Better Bootstrapping for Approximate Homomorphic Encryption

Kyoohyung Han, Dohyeong Ki

Seoul National University, Seoul 08826, South Korea
{satanigh,wooki7098}@snu.ac.kr
https://sites.google.com/view/kyoohyunghan

Abstract. After Cheon et al. (Asiacrypt' 17) proposed approximate homomorphic encryption for operations between encrypted real (or complex) numbers, this scheme is widely used in various fields with the needs on privacy-preserving in data analysis. After that, the bootstrapping method is firstly proposed by Cheon et al. (Eurocrypt' 18) by replacing modulus reduction with sine function. In this paper, we generalize Full-RNS variant of HEAAN scheme to reduce the number of special primes which are used in key-switching. As a result, our scheme can use a smaller ring dimension or supports more depth computation without bootstrapping while preserving the same security level. And, we propose a bootstrapping specified polynomial approximation method to evaluate sine function in an encrypted state. In our method, the degree of a polynomial approximation is related to the plaintext size. This gives a smaller number of non-scalar multiplications which is about half of the previous work. With our variant of Full-RNS scheme and new sine evaluation method, we firstly implement bootstrapping on Full-RNS variant of approximate homomorphic encryption. Our implementation shows that bootstrapping takes about 120 seconds with 19-bit precisions.

Keywords: Homomorphic Encryption · Bootstrapping · Polynomial Approximation.

1 Introduction

After the Gentry's first blueprint for fully homomorphic encryption scheme [9], homomorphic encryption is regarded as one of the important tools for privacy preserving. In various applications that need privacy protection, homomorphic operations between encrypted real number data is necessary. In 2017, Cheon et al. proposed a new homomorphic encryption scheme for efficient operations between real number data, which is called HEAAN [8]. From homomorphic operations on encrypted real numbers, various applications such as logistic regression and GWAS can be done in encrypted state [4, 14, 11, 13, 15, 17, 16].

Recently, a lot of data analysis tools and methods come out into the world and they become more and more complicated. For example, machine-learning algorithms such as convolutional neural network (CNN) and deep neural network

(DNN) are extremely complicated. Therefore, modern data analysis algorithms require huge depth, which makes them hard to display their all ability in encrypted state without bootstrapping, since only a limited number of levels can be provided by homomorphic encryption schemes. For example, *nGraph-HE* [3], which is a deep-learning prediction on encrypted data, does not use bootstrapping, hence it only supports a limited number of layers. Also, in the case of logistic regression, solutions without bootstrapping [15, 17] only support small number of iterations. For these reasons, the importance and the necessity of efficient bootstrapping for homomorphic encryption schemes becomes greater nowadays.

The first bootstrapping for `HEAAN` is proposed by Cheon et al. [7] and the improved methods are suggested in [5, 12]. However, there is a still room for improvements for the bootstrapping for `HEAAN`. First of all, the variant of `HEAAN` can be used instead of `HEAAN`. Recently, Cheon et al. [6] propose a Full-RNS variant of `HEAAN` which is characterized by that all variables are fully in residue number systems in the scheme. This scheme enables all computations in homomorphic operations to be done in small arithmetic, which makes it far faster than the original `HEAAN`. In this paper, to make bootstrapping more practical, we generalize the Full-RNS variant of `HEAAN` and improve sine evaluation for the bootstrapping method. At the same time, by putting togther, we firstly implement Full-RNS variant of `HEAAN` and its bootstrapping.

1.1 Our Contribution

- We suggest generalized key-switching method for Full-RNS variant of `HEAAN` scheme. We combine RNS-decomposition method in [2] and temporary special modulus technique in [10]. As a result, with the same security level, we can support more depth computation without bootstrapping or reduce the ring dimension which gives improvements in overall performance.
- Better way for sine evaluation: we firstly propose the method to use message size in sine evaluation. It means that we evaluate sine function by considering the ratio between ciphertext modulus for input and message size. As a result, our method only requires $\mathsf{Max}(\log K+3+\frac{1}{K}(\log\epsilon-1), \log\log q)$ levels for ratio between message and modulus space of input ciphertext ϵ. Furthermore, by using cosine instead of sine, we combine double-angle formula for cosine with our approximation method. As a result, we reduce the number of non-scalar multiplications almost by a half compare to the previous work [5].
- We put every technique together, and firstly implement bootstrapping on our Full-RNS variant of `HEAAN` scheme. As a result, our bootstrapping shows 200 seconds for plaintext space $\mathbb{C}^{4096}$ with 15 bit precision.

1.2 Related Works

In [6], Full-RNS variant `HEAAN` is proposed. Their scheme works on residue number system fully, and this leads to huge performance improvement. After that,

SEAL includes implementation of this scheme in version 3.0. To avoid special prime usage, they combined bit-decomposition and RNS-decomposition technique. In case of [16], they used one special modulus with RNS-decomposition instead of using bit-decomposition.

In [7], they firstly proposed bootstrapping method for approximate homomorphic encryption. They used sine function to represent modulus reduction with arithmetic on $\mathbb{R}$ because $x - \mathsf{round}(x)$ has similar value with $\frac{1}{2\pi}\sin(2\pi x)$ when the input x is close to some integer. After that, to evaluate $\sin(2\pi x)$, they evaluate $\sin(2\pi x/2^r)$ for $x \in (-K, K)$. Because $2\pi x/2^r$ is in a small range for large enough r, they simply use the Taylor approximation with small degree which gives good approximation near the origin point. And, by using double-angle formula, they expand the value to $\frac{1}{2\pi}\sin(2\pi x)$. This method gives small number of homomorphic multiplications, but it requires large degree $O(\log Kq)$ for the input ciphertext modulus q.

In [5], they used the Chebyshev approximation without double angle formula. They just find proper polynomial in global range $(-K, K)$. They also represent method for polynomial evaluation by revising the Paterson Stockmeyer algorithm for representation using the Chebyshev polynomial.

In [12, 5], they proposed a new method for evaluation of linear transformation in bootstrapping. They decompose the linear transformation to multiple number of layers. From the special structure of the transformation, the new method gives $O(\log \ell)$ complexity which was $O(\ell)$ in [7] for the length ℓ of plaintext vector.

1.3 Road Map

In Section 2, we briefly introduce polynomial approximation and HEAAN-RNS scheme with its fast base conversion. In Section 3, we discuss generalized key-switching method for Full-RNS variant of HEAAN. In Section 4, we propose better method for approximating sine function for the bootstrapping. In Section 5, we put everything together and implement our full-RNS variant of HEAAN scheme and its bootstrapping, and analyze results of our experiments. We complete the paper with suggesting future work for improving the bootstrapping.

2 Preliminary

2.1 Polynomial Approximation

Taylor Approximation. For the given range $[a, b]$ and $n > 0$, the Taylor approximation with respect to the middle point $\alpha = \frac{b+a}{2}$ is to find polynomial $p_n(x)$ such that

$$p_n(\alpha) = f(\alpha), \frac{d^k p_n(\alpha)}{dx^k} = \frac{d^k f(\alpha)}{dx^k} \text{ for } 1 \le k \le n$$

For this polynomial, following equation holds

$$||f(x) - p_n(x)|| \le \frac{|f^{(n+1)}(\psi_x)|}{(n+1)!} \cdot \left(\frac{b-a}{2}\right)^{n+1}$$

for some $\psi_x \in [a,b]$.

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

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

for $1 \le i \le n+1$. For those points, Chebyshev approximation of $f(x)$ is to find degree n polynomial $p_n(x)$ such that $p_n(x_i) = f(x_i)$ for all $1 \le i \le n+1$. For this polynomial, from the property of Chebyshev points, following equation holds

$$||f(x) - p_n(x)|| \le \frac{|f^{(n+1)}(\psi_x)|}{(n+1)!} \cdot \frac{1}{2^n} \cdot \left(\frac{b-a}{2}\right)^{n+1} \quad (1)$$

for some $\psi_x \in [a,b]$. Note that additional term $\frac{1}{2^n}$ makes the Chebyshev method much better than that of Taylor when the degree is large.

2.2 Full-RNS HEAAN

In this section, we introduce fast base conversion in [2] and `HEAAN-RNS` scheme in [6]. Using fast base conversion that does not require CRT composition, ciphertext of `HEAAN-RNS` scheme can stay in residue number system (RNS).

Fast Base Conversion. In `HEAAN-RNS` scheme, it converts the 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 $a \in \mathbb{Z}_Q$ into an element of $\mathbb{Z}_{p_0} \times \cdots \times \mathbb{Z}_{p_{k-1}}$ by computing

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

where $\hat{q}_j = \prod_{j'\ne j} q_{j'} \in \mathbb{Z}$. This conversion does not need CRT composition, but it has a noise which means the result is $a + kQ \in \mathbb{Z}_P$ for $|k| < \ell$. The effect of this noise is tiny in the case of `HEAAN-RNS` scheme, but we can reduce the noise by adapting base conversion in [11]. This base conversion additionally compute $v = \lceil \sum_{i=0}^{\ell-1}[a^{(j)} \cdot \hat{q}_j^{-1}]_{q_i}/q_i \rfloor$ in $\mathbb{R}$.

Using this conversion, authors of [6] define two algorithms to expand and reduce the modulus space which are called `ModUp` and `ModDown` (see Algorithm

1 and 2 in [6]):

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

for $\mathcal{D}=\{p_0,\ldots,p_{k-1},q_0,\ldots,q_{\ell-1}\}$, $\mathcal{B}=\{p_0,\ldots,p_{k-1}\}$, $\mathcal{C}=\{q_0,\ldots,q_{\ell-1}\}$, and $P=\prod_{i=0}^{k-1}p_i$. We note that `ModUp` algorithm expand the modulus space, which means it sends $a\in\mathcal{C}$ to $a\in\mathcal{D}$. And, `ModDown` algorithm reduce the modulus space and divide the value by $P=\prod p_i$, which means `ModDown` sends $a\in\mathcal{D}$ to $\lfloor a/P\rceil+\epsilon\in\mathcal{C}$ for small ϵ. This algorithm is used for modulus switching after key-switching and rescaling in `HEAAN-RNS` scheme.

Scheme Description. In this section, K is $(2N)$-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 $R_q=R/qR$.

$\underline{\texttt{Setup}(q,L,\eta;1^\lambda)}$. A base integer q, the number of levels L, and the bit precision η are given as inputs with the security parameter λ.

- Choose a basis $\mathcal{D}=\{p_0,\ldots,p_{k-1},q_0,q_1,\ldots,q_L\}$ such that $q_j/q\in(1-2^{-\eta},1+2^{-\eta})$ for $1\le j\le L$. We write $\mathcal{B}=\{p_0,\ldots,p_{k-1}\}$, $\mathcal{C}_\ell=\{q_0,\ldots,q_\ell\}$, and $\mathcal{D}_\ell=\mathcal{B}\cup\mathcal{C}_\ell=\{p_0,\ldots,p_{k-1},q_0,\ldots,q_\ell\}$ for $0\le\ell\le L$. Let $P=\prod_{i=0}^{k-1}p_i$ and $Q=\prod_{j=0}^{L}q_j$.
- Choose a power-of-two integer N.
- Choose a secret key distribution χ_{key}, an encryption key distribution χ_{enc}, and an error distribution χ_{err} over R.
- Let $\hat{p}_i=\prod_{0\le i'<k,i'\ne i}p_{i'}$ for $0\le i<k$. Compute the constants $[\hat{p}_i]_{q_j}$ and $[\hat{p}_i^{-1}]_{p_i}$ for $0\le i<k$, $0\le j\le L$.
- Compute the constants $[P^{-1}]_{q_j}=\left(\prod_{i=0}^{k-1}p_i\right)^{-1}\pmod{q_j}$ for $0\le j\le L$.
- Let $\hat{q}_{\ell,j}=\prod_{0\le j'\le\ell,j'\ne j}q_{j'}$ for $0\le j\le\ell\le L$. Compute the constants $[\hat{q}_{\ell,j}]_{p_i}$ and $[\hat{q}_{\ell,j}^{-1}]_{q_j}$ for $0\le i<k$, $0\le j\le\ell\le L$.

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

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

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

$\underline{\texttt{KeyGen}}$.

- Sample $s \leftarrow \chi_{\mathsf{key}}$ and set the secret key as $\mathsf{sk} \leftarrow (1, s)$.
- Sample $(a^{(0)}, \ldots, a^{(L)}) \leftarrow U\left(\prod_{j=0}^{L} R_{q_j}\right)$ and $e \leftarrow \chi_{\mathsf{err}}$. Set the public key as

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

  where $b^{(j)} \leftarrow -a^{(j)} \cdot s + e \pmod{q_j}$ for $0 \le j \le L$.
- Set the evaluation key as $\mathsf{evk} \leftarrow \texttt{KSGen}(s^2, s)$.

$\underline{\texttt{Enc}_{\mathsf{pk}}(m)}$. For $m \in R$, sample $v \leftarrow \chi_{\mathsf{enc}}$ and $e_0, e_1 \leftarrow \chi_{\mathsf{err}}$. Output 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 $0 \le j \le L$.

$\underline{\texttt{Dec}_{\mathsf{sk}}(\mathsf{ct})}$. For $\mathsf{ct} = \left(\mathsf{ct}^{(j)}\right)_{0 \le j \le \ell}$, output $\langle \mathsf{ct}^{(0)}, \mathsf{sk} \rangle \pmod{q_0}$.

$\underline{\texttt{Add}(\mathsf{ct}, \mathsf{ct}')}$. Given two ciphertexts $\mathsf{ct} = \left(\mathsf{ct}^{(0)}, \ldots, \mathsf{ct}^{(\ell)}\right), \mathsf{ct}' = \left(\mathsf{ct}'^{(0)}, \ldots, \mathsf{ct}'^{(\ell)}\right) \in \prod_{j=0}^{\ell} R_{q_j}^2$, output a 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 $0 \le j \le \ell$.

$\underline{\texttt{Mult}_{\mathsf{evk}}(\mathsf{ct}, \mathsf{ct}')}$. 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 procedures and return a 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. Compute $\texttt{ModUp}_{\mathcal{C}_\ell \to \mathcal{D}_\ell}(d_2^{(0)}, \ldots, d_2^{(\ell)}) = (\tilde{d}_2^{(0)}, \ldots, \tilde{d}_2^{(k-1)}, d_2^{(0)}, \ldots, d_2^{(\ell)})$.
3. Compute

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

   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 $0 \le i < k$, $0 \le j \le \ell$.

4. Compute

$$\left(\hat{c}_0^{(0)}, \ldots, \hat{c}_0^{(\ell)}\right) \leftarrow \texttt{ModDown}_{\mathcal{D}_\ell \to \mathcal{C}_\ell}\left(\tilde{c}_0^{(0)}, \ldots, \tilde{c}_0^{(k+\ell)}\right),$$
$$\left(\hat{c}_1^{(0)}, \ldots, \hat{c}_1^{(\ell)}\right) \leftarrow \texttt{ModDown}_{\mathcal{D}_\ell \to \mathcal{C}_\ell}\left(\tilde{c}_1^{(0)}, \ldots, \tilde{c}_1^{(k+\ell)}\right).$$

5. 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 $0 \le j \le \ell$.

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

3 Full-RNS variant of HEAAN

In the previous scheme, decomposition method is not used for key-switching (or relinearization). For this reason, we have to use the parameter $k \simeq L + 1$ which is length of temporary modulus chain in `HEAAN-RNS` scheme. By using large k, the number of switching key is reduced. However, the security of the scheme depends on the largest ciphertext modulus which is $(\prod q_i \cdot \prod p_i)$. So larger k makes the ring dimension larger for the same security level. Using temporary modulus has pros and cons, so we need to set proper number of primes for that. To keep ciphertext in RNS form and use proper $k < L + 1$, we combine RNS decomposition technique in [2] with temporary modulus technique in [10].

First, we will introduce RNS decomposition technique in the previous work which can be seen as following equations:

$$\mathsf{RNS\text{-}Decomp}_{\mathcal{C}}(a) = ([a \cdot \hat{q_0}^{-1}]_{q_0}, [a \cdot \hat{q_1}^{-1}]_{q_1}, \ldots, [a \cdot \hat{q_L}^{-1}]_{q_L})$$
$$\mathsf{RNS\text{-}Power}_{\mathcal{C}}(b) = (b \cdot \hat{q}_0, b \cdot \hat{q}_1, \ldots, b \cdot \hat{q}_L)$$

for $\mathcal{C} = \{q_0, q_1, \ldots, q_L\}$ and $\hat{q}_i = \prod_{j \ne i} q_j$. Those functions work similar as bit-decomposition and power of two technique:

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

for $Q = \prod_{i=0}^{L} q_i$. Because the maximum size of $\mathsf{RNS\text{-}Decomp}_{\mathcal{C}}(a)$'s elements is $\mathsf{max}(q_i)$ for $0 \le i \le L$, this function can be used for key-switching. In detail, with $(L+1)$ number of public keys which are corresponding to $\mathsf{RNS\text{-}Power}_{\mathcal{C}}(s(x)^2) + \texttt{Enc}(0)$, we can perform key-switching as follows:

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

In this method, $(L+1)$ public keys make the complexity for key-switching larger than $O(L^2)$ multiplications in R_{q_i}. Furthermore, the noise growth during key-switching is about $||e_{\mathsf{fresh}}|| \cdot \mathsf{max}(||q_i||)$ which is quite large compare to the size of plaintext.

To solve this problem, we will use special modulus and reduce the expansion ratio during decomposition[1]. Instead of using $\{q_i\}_{0\le i\le L}$ in RNS decomposition, we use $\{Q_j\}_{0\le j<\mathtt{dnum}} = \{\prod_{i=j\alpha}^{(j+1)\alpha-1} q_i\}_{0\le j<\mathtt{dnum}}$ for $\alpha = (L+1)/\mathtt{dnum}$ for prefixed dnum. In addition, we apply fast base conversion to avoid CRT composition in relinearization. And we use k number of special primes for $k=\alpha$ to reduce the noise growth during key-switching. The brief introduction about our relinearization method can be represented as follows:

0. For $k = (L+1)/\mathtt{dnum}$, set evaluation key as $\mathsf{evk} = \mathsf{RNS\text{-}Power}_{\mathcal{C}'}(s(x)^2) + \mathtt{Enc}_{s(x)}(0) \bmod PQ$ for $Q = \prod_{i=0}^{L} q_i$, $P = \prod_{i=0}^{k-1} p_i$, and $\mathcal{C}' = \{Q_j\}_{0\le j<\mathtt{dnum}}$.
1. For the 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
$$(b'(x), a'(x)) = \langle \mathsf{RNS\text{-}Decomp}_{\mathcal{C}'}(a(x)), \mathsf{evk}\rangle \bmod PQ.$$
In $\mathsf{RNS\text{-}Decomp}_{\mathcal{C}'}(a(x)) \bmod PQ$ computation, we avoid CRT composition using fast base conversion.
2. We apply modulus-switching using ModDown function to reduce the size of noise:
$$(b''(x), a''(x)) = \lfloor (b'(x), a'(x))/P \rceil \bmod Q$$
3. Return $(c(x) + b''(x), b(x) + a''(x))$.

The revised method can be adapted to homomorphic multiplication (or key-switching) process. The only difference is key-switching part, so we only need to revise $\mathtt{KSGen}(s_1, s_2)$, KeyGen, and $\mathtt{Mult}_{\mathsf{evk}}(\mathsf{ct}, \mathsf{ct}')$ algorithms in the previous scheme. We remark that $\mathtt{dnum} = 1$ case is same as the previous scheme in Section 2. Using larger dnum makes length of the vector evk larger, but dimension of ring will be reduced when we assume the same security level. Detail comparison with previous work will be at Section 3.2.

3.1 Scheme description

We will focus on the differences between the previous scheme and ours. Other algorithms which are not described in this section is same as the scheme in Section 2. In this section, let

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

for the given integer $\mathtt{dnum} > 0$ and $\alpha = (L+1)/\mathtt{dnum}$. And, $\hat{Q}_i = \prod_{j\ne i} Q_i$.

[1] In the case of SEAL, they use bit-decomposition technique with RNS-decomposition to reduce the noise growth. But this increase the number of public key for key-switching which directly related with complexity of the algorithm.

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

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

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 $0 \le i \le L$.

$\underline{\texttt{KeyGen}}$.

- Same with previous one
- Same with previous one
- Set the evaluation key as $\{\mathsf{evk}_i\}_{0 \le j \le \texttt{dnum}} \leftarrow \texttt{KSGen}(s^2, s)$.

For convenient explanation about homomorphic multiplication step, let partition of $\mathcal{C}$ as $\mathcal{C}_i = \{q_{i\alpha}, \ldots, q_{\mathsf{min}((i+1)\alpha-1, L+1)}\}$ and $\mathcal{D}_i = (\cup_{0 \le j < i} \mathcal{C}_j) \cup \{p_0, \ldots, p_{k-1}\}$.

$\underline{\texttt{Mult}_{\mathsf{evk}}(\mathsf{ct}, \mathsf{ct}')}$. Given two ciphertexts $\mathsf{ct} = \left(\mathsf{ct}^{(j)} = (c_0^{(j)}, c_1^{(j)})\right)_{0 \le j \le \ell}$ and $\mathsf{ct}' = \left(\mathsf{ct}'^{(j)} = (c'^{(j)}_0, c'^{(j)}_1)\right)_{0 \le j \le \ell}$, perform the following procedures and return a ciphertext $\mathsf{ct}_{\mathsf{mult}} \in \prod_{j=0}^{\ell} R_{q_j}^2$.

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

$$\begin{aligned} d_0^{(j)} &\leftarrow c_0^{(j)} c'^{(j)}_0 \pmod{q_j}, \\ d_1^{(j)} &\leftarrow c_0^{(j)} c'^{(j)}_1 + c_1^{(j)} c'^{(j)}_0 \pmod{q_j}, \\ d_2^{(j)} &\leftarrow c_1^{(j)} c'^{(j)}_1 \pmod{q_j}. \end{aligned}$$

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

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

   for $0 \le i < \alpha$ and $0 \le j < \beta$ and $Q' = \prod_{i=\ell+1}^{L} q_i$. Note that zero-padding is not needed when $j = \texttt{dnum} - 1$.

   2-2. RNS-Decompose: for each

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

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

2-3. Compute $\tilde{d}_{2,j} = \mathtt{ModUp}_{\mathcal{C}_j \to \mathcal{D}_\beta}(d'_{2,j})$.

3. Compute

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

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

4. Compute

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

5. 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 overall process of our multiplication algorithm from step 2 to step 4 which are key parts of our new algorithm. The gray area indicates special modulus which is corresponding to $\{p_0, p_1, \ldots, p_{k-1}\}$. By using multiple number

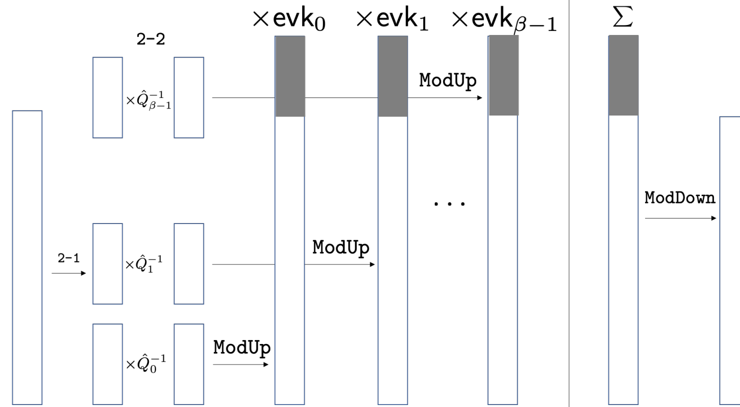


Fig. 1: Overview of our algorithm from step 2 to 4.

of evaluation keys, we can reduce the number of special primes. Because ring dimension of R is decided from the security of (R)LWE problem, smaller special primes will make the ring dimension N smaller for small security level λ:

$$\frac{N}{\sum_{i=0}^{L} \log_2 q_i + \sum_{i=0}^{k-1} \log_2 p_i} \propto \Omega(\frac{\lambda}{\log \lambda}).$$

3.2 Comparison

In our Full-RNS variant of HEAAN, we reduce the number of special primes by using multiple number of evaluation keys with RNS-decomposition technique and fast base conversion. In this section, we will show the complexity depends on the parameter dnum. This result gives the reason for using proper $1 < \texttt{dnum} < L+1$. First, we need to check the complexity for each step of multiplication. We assume that ciphertext is already NTT transformed by default. And the complexity in this section means the number of multiplications in $\mathbb{Z}_{p_i}$ or $\mathbb{Z}_{q_i}$.

Step 1. This step is to compute tensor product between two vectors (with length 2). By using karatsuba multiplication sense, this step only requires 3 polynomial multiplications. Therefore, this step require 3 hadamard multiplications and 3 inverse NTT transform for each ring R_{q_i}:

$$3(\ell+1)N + 3(\ell+1)N\log N$$

Step 2-2. The step 2-1 is just rearranging the vector and zero-padding which requires no complexity, and the multiplication in step 2-1 can be merged with the multiplication in step 2-2. So we will ignore this part. In the step 2-2, ℓ number of modulus multiplications are needed: $\ell \cdot N$.

Step 2-3. ModUp algorithm requires $n(m-n)$ multiplications for input vector size n and output vector size m. So, the complexity for $\texttt{ModUp}_{\mathcal{C}_j \to \mathcal{D}_\beta}$ is $\alpha(\alpha\beta+k-\alpha) = \alpha^2\beta$. We have to run it β times for each coefficients, so total complexity is following: $\alpha^2\beta^2 N \simeq \ell^2 N$.

Step 3. We assume that evaluation keys are already NTT transformed at key generation step. In detail, this process includes following 4 steps:

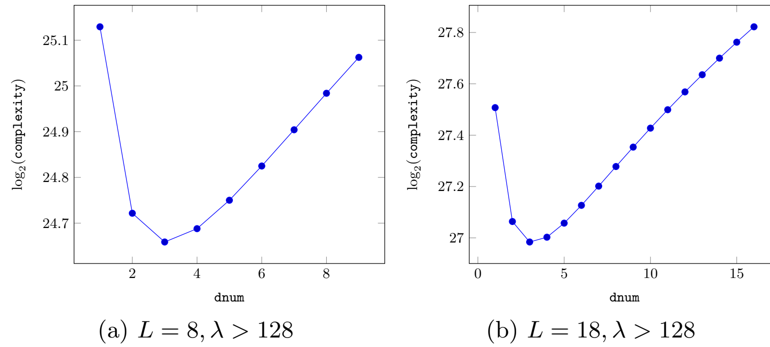
1. NTT transform: we need to apply NTT algorithm for all $\{\tilde{d}_{2,j}^{(i)}\}_{0 \le i < k+\alpha\beta}$:

$$\beta(k+\alpha\beta)N\log N \simeq \beta(k+\ell)N\log N$$

2. Hadamard Mult: $2\beta(k+\alpha\beta)N \simeq 2\beta(k+\ell)N$.
3. Summation: there is no multiplication.
4. Inverse NTT transform: $(k+\ell)N\log N$.

Step 4. ModDown algorithm requires $m(n-m)$ multiplications for input vector size n and output vector size m. So the complexity of $\texttt{ModDown}_{\mathcal{D}_\beta \to \mathcal{C}_\ell}$ is $\ell(k+\alpha\beta-\ell) \simeq \ell \cdot k$. We ignore step 5 which only needs some additions (no multiplication).

Total Complexity. For some given parameters, we compute total complexity and check the effect of dnum parameter. For the convenient comparison, we assume that the input ciphertext has maximum level L. And to use same security level, we reduce the ring dimension to keep $N/(\sum_{i=0}^{L}\log_2 q_i + \sum_{i=0}^{k-1}\log_2 p_i)$ values when the number of special prime is changed.



(a) $L = 8, \lambda > 128$ (b) $L = 18, \lambda > 128$

Fig. 2: Complexity with various `dnum` parameter.

Figure 2 shows changes in complexity for various use of `dnum`. For convenient comparison, we used $\log q_i = 40$ for $i > 0$ and $\log q_0 = 50$ and $\log p_i = 50$. And, to keep security level similar, we reduce the ring dimension N when `dnum` $(= (L+1)/k)$ increases to maintain $N/(40L + 10 + 50k)$. As a result, we found out that using `dnum` $= 3$ gives minimum complexity for both case in Figure 2.

Remark 1 (Quantization Method). The homomorphic encryption scheme in Section 2 is described for plaintext $m(x) \in R = \mathbb{Z}[X]/(X^N+1)$. To encrypt a vector of complex number, we can use isomorphism between $\mathbb{C}^{N/2}$ and $\mathbb{R}[X]/(X^N+1)$. Define $\rho : \mathbb{C}^{N/2} \to \mathbb{R}[X]/(X^N+1)$ and $\mathsf{Encode}(\boldsymbol{m}, \Delta) = \lceil \Delta \cdot \rho^{-1}(\boldsymbol{m}) \rfloor = m(x) \in R$. The previous scheme use the same scaling factor Δ for all level by using $q_i \simeq \Delta$ for all i. However, this method makes additional noise in `Rescale` process. As in `SEAL`, we use different scaling factor for each level. It means that we just regard `Rescale` process as dividing scaling factor by q_i.

4 Better Sine Evaluation

In [7], authors suggest the way to approximate the modulus operation, which is the hardest part of the bootstrapping, with sine function. After that, the key part of the bootstrapping for `HEAAN` becomes an homomorphic evaluation of $\sin(x)$ for encrypted x. In [7], to evaluate $\sin(x)$, they first scale down x 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. Then, they compute the original sine value of x by using double angle formula.

On the other hand, in [5], they use the Chebyshev polynomial approximation instead of the Taylor approximation. All those previous works can be presented as following equation:

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

$$\uparrow$$

$$\text{If } |\tfrac{x}{q} - I| \le \epsilon \text{ for some integer } |I| < K$$

for some suitable polynomial $p_n(x) \in \mathbb{R}[X]$. The difference between previous works occurs in the step of approximating sine function by a polynomial. The first approximation of the modulus operation with sine function is reasonable because of the fact that $\frac{x}{q}$ locates close enough to some integer. We note that the error $O(\epsilon^3)$ come out from the first approximation when $|\frac{x}{q} - I| \le \epsilon$ for some integer $|I| < K$.

However, all the previous works do not use the property that $\frac{x}{q}$ locates close enough to some integer in the second approximation. In other words, they just find a polynomial that approximates sine function well in the global sense. Therefore, there is a room for finding a better approximate polynomial $p(x)$ based on the property.

From now on, by scaling x, we approximate $\sin(2\pi x)$ (in fact, $\cos(2\pi x)$) instead of $\sin(\frac{2\pi x}{q})$ for simplicity. Now, we use the condition $|x - I| \le \epsilon$ for some integer $|I| < K$. Followings are key idea of our method for finding a better approximate polynomial.

- We use $\cos(2\pi x)$ instead of $\sin(2\pi x)$. Computing $\sin(2\pi x)$ using $\cos(2\pi x)$ just need one shifting $x \leftarrow x - 1/4$. Since double angle formula of $\cos(2\pi x)$ only consists of cosine function, this enables us to use the hybrid method that combines polynomial approximation and double angle formula.
- Our polynomial approximation is optimized for $x \in \cup_{i=-K+1}^{K-1} I_i$ for $I_i = [i - \frac{1}{4} - \epsilon, i - \frac{1}{4} + \epsilon]$. For that, we find a better choice of nodes than that of the Chebyshev method. As a result, our approach can effectively use the fact that x is contained in one of the small intervals.
- The error bound between $\cos(2\pi x)$ and $p_n(x)$ obtained from our method is given by $||\cos(2\pi x) - p_n(x)|| = O(\epsilon^{d_i})$ on I_i, where d_i's are the number of nodes in each interval I_i, which will be explained in more detail. In contrast, the error bound between $\cos(2\pi x)$ and $p_n(x)$ obtained from the Chebyshev method is given by $||\cos(2\pi x) - p_n(x)|| = O(1)$ with respect to ϵ.

4.1 Our Method

When a sufficiently smooth function is estimated by an interpolation polynomial, an error, difference between the real value and the 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 $x_0, x_1, \cdots, x_n \in [a,b]$*, i.e.* $p_n(x_i) = f(x_i)$ *for all* $0 \le i \le n$*. Then, for each* $x \in [a,b]$*, there exists a point* $\psi_x \in [a,b]$ *such that*

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\psi_x)}{(n+1)!} \cdot \prod_{i=0}^{n}(x - x_i). \tag{2}$$

Let $p_n(x)$ be the interpolation polynomial of degree $\le n$ that interpolates $\sin(2\pi x)$ (or $\cos(2\pi x)$) at $n+1$ distinct points. Then, the error bound between $\sin(2\pi x)$ (or $\cos(2\pi x)$) and $p_n(x)$ can be computed by the Equation 2. Even though the term $\frac{f^{(n+1)}(\psi_x)}{(n+1)!}$ in the Equation 2 is hard to be estimated exactly, it is bounded by the constant when f is $\sin(2\pi x)$ (or $\cos(2\pi x)$). Thus, the error bound of the polynomial approximation depends on the other term

$$w(x) = \prod_{i=0}^{n}(x - x_i)$$

for pre-determined $x_0, x_1, \ldots, x_n \in [a,b]$ which are called *nodes*. Thus, we need to choose $\{x_i\}_{1 \le i \le n}$ appropriately to minimize the maximum value of $w(x)$ in specified domain of x. In the case of the Chebyshev method, the nodes are chosen by $x_i = \frac{b+a}{2} + \frac{b-a}{2} \cdot \cos(\frac{2i-1}{2n+2}\pi)$ for $1 \le i \le n+1$ in the range $[a,b]$, and these nodes make the upper bound of $w(x)$ in the whole interval to be $\frac{1}{2^n} \cdot (\frac{b-a}{2})^{n+1}$ (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 the global sense, it is not appropriate for our purpose because it does not consider the condition that x is near one of the points. Therefore, we focus on the bound of $w(x)$ for $x \in \cup_{i=-K+1}^{K-1} I_i$ with $I_i = [i - \frac{1}{4} - \epsilon, i - \frac{1}{4} + \epsilon]$ and propose better methods for this setting.

Hermite Interpolation. There is an extended version of Taylor approximation which is called *Hermite* interpolation (or multi-point Taylor). Given m distinct points $x_1, \cdots, x_m \in [a,b]$, their respective multiplicities $d_1, \cdots, d_m$ and $n = \sum_{i=1}^{m} d_i - 1$, Hermite polynomial $p_n(x)$ of f is the polynomial of degree $\le n$ which satisfies following equations:

$$p_n(x_i) = f(x_i), \ \frac{d^k p_n(x_i)}{dx^k} = \frac{d^k f(x_i)}{dx^k} \text{ for } 1 \le k < d_i$$

for $1 \le i \le m$. As in the original polynomial interpolation which requires interpolated points to be distinct, Hermite interpolation also has the similar theorem as follows:

Theorem 2 (Hermite interpolation). *Let* f *be a function in* $C^{n+1}[a,b]$ *and* p_n *be a Hermite polynomial of degree* $\le n$ *that interpolates the function* f *at* m *distinct points* $x_1, \cdots, x_m \in [a,b]$ *with multiplicities* $d_1, \cdots, d_m$ *and* $n = \sum d_i - 1$*. Then, for each* $x \in [a,b]$*, there exists a point* $\psi_x \in [a,b]$ *such that*

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\psi_x)}{(n+1)!} \cdot \prod_{i=1}^{m}(x - x_i)^{d_i}. \tag{3}$$

By the same reason as in the case of original polynomial interpolation, we need to focus on the term

$$w(x) = \prod_{i=1}^{m}(x - x_i)^{d_i}.$$

when f is the function $\cos(2\pi x)$. This method is appropriate for the case when the input for the function is close enough to one of the nodes $\{x_i\}_{1\leq i\leq m}$, and our input for the function $\cos(2\pi x)$ is contained in one of the intervals $I_i = (i - \frac{1}{4} - \epsilon, i - \frac{1}{4} + \epsilon)$. For this reason, we choose $2K-1$ nodes $x_i = i - \frac{1}{4}$ for $-K < i < K$. (We use the negative indices for simplicity.) In this case, from the Equation 3, the bound of $||w(x)||$ is given as follows:

$$||w(x)|| \leq \epsilon^{d_i} \cdot \prod_{j=1}^{K-1-i}(j+\epsilon)^{d_{i+j}} \cdot \prod_{j=1}^{K-1+i}(j+\epsilon)^{d_{i-j}}$$

when $x \in I_i = [i - \frac{1}{4} - \epsilon, i - \frac{1}{4} + \epsilon]$. Hence, the error bound between $\cos(2\pi x)$ and $p_n(x)$ is given by

$$||\cos(2\pi x) - p_n(x)|| \leq \frac{(2\pi)^{n+1}}{(n+1)!} \cdot \epsilon^{d_i} \cdot \prod_{j=1}^{K-1-i}(j+\epsilon)^{d_{i+j}} \cdot \prod_{j=1}^{K-1+i}(j+\epsilon)^{d_{i-j}}$$

when $x \in I_i = [i - \frac{1}{4} - \epsilon, i - \frac{1}{4} + \epsilon]$. Thus, $||\cos(2\pi x) - p_n(x)|| = \mathrm{O}(\epsilon^{d_i})$ on I_i, which means that the error bound decreases as ϵ, representing the ratio between the size of message and the size of cipertext modulus, gets smaller. In contrast, the error between $\cos(2\pi x)$ and $p_n(x)$ 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 $M_i = \mathsf{Max}_{x\in I_i}||w(x)||$ for each $-K < i < K$, then we obtain

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

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

Our Optimized Nodes. We can reduce the error bound further by mixing ideas of two methods, Chebyshev and Hermite. 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 $x_{i,j} = i - \frac{1}{4} + \epsilon \cdot \cos\left(\frac{2j-1}{2d_i}\pi\right)$ for $1 \leq j \leq d_i$. Let $n = \sum d_i - 1$ and p_n be the polynomial of degree $\leq n$ that interpolates the function $\cos(2\pi x)$ at $n+1$ distinct points $x_{i,j}$ $(1 \leq i \leq m, 1 \leq j \leq d_i)$. In other words, p_n satisfies the following equation:

$$p_n(x_{i,j}) = \cos(2\pi x_{i,j}) \text{ for } 1 \leq i \leq m, 1 \leq j \leq d_i$$

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

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

when $x \in I_i = [i - \frac{1}{4} - \epsilon, i - \frac{1}{4} + \epsilon]$. We can see that the error bound decreases by a factor of $2^{d_i - 1}$ compare to that of the Hermite interpolation, and also $||\cos(2\pi x) - p_n(x)|| = \mathrm{O}(\epsilon^{d_i})$ on I_i as the Hermite method. Denote $M_i = \mathsf{Max}_{x \in I_i}||w(x)||$, then we obtain

$$||\cos(2\pi x) - p_n(x)|| \leq \frac{(2\pi)^{n+1}}{(n+1)!} \cdot \mathsf{Max}\{M_{-K+1}, M_{-K+2}, \ldots, M_{K-1}\}. \quad (4)$$

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

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 $i_0 = \mathrm{argmax} M_i$, we increase d_{i_0} by 1. We repeat this process until the total degree ($= \sum d_i$ - 1) becomes target degree. Sage code corresponding to

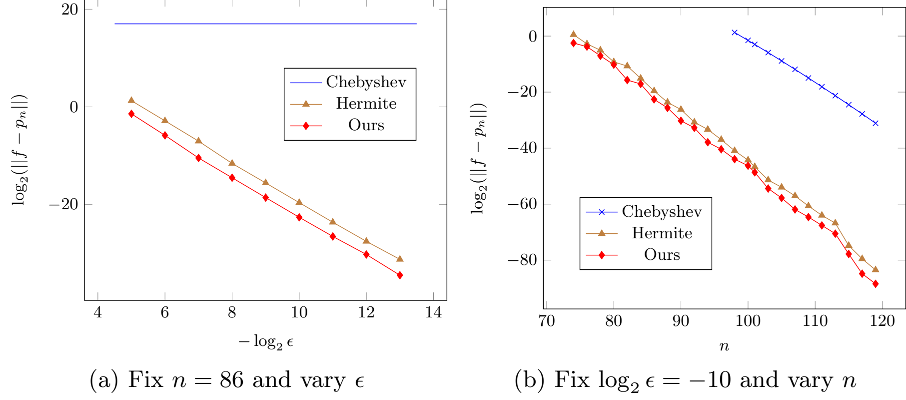
Algorithm 1 Choosing the number of nodes in each interval

1: **Input** : Target degree n
2: Initialize $d_i = 1$ for all i
3: **while** $\sum d_i - 1 \leq n$ **do**
4: Compute M_i for each i
5: Find i_0 = argmax M_i
6: $d_{i_0} \leftarrow d_{i_0} + 1$
7: **end while**
8: **Output** : d_i's

computing M_i and Algorithm 1 is included in the Appendix.

Comparison. We conduct an experiment to compare our method and the other methods. First, we compare the theoretical error bound of our method, Equation 4, with those of other methods. Figure 3 shows theoretical error bounds between the function $\cos(2\pi x)$ and its approximate polynomials obtained from each method. Figure 3(a) is obtained by fixing $n = 86$ and varying ϵ and Figure 3(b) is obtained by fixing $\log_2 \epsilon = -10$ and varying n. As seen in Figure 3, our method is far better than the Chebyshev method and slightly better than the Hermite method. For fixed $n = 86$, while the theoretical error bound of Chebyshev method has enormous value and does not change at all as ϵ varies, that of our method has fairly small value and even gets smaller as ϵ decreases. For example, $\log_2$ value of the theoretical error bound of ours is about -22.6 when $\log_2 \epsilon = -10$, but that of the Chebyshev method is about 17.0 independent of ϵ.

Also, for fixed $\log_2 \epsilon = -10$, the Chebyshev method requires about 30 more degrees to yield the same level of theoretical error bound as our method. For



(a) Fix $n = 86$ and vary ϵ

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

Fig. 3: Theoretical error bounds $\log_2(||f - p_n||)$ for our optimized interpolation ($K = 12$).

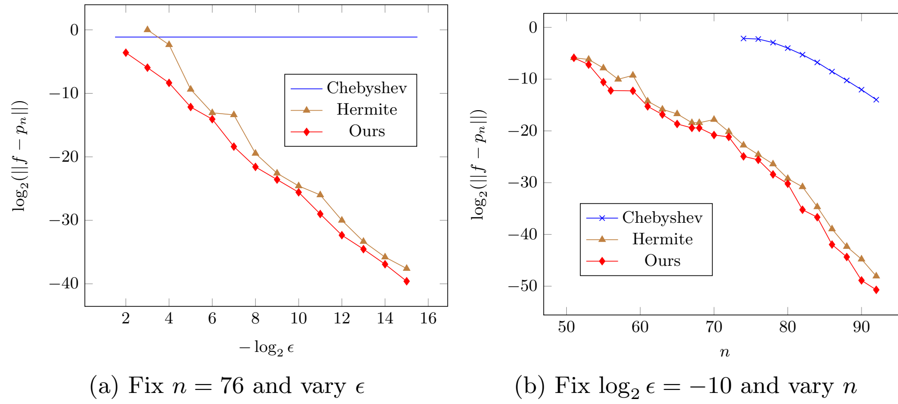
instance, the degree of an approximate polynomial of the Chebyshev method should be greater than 115 to have the same level of the theoretical error bound as the approximate polynomial of degree 86 of ours. Furthermore, for this reason, the Chebyshev method appears on the graph after the degree becomes greater than 98.

Next, we compare the experimental error bound of our method with those of other methods. Figure 4 shows experimental error bounds between the function $\cos(2\pi x)$ and its approximate polynomials obtained from each method. The Figure 4(a) is obtained by fixing $n = 76$ and varying ϵ and Figure 4(b) is obtained by fixing $\log_2 \epsilon = -10$ and varying n. Since we bound the term $\frac{f^{(n+1)}(\psi_x)}{(n+1)!}$ in the Equation 2 by a constant, an exact error bound is smaller than the theoretical error bound. For this reason, we plot Figure 4(a) with smaller fixed value $n = 76$ compare to the theoretical case.

As in the theoretical case, we can see that our method is much better than the Chebyshev method and slightly outdoes the Hermite 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 $p_n(x) \in \mathbb{R}[X]$

After we get an approximate polynomial of degree $\leq n$ using our optimized method, we need to evaluate a value of the function at x in a homomorphic way. Naive approach is to compute x^i for all $i \in \{0, 1, \ldots, n\}$ first and then evaluate $p_n(x) = \sum_{i=0}^{n} p_i \cdot x^i$. Unfortunately, due to the un-stability of the coefficients, this



(a) Fix $n = 76$ and vary ϵ

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

Fig. 4: Experimental error bounds $\log_2(||f - p_n||)$ for our optimized interpolation ($K = 12$).

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 huge modulus values to encrypt these values. To avoid this problem, we represent the polynomial with the Chebyshev basis instead of x^i's.

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

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

Then, T_i satisfies the equation $T_i(\cos x) = \cos(ix)$ for all x, and thus $|T_i(x)| \leq 1$ for all $|x| \leq 1$. Since our approximate polynomial has the domain $[-K, K]$, we use $\tilde{T}_i(x) = T_i(\frac{x}{K})$ instead of T_i for each i. Note that $|\tilde{T}_i(x)| \leq 1$ for all $|x| \leq K$. Some $\tilde{T}_i$'s with small i in the case of $K = 12$ are given in Figure 5.

Since each $\tilde{T}_i$ is the polynomial of degree i, $\{\tilde{T}_i\}_{i=0}^n$ forms a basis for the vector space of polynomials of degree $\leq n$. Thus, $p_n(x)$ can be represented by a linear combination of $\{\tilde{T}_i\}_{i=1}^n$ as

$$p_n(x) = \sum_{i=0}^{n} c_i \cdot \tilde{T}_i(x)$$

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 x^i basis representation. However, since $|\tilde{T}_i(x)| \leq 1$ for all $|x| \leq K$, the term $c_i \cdot \tilde{T}_i(x)$ with extremely small c_i has little effect on the value of $p_n(x)$. 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, and make the computation stable.

Then, we can use the Baby-step Giant-step algorithm given in Algorithm 2 to evaluate the polynomial $p_n(x)$. This algorithm enables us to evaluate $p_n(x)$ in

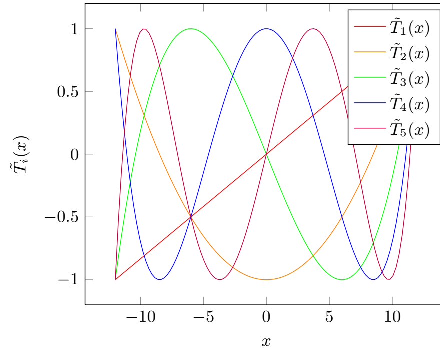


Fig. 5: Chebyshev polynomials for range $[-12, 12]$

$2\sqrt{2n} + \frac{1}{2}\log_2 n + O(1)$ 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.

Algorithm 2 Baby-step Giant-step algorithm

Input : A polynomial of degree n, $p = \sum_{i=0}^{n} c_i T_i$.
Let m be the smallest integer integer satisfying $2^m > n$ and $l \approx m/2$.
Evaluate $T_2(x), T_3(x) \cdots, T_{2^l}(x)$ inductively.
Evaluate $T_{2^l+1}(x) \cdots, T_{2^{m-1}}(x)$ using the equation $T_{2i}(x) = 2T_i(x)^2 - 1$.
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.
Evaluate $q(x)$ and $r(x)$ recursively. (Repeat the above step with replacing p with q and r until the degree of the quotient and the remainder become smaller than 2^l)
Evaluate $p(x)$ with $q(x)$, $r(x)$ and $T_{2^{m-1}}(x)$.
Output : $p(x)$

Also, we can use the Paterson-Stockmeyer algorithm for the Chebyshev polynomial suggested in [5]. In [5], authors modify the original Paterson-Stockmeyer algorithm [18] 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 $\sqrt{2n} + \log_2 n + O(1)$ non-scalar multiplications. Refer to [5] for more detail. By using this algorithm, we can evaluate $p_n(x)$ with $\sqrt{2n} + \log_2 n + O(1)$ 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 $p_n(x)$ 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 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 x 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[2]:

$$\cos(2x) = 2\cos^2 x + 1.$$

Suppose we scale down x by 2^r before using our method and let $y = x/2^r$. We say the number of scaling is r in this case. Then, we need to approximate $\cos y$ based on the fact that y 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 $\tilde{x}_{i,j} = \frac{1}{2^r}\left(i - \frac{1}{4} + \epsilon \cdot \cos\left(\frac{2j-1}{2d_i}\pi\right)\right)$ for $1 \le j \le d_i$ and $|i| < K$ to apply our method. Then, we can see from the Equation 2 that all the terms in $w(x) = \prod_{i=0}^{n}(x - x_i)$ decrease by a factor of 2^r compare 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 for each number of scaling. The result of the experiment is given in Table 1. The first column indicates degree of approximate polynomials from the original method and the other columns represents 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 depth $\lceil \log_2 n \rceil$ to evaluate $p_n(x)$ in a homomorphic way and require r more depth for double angle formula for cosine function when the number of scaling is r. As our expectation, degree of an approximate polynomial gradually decreases as 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.

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

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 1: 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 $\log_2 \epsilon = -10$)

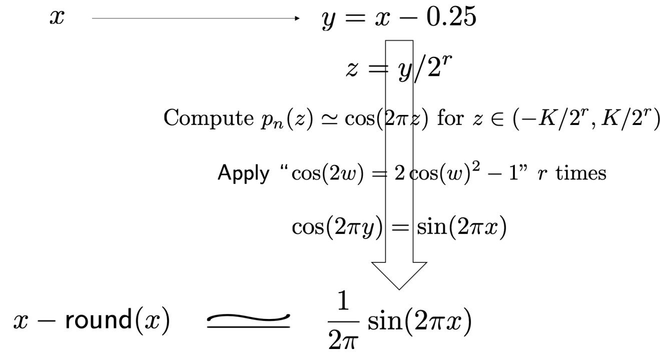


Fig. 6: Overview of our hybrid method

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 2 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 compare to the original method. Thus, in this case, we need to consider the trade-off between the number of non-scalar multiplications and depth consumption. In conclusion, we need to conduct an experiment to decide whether we use the hybrid method or not, and the decision depends on the trade-off between the number of non-scalar multiplications and depth consumption.

4.4 Overall Comparison

As the last step, we compare our method with the previous work [5]. In [5], authors use the approximate polynomial of degree 119 obtained from the Cheby-

shev method. [3] In our case, we choose degree by 74 because the approximate polynomial of degree 74 obtained from our method gives the same level of error as the inevitable error occurs between x and $\frac{1}{2\pi}\sin(2\pi x)$ under the parameter sets that used by the previous work. The comparison between our method and the previous work is given in Table 2. As seen from the table, by using the hybrid method with the number of scaling 2 and evaluating obtained approximate polynomial of degree 30 with Algorithm 2, we can decrease the number of non-scalar multiplications almost by half while consuming the same depth.

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

Table 2: Comparison between our method and the previous work. ($K = 12$ and $\log_2 \epsilon = -10$)

5 Put Everything Together

Using the scheme in Section 3 and sine evaluation method in Section 4, we can do the better bootstrapping for Full-RNS variant of `HEAAN`. The bootstrapping needs three steps: linear transformation, sine evaluation, and inverse linear transformation. About linear transformation and inverse part, we used previous techniques in [12, 5]. In this technique, they decompose the linear transformation into several steps which leads to huge improvement.

Sine Evaluation. To approximate sine function, we use the polynomial interpolation with our optimized nodes, which performs better than the other methods because input for 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 same depth.

With the approximate polynomial obtained from our method, we evaluate it with Algorithm 2, which shows better performance when the degree of an approximate polynomial is small. In our implementation, we fix $\log_2 \epsilon = -10$

[3] In fact, they use the nodes $x_i = K\cos\left(\frac{i\pi}{n}\right)$ for $0 \le i \le n$ instead of nodes $x_i = K\cos\left(\frac{2i-1}{2n+2}\pi\right)$ for $1 \le i \le n+1$. But, there is no big difference.

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, scaling factor is changes. This means that bootstrapping process will change the scaling factor also. This makes some problem when we try to compute output of bootstrapping with fresh ciphertext. Even though they are in same level, there scaling factor is different. So homomorphic addition between these two ciphertexts will give un-expected result.

To solve this problem, at the last step of bootstrapping, evaluation, we multiple 1 with scaling factor Δ' such that $(\Delta' \cdot \Delta)/q_{L'} = \Delta_{L'-1}$ for $\Delta_{L'-1}$ = scaling factor at level $L' - 1$ when current scaling factor is Δ. This makes one level consume, but we can optimize it by merging this process with the last step of linear transformation in bootstrapping.

5.1 Implementation

We implement our full-RNS variant of HEAAN scheme in Section 3 and bootstrapping for this scheme. Our experiment and timing results are checked in PC with Intel(R) Core(TM) i9-9820X CPU @ 3.30GHz using single-thread.

Performance of basic homomorphic operations. First, we shows performance of basic homomorphic operations in Table 3. By using `dnum` > 1, we can reduce the ring dimension. For example, in Table 3, the bit size of modulus is 276 when $\log q_i = 45$ with $L = 5$. This means that the bit size of modulus of relinearization key is 552 when `dnum` $= 1$. Therefore, the security of the parameter set is $\simeq 89 < 128$. But, if we apply `dnum` $= 3$, the security of the parameter set becomes > 128 which makes us to use $N = 2^{14}$.

	$\log q_i$	L	`dnum`	`Enc`	`Dec`	`Add`	`Mult`
$N = 2^{14}$	45	5	2	15 ms	0.2 ms	0.3 ms	42 ms
		7	4	19 ms	0.2 ms	0.4 ms	77 ms
$N = 2^{15}$	45	11	2	60 ms	1.7 ms	1.3 ms	175 ms
		14	3	73 ms	1.7 ms	1.6 ms	249 ms
		15	4	77 ms	1.8 ms	1.8 ms	303 ms

Table 3: Performance of Our Full-RNS variant of `HEAAN`

Bootstrapping Performance. For an experiment about bootstrapping, we first set two parameter sets using LWE estimator [1] in Table 4. The bit size of each prime in modulus chain is set to be 35 for `Param 1` and 45 for `Param 2`, and

$\log q_0$ is 45 and 55 respectively. In the case of `Param 1`, with large `dnum` $= 10$, we can use $\log_2 N = 15$ which is 16 with previous scheme. In our experiment, we use 2 scaling with degree 30 polynomial for sine evaluation (see 3rd parameters of our method in Table 2).

	L	`dnum`	N	$\log Q$	$\log Q + \log P$	Security
`Param 1`	19	10	32768	710	781	121.4
`Param 2`	27	7	65536	1270	1452	127.2

Table 4: Parameter Set

Using those two parameter sets, we run our bootstrapping with various number of slots ℓ. Because of the computational error in the linear transformation part, larger ℓ gives lower precision. Here the meaning of precision is $-\log_2 e$ for e is average noise during bootstrapping. For example, precision 10.0 in the first low means that noise with average size 2^{-10} is added when we compare decrypted results before and after bootstrapping.

	ℓ	`LT`	`Eval Sine`	`Total`	`Precision`	`After Level`
`Param 1`	2^2	15 s	26.3 s	41.3 s	10.0	6
	2^3	20.1 s	25 s	45.1 s	9.9	2
`Param 2`	2^3	51.6 s	69.5 s	121.1 s	19.1	10
	2^{12}	148.3 s	75.9 s	224.2 s	13.1	10
	2^{15}	202 s	74 s	276 s	10.9	10

Table 5: Performance of Bootstrapping on our scheme

In our experiment, we used fixed $\epsilon = 2^{-10}$. Because of the difference between x and $\sin(x)$[4], the maximum precision for bootstrapping is $\simeq \epsilon^2 = 2^{-20}$. As the $\ell = 8$ case with `Param 2` gives 19.1 precision, we can see that sine evaluation with our method gives accurate enough result. We remark that our implementation is proof of concept without special optimization technique. And this is the first implementation of bootstrapping on Full-RNS variant of approximate homomorphic encryption scheme. So, we omit the performance comparison with previous implementations [7, 5]. We expect that fine implementation and parallelization will give huge performance improvements.

[4] $|x - \sin x| < O(x^3)$ for x near the origin point.

6 Conclusion

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

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

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. Jean-Claude Bajard, Julien Eynard, M Anwar Hasan, and Vincent Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *International Conference on Selected Areas in Cryptography*, pages 423–442. Springer, 2016.
3. Fabian Boemer, Yixing Lao, and Casimir Wierzynski. ngraph-he: A graph compiler for deep learning on homomorphically encrypted data. *arXiv preprint arXiv:1810.10121*, 2018.
4. Sergiu Carpov, Nicolas Gama, Mariya Georgieva, and Juan Ramon Troncoso-Pastoriza. 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. Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. *IACR Cryptology ePrint Archive*, 2018:1043, 2018.
6. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In *International Conference on Selected Areas in Cryptography*, pages 347–368. Springer, 2018.
7. 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.
8. 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.
9. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.
10. Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO 2012*, pages 850–867. Springer, 2012.

11. Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. In *Cryptographers Track at the RSA Conference*, pages 83–105. Springer, 2019.
12. Kyoohyung Han, Minki Hhan, and Jung Hee Cheon. Improved homomorphic discrete fourier transforms and fhe bootstrapping. *IEEE Access*, 2019.
13. Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. Efficient logistic regression on large encrypted data. Cryptology ePrint Archive, Report 2018/662, 2018.
14. Yichen Jiang, Chenghong Wang, Zhixuan Wu, Xin Du, and Shuang Wang. Privacy-preserving biomedical data dissemination via a hybrid approach. In *AMIA Annual Symposium Proceedings*, volume 2018, page 1176. American Medical Informatics Association, 2018.
15. Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics*, 11(4):83, 2018.
16. Miran Kim, Yongsoo Song, Baiyu Li, and Daniele Micciancio. Semi-parallel logistic regression for gwas on encrypted data. Cryptology ePrint Archive, Report 2019/294, 2019. https://eprint.iacr.org/2019/294.
17. Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, and Xiaoqian Jiang. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR medical informatics*, 6(2):e19, 2018.
18. Michael S Paterson and Larry J Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, 2(1):60–66, 1973.

A Sage Code

```
def err_bnd_our(idx, K, e, deg):
    n = deg[0]
    for i in range(K-1):
        n = n + 2 * deg[i+1]
    res = RR(n+1) * log(2 * RR(pi))
    for i in range(n+1):
        res = res - log(RR(i+1))
    res = res - (deg[idx] - 1) * log(RR(2))
    res = res + deg[idx] * log(RR(e))
    for i in range(K+idx):
        res = res + deg[abs(idx - i)] * log(RR(i + 1) + e)
    for i in range(K-idx):
        res = res + deg[abs(idx + i)] * log(RR(i + 1) + e)
    return res / log(RR(2))
```

Listing 1.1: Algorithm for compute log(`error`) for our interpolation.

```
def err_bnd_cheby(K, tot_deg):
    res = RR(tot_deg + 1) * log(2 * RR(pi))
    for i in range(tot_deg + 1):
        res = res - log(RR(i+1))
    res = res - RR(tot_deg - 1) * log(RR(2))
```

```
    res = res + tot_deg * log(RR(K))
    return res / log(RR(2))
```

Listing 1.2: Algorithm for compute log(`error`) for Chebyshev interpolation.

```
def find_proper_deg(target_deg, e):
    # initial degree setting
    deg = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
    init_tot_deg = 23
    # target degree
    for t in range((target_deg - init_tot_deg)/2):
        print t
        v = []
        for i in range(12):
            v.append((i, err_bnd_our(i, 12, RR(2*pi) * e, deg)))
        # sort v by error bound
        w = sorted([(value,key) for (key,value) in v])
        # deg[i] += 1 for i that has largest error
        # deg_{-i} = deg_i so we only have positive index case
        # so, deg[i] += 1 makes total degree += 2
        deg[w[11][1]] = deg[w[11][1]] + 1
    return deg
```

Listing 1.3: Algorithm for get optimal degree $\{d_i\}_{-K<i<K}$.