), A_i(x)) \in R^2_{PQ}$ which satisfy $B_i(x) + A_i(x) \cdot s(x) = P \cdot \hat{Q}_i \cdot s^2(x) + E_i(x) \in R_{PQ}$, where $\|E_i(x)\|_\infty < e_{\mathsf{fresh}}$, are generated in the key generation step. Then, the inner product step returns $(B'(x), A'(x)) = \sum_{i=0}^{\beta-1} \left[\tilde{d_2}^{(i)}(x) \cdot (B_i(x), A_i(x))\right]$ and it satisfies the following equation:

$$\begin{aligned} B'(x) + A'(x) \cdot s(x) &= P \sum_{i=0}^{\beta-1} \left(\tilde{d_2}^{(i)}(x) \cdot \hat{Q}_i \cdot s^2(x)\right) + \sum_{i=0}^{\beta-1} \left(\tilde{d_2}^{(i)}(x) \cdot E_i(x)\right) \\ &= P \cdot d_2(x) \cdot s^2(x) + E'(x) \in R_{PQ}, \end{aligned}$$

where $\|E'(x)\|_\infty < N \sum_{0 \le i < \beta} Q_i \cdot e_{\mathsf{fresh}} \le N \cdot \beta \cdot e_{\mathsf{fresh}} \cdot (\max_{0 \le i < \beta} Q_i)$ and N is the dimension of the ring.

After that, we apply modulus-down process to revert the modulus space from R_{PQ} to R_Q and to reduce the size of $E'(x)$. Let $(\tilde{B}(x), \tilde{A}(x))$ be the return of modulus-down step with CRT decomposed representation. From the modulus switching technique and Equation A.3, we can see that $(\tilde{B}(x), \tilde{A}(x))$ has the following property:

$$\tilde{B}(x) + \tilde{A}(x) \cdot s(x) = d_2(x) \cdot s^2(x) + \left\lfloor \frac{E'(x)}{P} \right\rceil + \epsilon(x) \in R_Q,$$

where $\|\epsilon(x)\|_\infty < \|s(x)\|_1$. Since $P > 2 \cdot N \cdot \beta \cdot e_{\mathsf{fresh}} \cdot (\max_{0 \le i < \beta} Q_i)$, each coefficient of $E'(x)/P$ is in the range $(-0.5, 0.5)$, and thus rounding of the polynomial becomes a zero polynomial. Therefore, it follows that $\tilde{B}(x) + \tilde{A}(x) \cdot s(x) = d_2(x) \cdot s(x)^2 + \epsilon(x) \in R_Q$.

At the last step, we compute and return $(b_3(x), a_3(x)) = (d_0(x) + \tilde{B}(x), d_1(x) + \tilde{A}(x))$. Then, from the equation

$$\begin{aligned} b_3(x) + a_3(x) \cdot s(x) &= d_0(x) + d_1(x) \cdot s(x) + d_2(x) \cdot s(x)^2 + \epsilon(x) \\ &= M_0(x) \cdot M_1(x) + e_2(x) + \epsilon(x), \end{aligned}$$

the correctness of homomorphic multiplication is followed. Furthermore, the size of the noise after multiplication is given by $M_0(x) \cdot e_1(x) + M_1(x) \cdot e_0(x) + e_0(x) \cdot e_1(x) + \epsilon(x)$, where $\|\epsilon(x)\|_\infty < \|s(x)\|_1$. □