META-BTS: Bootstrapping Precision Beyond the Limit

Youngjin Bae
youngjin.bae@cryptolab.co.kr
CryptoLab. Inc.

Jung Hee Cheon
jhcheon@snu.ac.kr
The Seoul National
University/CryptoLab. Inc.

Wonhee Cho
wony0404@snu.ac.kr
The Seoul National
University

Jaehyung Kim
jaehyungkim@cryptolab.co.kr
CryptoLab. Inc.

Taekyung Kim
taekyung.kim@cryptolab.co.kr
CryptoLab. Inc.

ABSTRACT
Bootstrapping, which enables the full homomorphic encryption scheme that can perform an infinite number of operations by restoring the modulus of the ciphertext with a small modulus, is an essential step in homomorphic encryption. However, bootstrapping is the most time and memory consuming of all homomorphic operations. As we increase the precision of bootstrapping, a large amount of computational resources is required. Specifically, for any of the previous bootstrap designs, the precision of bootstrapping is limited by rescaling precision.

In this paper, we propose a new bootstrapping algorithm of the Cheon-Kim-Kim-Song (CKKS) [11] scheme to use a known bootstrapping algorithm repeatedly, so called Meta-BTS. By repeating the original bootstrapping operation twice, one can obtain another bootstrapping with its precision essentially doubled; it can be generalized to be \( k \)-fold bootstrapping operations for some \( k > 1 \) while the ciphertext size is large enough. Our algorithm overcomes the precision limitation given by the rescale operation.

KEYWORDS
Fully Homomorphic Encryption, CKKS scheme, Approximate Bootstrapping, High Precision, Small parameters

1 INTRODUCTION
Homomorphic encryption (HE) refers to a class of encryption schemes that allows computing over encrypted data without decryption. Since Gentry et al. [17] introduced the first construction of FHE, there have been extensive studies [3–5, 8, 11–16, 18, 26] suggesting efficient designs and adding new functionalities of HE. Especially, the Cheon-Kim-Kim-Song (CKKS) [11] scheme is emerging as one of the notable HE schemes as it provides efficient privacy-preserving computation over real and complex numbers. To be specific, unlike other HE schemes that support exact computation over finite fields [3–5, 16] or of binary circuits [13–15], the CKKS scheme has complex vector space \( \mathbb{C}^{N/2} \) as its default message space and naturally supports approximated SIMD (Single Instruction/Multiple Data) computation over real and complex numbers.

In order to support complicated applications, HE should be equipped with faster homomorphic operations and should be able to evaluate circuits of huge depth. However, as homomorphic operations progress, the ciphertext modulus decreases and finally becomes too small to sustain further operations in the CKKS scheme. To solve these problems, Gentry’s blueprint of bootstrapping provides the idea of homomorphic re-encryption of a ciphertext. Re-encryption of a ciphertext means making the ciphertext modulus larger again while retaining the encrypted message of the ciphertext, and the resulting ciphertext is ready for further homomorphic operations.

In the CKKS scheme, the standard way for re-encryption, or bootstrapping (ATS) of ciphertexts is to evaluate an integral modular reduction homomorphically. Although the CKKS scheme provides the addition and the multiplication of ciphertexts, it is highly nontrivial to represent such a modular reduction with these basic algebraic operations. This means the bootstrapping requires a complicated combination of these basic operations, and it causes a major running time degradation and consumes a large amount of ciphertext modulus after bootstrapping. In practice, this is a severe bottleneck to use HE for various applications.

Also, in real use of HE, there are many cases require high precision bootstrapping. Evaluating a circuit of large multiplicative depth, including ML/DNN training, statistics, and sorting, require high precision bootstrapping since the ciphertext loses precision with each multiplication and bootstrapping procedure.

Recently kim et al. [21] achieves high precision with small \( N \) by using blind rotation which used for FHEW/TFHE bootstrapping. More precisely, when \( N = 2^{13} \), their precision is 73.58 where the number of slot is \( 2^{19} \). However, unlike other CKKS bootstrapping algorithms, it does not support SIMD (Single Instruction Multiple Data) computation during bootstrapping procedure. They split a ciphertext by the number of slots, bootstrap each piece, and combine them back into a ciphertext. Such split increases memory and time consumption in proportion to the number of slots, making it difficult for this method to deal with a large number of data.

Therefore, to preserve SIMD computation, various studies [2, 6, 7, 9, 19, 20, 22–24] have been conducted to represent modular reduction as a polynomial approximation to improve running time performance and precision of bootstrapping. Despite these efforts, improvements on the bootstrapping precision are limited with respect to the ciphertext dimension \( N \), once we require a reasonable security level in practice, and it turns out that we cannot avoid from adopting larger parameters in order to obtain a bootstrapping with better precision.
To address these problems, we propose a novel bootstrapping algorithm, called Meta-BTS, which can be used to obtain higher precision on given parameters.

1.1 Overview of Our Algorithm

CKKS bootstrapping can be understood as a homomorphic evaluation of the decryption circuit to re-encrypt a ciphertext. In bootstrapping, a bootstrapping error is added because the modular reduction is only approximated by evaluating a polynomial. Here we briefly present the main idea of Meta-BTS algorithm.

(1) Extract bootstrapping error by subtracting the original ciphertext from the naively bootstrapped ciphertext.

(2) Bootstrap the extracted error.

(3) Subtract the bootstrapped error from the bootstrapped ciphertext.

By repeating this procedure, the bootstrapping error can be partially removed from the bootstrapped ciphertext and the resulting ciphertext is in higher precision.

Techniques We describe our new algorithm to achieve a 2n-bit precision bootstrapping BTS\(^{(2)}\) using an n-bit precision bootstrapping BTS\(^{(1)}\), as shown in Fig. 1. Before we define bootstrapping and its precision, we first define a ciphertext of HE. Let \(R = \mathbb{Z}[X]/(X^N+1)\) be the cyclotomic ring with a power-of-two dimension \(N\), and \(m \in \mathbb{C}^{N/2}\) be a message. We denote \(R_Q = R/QR\) for a positive integer \(Q\). When used in this context, we call \(Q\) a modulus. Let \(ct(m, q) \in R^2_Q\) denote a ciphertext whose modulus is \(q\) and the message is \(m\).

Now, we define bootstrapping precision as follows. By applying a bootstrapping to a ciphertext \(ct(2^n m, q)\), we obtain the new ciphertext \(ct_{BTS} = ct(2^n m + 2^{-n}e, Q_{rem})\) where the infinite norm of the message \(\|2^n m\|_\infty\) is less than or equal to \(2^n\) to apply the bootstrapping, \(Q_{rem}\) is the ciphertext modulus after bootstrapping, \(e \in \mathbb{C}^{N/2}\), and \(\|e\|_\infty\) is less than or equal to \(1\). The precision of the bootstrapping is defined to be \(r + n\).

2-fold Bootstrapping, the simplest case of Meta-BTS, consists of two bootstrapping steps: bootstrapping the ciphertext and bootstrapping the error.

Bootstrapping a ciphertext Given a ciphertext \(ct_1 = ct(2^m m, 2^n q)\) where \(\|m\|_\infty\) is less than or equal to \(1\), we rescale the ciphertext by \(2^n\). Then, we get the new ciphertext \(ct_1 = ct(m, q)\). Now, we apply an n-bit precision BTS\(^{(1)}\) to \(ct_1\) and obtain new ciphertext \(ct_2 = ct(m + 2^{-n}e_1, Q_{rem})\) where \(\|e_1\|_\infty\) is less than or equal to \(1\). To extract the bootstrapping error, we multiply \(ct_2\) by \(2^n\), and get new ciphertext \(ct_3 = ct(2^n m + e_1, Q_{rem})\). Since \(Q_{rem}\) is multiple of \(q\) in the CKKS scheme, we can compute the ciphertext of the bootstrapping error, \(ct_{error} = [ct_3]_q - [ct_2]_q = ct(e_1, q)\). Now, we get the ciphertext of the bootstrapping error \(e_1\).

Bootstrapping the error Now, we apply an n-bit precision BTS\(^{(1)}\) to \(ct_{error}\) and obtain new ciphertext \(ct_4 = ct(e_1 + 2^{-n}e_2, Q_{rem})\) where \(\|e_2\|_\infty\) is less than or equal to \(1\). Finally, we compute \(ct_5 = ct_3 - ct_4 = ct(2^n m + 2^{-n}e_2, Q_{rem})\). Since \(\|e_2\|_\infty\) is less than or equal to \(1\), the ciphertext \(ct_5\) still has \(2^n m\) as a message with n-bit precision and \(Q_{rem}\) is the ciphertext modulus after bootstrapping. Therefore, we get a 2n-bit precision BTS\(^{(2)}\) of using an n-bit precision BTS\(^{(1)}\).

In addition, Meta-BTS can easily be extended to iterate the basic BTS more than twice. The details on how to repeat multiple times of our algorithm are introduced in Section 3.

1.2 Our Contributions

We propose a Meta-BTS algorithm which enables us to obtain high precision bootstrapping by iterating lower precision bootstrapping. That is, given a FHE parameter, it is possible to construct a bootstrapping algorithm of which precision is even beyond the state-of-the-art limitation. More specifically, given an n-bit precision BTS\(^{(1)}\) with an input range of \([-1, 1]\), a kn-bit precision BTS\(^{(k)}\) can be obtained by repeating BTS\(^{(1)}\) \(k\) times for possible \(k\). Until now, the parameters of HE are optimized for a specific precision, and if a higher precision operation is desired, a user needs to make new parameters for a higher precision. But it is hard to do unless a user is an expert of HE. In contrast, Meta-BTS enables higher precision operations without changing parameters.

We propose new HE parameters with a much higher precision than existing one according to the ciphertext polynomial dimension \(N\) using Meta-BTS method. Since the previous methods use a large number of modulus to increase precision of bootstrapping, precision is very limited by the ciphertext polynomial dimension \(N\). The best known precision to be implemented was 15 [2], 45 [6], 100 [23] when \(N\) is of \(2^{15}, 2^{16}, 2^{17}\), respectively.

Our Meta-BTS can increase the bootstrapping precision by using a fixed bootstrapping algorithm several times, so we only need to increase the precision in the multiplication process except the bootstrapping algorithm. Using Meta-BTS, we can obtain a higher precision compared to the previous methods for each corresponding \(N\), as shown in Table 1. The upper bound of precision and the parameter sets for obtaining the high precision are described in Section 4.2, 5, respectively.

Table 1: Comparison of the bootstrapping precision of previous works and ours. depth denotes the number of possible multiplications after the bootstrapping algorithm.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>(N)</th>
<th>(\text{slot})</th>
<th>(\text{iter})</th>
<th>depth</th>
<th>Bit prec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMT+21 [2]</td>
<td>(2^{15})</td>
<td>(2^{14})</td>
<td>-</td>
<td>3</td>
<td>15.5</td>
</tr>
<tr>
<td>JM22 [6]</td>
<td>(2^{16})</td>
<td>(2^{3})</td>
<td>-</td>
<td>-</td>
<td>45</td>
</tr>
<tr>
<td>LLK+22 [23]</td>
<td>(2^{17})</td>
<td>(2^{3})</td>
<td>-</td>
<td>-</td>
<td>100.11</td>
</tr>
<tr>
<td>This work</td>
<td>(2^{15})</td>
<td>(2^{14})</td>
<td>3</td>
<td>1</td>
<td>48</td>
</tr>
</tbody>
</table>

Our Meta-BTS algorithm also improves asymptotic time complexity in the high precision bootstrapping algorithm. Our algorithm uses \(n/k\)-bit precision bootstrapping \(k\) times to obtain \(n\)-bit precision bootstrapping. The advantage of small precision bootstrapping is that small precision bootstrapping uses small ciphertext modulus and performs fewer operations. If the target precision \(n\) is large enough, the ciphertext modulus for an \(n\)-bit precision bootstrapping is approximated by \(O(n^{5/4})\).
Let $C_n$ be the the number of multiplications used by $n$-bit precision BTS, which is an incremental function for $n$. Since the time complexity of $m$-bit number multiplication is at least $O(m \log m)$, we assume of it as $O(m \log m)$, where our algorithm is least efficient. Using Meta-BTS, the time complexity of an $n$-bit precision bootstrapping decrease from $C_n \cdot O(n^{5/4} \log n)$ to $C_{n/k} \cdot O \left( n^{9/4} \log \left( \frac{n}{k} \right) \right)$ for a positive integer $k$.

Our algorithm provides higher precision bootstrapping under the same parameters, so smaller parameters are available in certain situations. If one wants to obtain 100-bit precision FHE, one uses the ciphertext polynomial dimension $N = 2^{17}$, but our new method achieves the same precision with $N = 2^{16}$. According to Lattigo preset I [2] which uses $N = 2^{16}$, total memory for storing bootstrapping keys is at least 10.25GB. If $N$ grows to $2^{17}$, total memory is more than four times larger. So Meta-BTS can be useful in environments such as hardware or IoT with limited memory.

Finally, our new algorithm uses bootstrapping algorithm as a black box, so it can be used for all bootstrapping algorithms of the CKKS scheme. We adapt the existing bootstrapping algorithms for CKKS schemes [7, 9, 19] to build a high precision bootstrapping algorithm. If one applies the state-of-the-art bootstrapping algorithm [2, 6, 23], one may get a better bootstrapping algorithm in terms of precision using Meta-BTS.

1.3 Related Works

**Bootstrapping of the CKKS Scheme.**

The core of the bootstrapping algorithm is the polynomial approximation of a modular reduction. The first proposed method is to approximate the modular reduction with a trigonometric function and approximate it with a polynomial function using Taylor series [9]. After that, the studies [7, 19] were conducted to change Taylor series to Chebyshev interpolation. Then, a technique for direct approximation of the modular reduction was proposed using the least squares method [24] and Lagrange interpolation [20]. Also, to reduce the error caused by trigonometric approximation, the use of the inverse sine approximation was presented in [22] and the sine series was presented in [6]. Separately, to reduce the computation time for homomorphic linear transformation, the use of double hoisting technique was presented in [2]. Most recently, a method of minimizing the error variance in homomorphic computation and bootstrapping algorithm was proposed in [23].

**Difference of Our works.**

Our method mitigates the overall bootstrapping precision which is the combination of error coming from the approximate modular reduction and error from the CKKS scheme itself (e.g. rescale error). Many papers in the past have focused on improving modular reduction errors, and the recent work [23] proposes a first technique that considered the noise added during the bootstrapping procedure by the different homomorphic operations. As our paper takes a completely different point of view from other works, we use a bootstrapping algorithm in a black-box manner. We defined bootstrapping precision in Definition 2.2 as the sup-norm of the difference between the two decrypted messages of the "original ciphertext" and the "bootstrapped ciphertext" respectively. We propose a method to obtain higher precision bootstrapping by repeating a given bootstrapping.

2 BACKGROUND

2.1 Notation

We denote vectors in lower-case bold face, e.g. $a$, and matrices in upper-case bold face, e.g. $A$. We denote the inner product of two vector by $\langle \cdot, \cdot \rangle$ or simply $\cdot$. For a real number $r$, $\lfloor r \rfloor$ denotes the rounding function. We denote by $\lfloor \cdot \rfloor_q$ the modular reduction by $q$. We denote the infinite norm by $\| \cdot \|_\infty$ and Hadamard multiplication by $\circ$. We use $x \leftarrow D$ to denote the sampling $x$ according to distribution $D$. When a set $S$ is used instead of a distribution, $x \leftarrow S$ means that $x$ is sampled uniformly at random among elements of $S$. We set $\lambda$ to be a fixed security parameter throughout the paper: all known valid attacks against the cryptographic scheme under scope should take $\Omega(2^\lambda)$ bit operations.

Let $R = \mathbb{Z}[X]/(X^N + 1)$ be the ring of integers of the $2N$-th cyclotomic field with a power-of-two dimension $N$ and we write $R_q = R/qR$. A polynomial $a(X)$ can be denoted by $a$ by omitting $X$.

2.2 The CKKS scheme

We first recap the FHE scheme CKKS [11] and its packing methods.
The CKKS scheme uses a complex vector as a message (i.e. \( m \in \mathbb{C}^{N/2} \)) and provides homomorphic SIMD operations of ciphertexts (i.e. entry-wise operations) such as addition, subtraction, and Hadamard multiplication. To encrypt a vector of complex numbers \( m \in \mathbb{C}^{N/2} \), we use an isomorphism \( \tau : \mathbb{R}[X]/(X^N + 1) \rightarrow \mathbb{C}^{N/2} \) called the canonical embedding and a positive real number \( \Delta \) called scaling factor. A method of encoding/decoding a message is as follows:

- **Encoding**: \( m(X) \leftarrow \text{Ecd}(m, \Delta) \). Given a message \( m \in \mathbb{C}^{N/2} \) and a scaling factor \( \Delta \), the encoding map returns \( m(X) = [\Delta \cdot \tau^{-1}(m)] \in \mathbb{R} \). When encoding, \(|m| \leq 1\).

- **Decoding**: \( m \leftarrow \text{Decd}(m(X), \Delta) \). Given a plaintext \( m(X) \in \mathbb{R} \) and a scaling factor \( \Delta \), the decoding map returns \( m = \tau^{-1}(m(X)) \in \mathbb{C}^{N/2} \), where \( m'(X) = \Delta^{-1} \cdot m(X) \) is considered as an element of \( \mathbb{Q}[X]/(X^N + 1) \).

The CKKS scheme Consider the CKKS scheme \( \text{Enc} : M \times \mathcal{K} \rightarrow C \) for the plaintext space \( M = \mathbb{R} \), the key space \( \mathcal{K} \) and the ciphertext space \( C = \mathbb{R}^2_Q \). The ciphertext modulus \( Q \) is one of the positive integers \( Q_0 < Q_1 < \cdots < Q_{L} \) depending on the level of the ciphertext, which decreases over the homomorphic computations from \( L \) to 0. When the level becomes 0, homomorphic multiplication can no longer be performed on the ciphertext. To enable extra homomorphic multiplications, a bootstrapping algorithm is performed to raise the modulus of the ciphertext. In the bootstrapping algorithm, the level is recovered only up to some \( L_{\text{RTS}} < L \) due to the levels consumed during the homomorphic operations inside of the bootstrapping algorithm. The distributions \( \chi_{\text{enc}} \) and \( \chi_{\text{err}} \) denote the discrete Gaussian distribution with some fixed standard deviation. The distribution \( \chi_{\text{key}} \) outputs a polynomial with coefficients in \{-1, 0, 1\}.

### Basics Operations of the CKKS scheme

- **Setup**: \( \text{params} \leftarrow \text{FHE.Setup}(\lambda) \). Take the security parameter as an input and return the public parameters such as the degree \( N \) and the chain of modulus \( Q_0 < Q_1 < \cdots < Q_{L} \).

- **Key Generation**: \((\text{sk}, \text{pk}) \leftarrow \text{KeyGen}()\). Output a pair of secret and public key.
  - Sample \( s \leftarrow \chi_{\text{key}} \) and set the secret key as \( \text{sk} = (1, s) \). We denote by \( h = \text{wt}(s) \) the number of nonzero numbers of \( s \).
  - Sample \( a \leftarrow R_{Q_k} \) and \( e \leftarrow \chi_{\text{err}} \). Set the public key as \( \text{pk} = (b, a) \in R_{Q_k}^2 \) where \( b = [-a \cdot s + e]_{Q_k} \).

- **Switching Key Generation**: \( \text{swk} \leftarrow \text{KGen}_{\text{sk}}(s') \). For auxiliary modulus \( P = \prod_{i=0}^{L} p_i \), sample \( a'_{k} \leftarrow R_{PQ_k} \) and \( e'_{k} \leftarrow \chi_{\text{err}} \), output the switching key \( \text{swk} = (\text{swk}_0, \text{swk}_1) = (b'_k, a'_k) \in R_{PQ_k}^2 \) where \( b'_k = -a'_k s + e'_{k} + P \cdot s' \mod PQ_k \).
  - Set the evaluation key as \( \text{ekv} = \text{KGen}_{\text{sk}}(s_k^2) \).
  - Set the rotation key as \( \text{rk}_j = \text{KGen}_{\text{sk}}(s_k^3) \) for \( 1 \leq j \leq N/2 \).
  - Set the bootstrapping key as \( \text{btk} = (\text{rk}_j)_{j \in J} \) where \( J \) is a subset of \( \{1, 2N/2\} \).

- **Encryption**: \( \text{ct} \leftarrow \text{Enc}_{\text{pk}}(m(X)) \). Given a plaintext \( m(X) \in \mathbb{R} \), sample \( \nu \leftarrow \chi_{\text{enc}} \) and \( e_0, e_1 \leftarrow \chi_{\text{err}} \), output the ciphertext \( \text{ct} = [\nu \cdot \text{pk} + (m(X) + e_0, e_1)]_{Q_k} \). We call the message \( m \) for \( \text{Dcd}(m(X), \Delta) \).

- **Decryption**: \( m(X) \leftarrow \text{Dec}_{\text{ct}}(\text{ct}) \). Given a ciphertext \( \text{ct} \in R_{Q_k}^2 \), output \( m(X) = [(\text{ct}, \text{sk})]_{Q_k} \) where \( Q_0 \) is 0-level modulus.

- **Rescale**: \( \text{ct}_{\text{RS}} \leftarrow \text{RS}(\text{ct}, \text{ct}) \). For a given ciphertext \( \text{ct} \in R_{Q_k}^2 \), output \( \text{ct}_{\text{RS}} = [Q^{-1} \cdot \text{ct}] \mod Q_k/Q \) for some integer \( Q \) which is a factor of \( Q_k \). Rescale process plays two roles: reducing the size of the error and maintaining the scaling factor of each slot.

- **Addition/Subtraction**: \( \text{ct}_{\text{add}}/\text{ct}_{\text{sub}} \leftarrow \text{Add/Sub}(\text{ct}, \text{ct}') \). Given two ciphertext \( \text{ct}, \text{ct}' \in R_{Q_k}^2 \), output the ciphertext \( \text{ct}_{\text{add}}/\text{ct}_{\text{sub}} = ([\text{ct} \pm \text{ct}']_{Q_k} \) with the corresponding message vector \( m \pm m' \).

- **Multiplication**: \( \text{ct}_{\text{mul}} \leftarrow \text{Mult}_{\text{ekv}}(\text{ct}, \text{ct}') \). Given two ciphertexts \( \text{ct}, \text{ct}' \in R_{Q_k}^2 \), output a level-downed ciphertext \( \text{ct}_{\text{mul}} \in R_{Q_{k-1}}^2 \) with the corresponding message vector \( m \odot m' \).

- **Homomorphic evaluation**:
  \[ \text{ct} \leftarrow \text{Eval}(\text{C}, (\text{ct}_1, \cdots, \text{ct}_t), \{\text{swk}_k\}) \]

Given a circuit \( C \), a tuple of ciphertexts \( (\text{ct}_1, \cdots, \text{ct}_t) \) and the switching keys \( \{\text{swk}_k\} \) for computations, outputs an evaluated ciphertext \( \text{ct} \).

For the rest of the paper, we may denote the operations between ciphertexts or ciphertext and plain vector by common symbols, such as \( \text{Add}(\text{ct}_1, \text{ct}_2) = \text{ct}_1 + \text{ct}_2 \) or \( \text{Mult}(\text{ct}, \text{ct}') = \text{ct} \cdot \text{ct}' \) for simplicity. Also, we denote \( \text{ct}(m, q) \) by a ciphertext with the message \( m \in \mathbb{C}^{N/2} \) and the modulus \( q \).

### 2.3 Bootstrapping of the CKKS scheme

There have been extensive studies for the bootstrapping of the CKKS scheme. Although they vary in details, all of CKKS bootstrapping consists of four steps: StC, ModRaise, CTS, EvalMod. StC and CTS are specific linear transformations, which require rotation operations on encrypted vectors.

**Slot to Coefficient (StC).** In CKKS scheme, the canonical embedding is used to encode a complex vector to a polynomial which is an element of \( \mathbb{K} \). Homomorphic operations are performed slot-wise on \( \mathbb{C}^{N/2} \), but changing the modulus is only possible for coefficients of ciphertext polynomials. Therefore, for modulus raising, the homomorphic linear transformation is performed to move the value in the slot to the coefficient.

**Modulus raising (ModRaise).** ModRaise increase the ciphertext modulus to a larger modulus to recover the level of the ciphertext. Let \( \text{ct}(m, Q_i) = (b = -as + m(X) + e, a) \mod Q_i \) be a ciphertext where \( m(X) = \text{Ecd}(m, \Delta) \). By only changing its modulus, the ciphertext \( \text{ct}(m, Q_i) \) can be considered as \( \text{ct}(m', Q_k) \) for \( Q_k < Q_i \) where \( m' = \text{Dcd}(m(X) + Q_i l(X), \Delta) \) for some \( l(X) \in \mathbb{Z}[X]/(X^N + 1) \).

**Coefficient to Slot (CTS).** To compute modular reduction, a homomorphic linear transformation is performed to move the value in the coefficient back to the slot. Then each slot of the ciphertext has \( m_j + Q_i l_i \) as its encrypted message where \( m_j + Q_i l_i \) is the \( i \)-th coefficient of \( m(X) + Q_i l(X) \).

**Approximate Evaluation of the Modular Reduction (EvalMod).** An approximate evaluation of the modular reduction is performed in this step. The modular reduction cannot be accurately expressed.
only by addition and multiplication, but it can be approximated by evaluating a polynomial. Thus, the modular reduction is approximated through a polynomial homomorphic operation, where a bootstrapping error occurs due to approximation, which is greater than the rescale error that occurs during homomorphic operations. Various studies are being conducted to increase the precision of this polynomial approximation [6, 7, 9, 19, 22, 23].

**Definition 2.1. (BTS)** A bootstrapping algorithm is an algorithm that allows to re-encrypt a ciphertext and makes homomorphic operations sustainable in homomorphic encryption. A bootstrapping algorithm BTS is a PPT algorithm with the following properties:

- BTS(ct(m, q), btk) \(\rightarrow\) ct\(_{\text{BTS}}\) = ct(m', Q\(_{\text{rem}}\)): Given input keys for bootstrapping btk and a ciphertext ct(m, q) whose message m satisfies \(\|m\|_\infty \leq 2^r\) for some r, the bootstrapping algorithm returns a ciphertext ct\(_{\text{BTS}}\)(m', Q\(_{\text{rem}}\)) where Q\(_{\text{rem}}\) > q is a ciphertext modulus after the bootstrapping algorithm. The upper bound of \(\|m\|_\infty, 2^r\), is called the input bound of the BTS. When the input bound is 1, BTS is said to be standard.

**Definition 2.2. (PERF\(_{\text{BTS}}\))** Given a bootstrapping algorithm BTS, Performance PERF\(_{\text{BTS}}\) is the measure

\[\text{PERF}_{\text{BTS}} = (\text{Precision, RemainModulus, Time})\]

of BTS with the following properties:

- PERF\(_{\text{BTS}}\).Precision \(\rightarrow\) r + n: Returns the average number of bits preserved by BTS, n is the largest positive integer such that \(|m - m'|_\infty \leq 2^n\) holds without negligible probability where m' is the decrypted message of the bootstrapped ciphertext ct\(_{\text{BTS}}\) = BTS(ct(m, q)). \(2^r\) is the input bound of the BTS. That is, m is assumed to satisfy \(\|m\|_\infty \leq 2^r\).

- PERF\(_{\text{BTS}}\).RemainModulus \(\rightarrow\) Q\(_{\text{rem}}\)/q: Returns the modulus after bootstrapping except the minimal modulus for BTS. This means that the size of modulus that we can perform homomorphic operations excluding bootstrapping.

- PERF\(_{\text{BTS}}\).Time \(\rightarrow\) t: Returns the running time of BTS.

As bootstrapping consists of homomorphic operations, an error is added to the message of bootstrapped ciphertext. By Definition 2.1, 2.2, after bootstrapping, bootstrapped ciphertext contains a bootstrapped message m' = m + 2\(^{-n}\)e where m is the message of input ciphertext, \(\|m\|_\infty \leq 2^r\), e \(\in\mathbb{C}/\mathbb{N}\), and \(\|e\|_\infty \leq 1\).

### 3 THE META-BTS ALGORITHM

In this section, we present the Meta-BTS, a new method of combining bootstraps (in other words, iterating bootstraps) to get a higher precision bootstrap. We describe the algorithms of 2-fold bootstrapping and combining different bootstrapping. Since Meta-BTS use bootstrapping algorithm as a black box, it can be used even if a new bootstrapping algorithm is developed in the future.

#### 3.1 Difficulties in improving the performance of bootstrapping

In the CKKS bootstrapping, the modular reduction over encrypted data is required without decryption. Since the CKKS scheme provides only the multiplication and addition of ciphertexts, the modular reduction for the bootstrapping algorithms should be approximated by polynomials to compute over encrypted data. In this process, the bootstrapping error is caused by the homomorphic operations of the polynomial approximation as well as several rescalings with errors. In order to obtain high precision, one reduces the bootstrapping error by increasing a scaling factor \(\Delta\) and/or increasing the degree of polynomial approximation. Since the high precision bootstrapping algorithm consumes a lot of bits of the ciphertext modulus, it reduces the ciphertext modulus significantly after the bootstrapping and results in the large HE parameters.

#### 3.2 Algorithm description

In this subsection, we propose a new algorithm of increasing the bootstrapping precision by bootstrapping the errors caused by the bootstrapping algorithm. Furthermore, we propose a method of repeating the bootstrapping more than twice by generalizing our method. In case of using the RNS variant of CKKS[CTIE], every rescaling by \(2^n\) below means that we rescale the ciphertext by a modulus which is close to \(2^n\).

**3.2.1 2-fold Bootstrapping.** We propose an algorithm for constructing a 2n-bit precision bootstrapping BTS\(_{(2)}\) using an n-bit precision bootstrapping BTS\(_{(1)}\).

We start with a standard bootstrapping algorithm BTS\(_{(1)}\) with PERF\(_{\text{BTS}_{(1)}}\) = (n, Q\(_{\text{rem}}\)/t). In other words, BTS\(_{(1)}\) converts a ciphertext ct\((m, q)\) into ct\((m + 2^ne, Q_{\text{rem}})\) where \(\|m\|_\infty \leq 1\) and \(\|e\|_\infty \leq 1\).

**Algorithm 1: Construct BTS\(_{(2)}\) using BTS\(_{(1)}\)**

**Input:** ct\(_f\) = ct(2\(^n\)m, 2\(^{-n}\)

**Output:** ct\(_Q\) = ct(2\(^n\)m - 2\(^{-n}\)e\(_2\), Q\(_{\text{rem}}\))

1. \(c_{t_1}\) ← RS(2\(^n\), ct\(_f\)) \(\Rightarrow c_{t_1}\) = ct(m + e\(_{rs}\), q)
2. \(c_{t_2}\) ← BTS\(_{(1)}\)(ct\(_1\)) \(\Rightarrow c_{t_2}\) = ct(m + e\(_{rs}\) + 2\(^{-n}\)e\(_1\), Q\(_{\text{rem}}\))
3. ct\(_3\) ← 2\(^n\) \cdot ct\(_2\) \(\Rightarrow c_{t_3}\) = ct(2\(^n\)m + 2\(^{n}\)e\(_{rs}\) + e\(_1\), Q\(_{\text{rem}}\))
4. ct\(_4\) ← [ct\(_3\)]\(_2^n\) \(\Rightarrow c_{t_4}\) = ct(2\(^n\)m + 2\(^n\)e\(_{rs}\) + e\(_1\), 2\(^n\) \cdot q)
5. ct\(_5\) ← ct\(_4\) - ct\(_f\) \(\Rightarrow c_{t_5}\) = ct(2\(^n\)e\(_{rs}\) + e\(_1\), 2\(^n\) \cdot q)
6. ct\(_6\) ← BTS\(_{(1)}\)(ct\(_5\)) \(\Rightarrow c_{t_6}\) = ct(2\(^n\)e\(_{rs}\) + e\(_1\) + 2\(^{-n}\)e\(_2\), Q\(_{\text{rem}}\))
7. ct\(_Q\) ← ct\(_3\) - ct\(_6\)
8. return ct\(_Q\) = ct(2\(^n\)m - 2\(^{-n}\)e\(_2\), Q\(_{\text{rem}}\))

The BTS\(_{(2)}\) is as follows. Given an input ciphertext ct\(_f\) = ct(2\(^n\)m, 2\(^{-n}\)

where \(\|m\|_\infty \leq 1\), rescale ct\(_f\) to 2\(^n\) to reduce the size of message for applying BTS\(_{(1)}\). After rescale process, the ciphertext is ct\(_1\) = ct(m + e\(_{rs}\), q). Here, we assume that the rescale error e\(_{rs}\) is smaller than the error of the basic bootstrapping BTS\(_{(1)}\), so it satisfies \(\|e_{rs}\|_\infty \leq 2^{-n}\). By applying BTS\(_{(1)}\) to ct\(_1\), we get ct\(_2\) = ct(m + e\(_{rs}\) + 2\(^{-n}\)e\(_1\), Q\(_{\text{rem}}\)) where \(\|e_{1}\|_\infty \leq 1\). To extract the error, multiplying ct\(_2\) by 2\(^n\), we get ct\(_3\) = 2\(^n\) \cdot ct\(_2\) = ct(2\(^n\)m + 2\(^{n}\)e\(_{rs}\) + e\(_1\), 2\(^n\) \cdot q).

To calculate the two ciphertexts ct\(_3\), ct\(_5\), we make modulus of ct\(_3\) down. So, we obtain the ciphertext ct\(_4\) = [ct\(_3\)]\(_2^n\) = ct(2\(^n\)m + 2\(^{n}\)e\(_{rs}\) + e\(_1\), 2\(^n\) \cdot q). And then, computing ct\(_4\) - ct\(_f\), we get ct\(_5\) = ct\(_3\) - ct\(_f\) = ct(2\(^n\)e\(_{rs}\) + e\(_1\), 2\(^n\) \cdot q).

Now, we get the ciphertext ct\(_5\) whose message is the rescale and bootstrapping error 2\(^n\)e\(_{rs}\) + e\(_1\). Since \(\|2\(^n\)e_{rs} + e_{1}\|_\infty \leq 1\), we can apply BTS\(_{(1)}\) to ct\(_5\). By applying BTS\(_{(1)}\) to ct\(_5\), we get ct\(_6\) = ct(2\(^n\)e\(_{rs}\) + e\(_1\) + 2\(^{-n}\)e\(_2\), Q\(_{\text{rem}}\)) where \(\|e_{2}\|_\infty \leq 1\). Finally, computing ct\(_3\) - ct\(_6\), we get ct\(_Q\) = ct(2\(^n\)m - 2\(^{-n}\)e\(_2\), Q\(_{\text{rem}}\)).
Theorem 3.1 (2-fold Bootstrapping Algorithm). Given a standard bootstrapping algorithm BTS\(^{(1)}\) with performance

\[
\text{PERF}_{\text{BTS}\,(1)} = (n, Q_{\text{rem}}/q, t),
\]

there is a bootstrapping algorithm BTS\(^{(2)}\) whose performance is

\[
\text{PERF}_{\text{BTS}\,(2)} = (2n, Q_{\text{rem}}/(2^n \cdot q), 2t),
\]

if

\[
Q_{\text{rem}}/(2^n \cdot q) \geq 1.
\]

Proof. Given a ciphertext \(ct(2^nm, 2^n \cdot q)\) where \(m \in \mathbb{C}^{N/2}\) and \(\|m\|_\infty \leq 1\). According to Algorithm 1, the output ciphertext of BTS\(^{(2)}\) is \(ct(2^nm - 2^{-n}e_2, Q_{\text{rem}})\). Since \(2^{-n}e_2\) is the error of BTS\(_1\), \(e_2 \in \mathbb{C}^{N/2}\) and \(\|e_2\|_\infty \leq 1\). Therefore, BTS\(^{(2)}\) preserves 2n-bit precision of the message \(m\). Also, the minimal modulus for BTS\(_2\) grows from \(q\) to \(2^n \cdot q\) and the running time of BTS\(_2\) is almost \(2t\) because the running time of BTS is overwhelmingly large compared to other times. So, the performance of BTS\(^{(2)}\) is \((2n, Q_{\text{rem}}/(2^n \cdot q), 2t)\).

3.2.2 \(k\)-fold bootstrapping. We propose an algorithm for constructing a \(kn\)-bit precision BTS\(^{(k)}\) using an \(n\)-bit precision bootstrapping BTS\(^{(1)}\) for a positive integer \(k\).

Let BTS\(^{(1)}\) be a standard bootstrapping algorithm whose performance is \((n, Q_{\text{rem}}/q, t)\). Assume that BTS\(^{(k)}\) holds the following conditions for a positive integer \(k\).

- Input: \(ct((2^{(k-1)n})m, 2^{(k-1)n} \cdot q)\).
- Output: \(ct(2^{(k-1)n}m + 2^{(k-1)n}e_k, Q_{\text{rem}})\) where \(\|m\|_\infty, \|e_k\|_\infty \leq 1\).
- \(\text{PERF}_{\text{BTS}\,(k)} = (kn, Q_{\text{rem}}/(2^{(k-1)n} \cdot q), kt)\).

To prove above argument, we use mathematical induction for a positive integer \(k\). When \(k\) is 1, 2, we already propose the Algorithm 1 and BTS\(^{(1)}\), BTS\(^{(2)}\) satisfy all conditions by Definition 2.2 and Theorem 3.1. If we can construct BTS\(^{(k+1)}\) using BTS\(^{(k)}\) and BTS\(^{(1)}\) for a positive integer \(k\) and BTS\(^{(k+1)}\) satisfies all conditions, we can construct BTS\(^{(k)}\) which satisfies the all conditions using only BTS\(^{(1)}\) for a positive integer \(k\).

The BTS\(^{(k+1)}\) is as follows. Given an input ciphertext \(ct(I) = ct(2^km, 2^kn \cdot q)\) where \(\|m\|_\infty \leq 1\), rescale \(ct_I\) to \(2^n\) to reduce the size of message for applying BTS\(^{(k)}\). After rescale process, the
we can apply the bootstrapping algorithm. Meta-BTS algorithm also cannot grow \( k \) in BTS\(^{2}\). Standard bootstrapping algorithm computing \( c_t \) is smaller than the error of the basic bootstrapping \( \text{BTS}^{2} \) which satisfies the all conditions with just an \( n \) of bootstrapping. Even the best bootstrapping algorithm can vary between \( 6, 7, 9, 19, 20, 22–24 \), and each method has different performance and scaling factor. In this sense, \( \tau (m(X)) \) can be regarded as a slot-side value of the ciphertext that encrypts \( m(X) \). Let \( \tau (m(X)) \) be a scaled message of the ciphertext.

For example, when we multiply two ciphertexts without rescaling, the output ciphertext has a scaled message of \( \Delta \cdot m_1 \odot m_2 \). We rescale the ciphertext by \( \Delta \) to maintain scaling factor, and the scaled message becomes \( m_1 \odot m_2 \). Given a \( (r + n) \)-bit precision bootstrapping algorithm BTS with input bound \( 2^r \) and scaling factor \( \Delta \), we can understand BTS from the perspective of scaled message. BTS outputs \( c_t(m, q) \) to \( c_t(m + 2^r e, Q_{rem}) \) where \( ||m||_\infty \leq 2^r, ||e||_\infty \leq 1 \). In the perspective of scaled message, BTS outputs \( m \cdot \Delta \cdot (m + 2^r e) \).

**LEMMA 4.1 (STANDARDIZATION).** Let BTS be an \( r + n \)-bit precision bootstrapping algorithm with input bound \( 2^r \) and a scaling factor \( \Delta \). If one uses a scaling factor as \( \Delta = \Delta \cdot 2^r \) for encoding, one can construct a fully homomorphic encryption having BTS as its standard bootstrapping algorithm, retaining \( (r + n) \)-bit precision.

**Proof.** Let \( Q_{rem}/q \) be the remaining modulus of BTS. Let \( m \) be a message such that \( ||m||_\infty \leq 1 \). Given an input ciphertext \( c_t(m, q) \) with a scaling factor \( \Delta' \), it can be considered as \( c_t(2^r m, q) \) with a scaling factor \( \Delta \). After performing BTS in the initial sense, we get the output ciphertext \( c_t(2^r m + 2^r e, Q_{rem}) \) with a scaling factor \( \Delta' \). Again, the output can be considered as \( c_t(m + 2^r e, Q_{rem}) \). This gives a complete construction of a fully homomorphic encryption having BTS as its standard bootstrapping algorithm.

**4.2 Getting unbounded operations of high precision with fixed bootstrapping**

We analyze the upper bound of the precision which supports unbounded operations using our Meta-BTS algorithm. However, the CKKS bootstrapping can be implemented in various methods [2, 6, 7, 9, 19, 20, 22–24], and each method has different performance of bootstrapping. Even the best bootstrapping algorithm can vary...
depending on the desired precision. Therefore, we assume that the bootstrapping algorithm BTS is given, and compute the upper bound of the number of BTS iterations and the precision at this time.

Theorem 4.3 (Upper bound of the precision). Let BTS\(^{(1)}\) be a standard bootstrapping algorithm with performance \((n, Q_{rem}/q, t)\). Let \(N\) be the ciphertext polynomial dimension, and \(h\) be the number of nonzero coefficients in the secret key polynomial. Then the \(k\)-fold bootstrapping BTS\(^{(k)}\) can be constructed for

\[ k \leq \frac{\log(Q_{rem}/(q\sqrt{Nh})) + n}{2n}. \]

Let \(n_{\text{max}}\) be the maximum precision of BTS\(^{(k)}\) in order for ciphertexts to be multiplied indefinitely. Then \(n_{\text{max}}\) is given by

\[ n_{\text{max}} = \left\lfloor \frac{\log(Q_{rem}/q\sqrt{Nh}) + n}{2n} \right\rfloor \cdot n. \]

Proof. Suppose that the multiplicative depth which is the number of possible multiplication after BTS\(^{(k)}\) is \(1\) to compute the upper bound of the iteration number \(k\) and the precision \(n_{\text{max}}\). According to Theorem 3.2, we can construct a new bootstrapping algorithm BTS\(^{(k)}\) and its performance is \((kn, Q_{rem}/(2^{(k-1)n} \cdot q))\) with a positive integer \(k\). Then, we set the scaling factor \(\Delta\) as \(Q_{rem}/(2^{(k-1)n} \cdot q)\) and the precision after a ciphertext multiplication is approximated by \((\log(Q_{rem}/(2^{(k-1)n} \cdot q)) - \log \sqrt{Nh})\)-bit.

Until the precision after the bootstrapping is increased to the precision after a ciphertext multiplication, the ciphertext can continue homomorphic operations while maintaining the precision. So, for unbounded homomorphic operations, we obtain the following inequality.

\[ kn \leq \log(Q_{rem}/(2^{(k-1)n} \cdot q)) - \log \sqrt{Nh}, \]

\[ 2kn \leq \log(Q_{rem}/(q\sqrt{Nh})) + n, \]

\[ k \leq \frac{\log(Q_{rem}/(q\sqrt{Nh})) + n}{2n}. \]

Therefore,

\[ k \leq \frac{\log(Q_{rem}/(q\sqrt{Nh})) + n}{2n} \]

and

\[ n_{\text{max}} = \left\lfloor \frac{\log(Q_{rem}/q\sqrt{Nh}) + n}{2n} \right\rfloor \cdot n. \]

Based on this analysis, we introduce experiments to obtain a high precision using Meta-BTS according to \(N\) in section 5.

4.3 Accelerating bootstrapping with fixed precision

In this subsection, we analyze the bootstrapping time complexity when making an \(n\)-bit precision bootstrapping algorithm by repeating \(n/k\)-bit precision bootstrapping algorithm \(k\) times. The time complexity of \(m\)-bit integer multiplication is at least \(O(m \log m)\).

Before we analyze our Meta-BTS algorithm, we analyze the time complexity of an \(n\)-bit precision bootstrapping algorithm. A parameter set of HE such as the ciphertext polynomial dimension \(N\) and the number of nonzero numbers in the secret key \(h\) is given.

Also, for a fair comparison, we assume that modulus of a ciphertext after a bootstrapping is \(Q_{rem}\).

The CKKS bootstrapping consists of four step: StC, ModRaise, CtS, EvalMod. Since ModRaise does not use the level of a ciphertext and the number of levels to perform StC, CtS are not affected by the precision, we only define the level for EvalMod. Rest steps of BTS. Let \(d_{\text{EvalMod}}\) be the number of levels to perform EvalMod, rest steps. Then, the size of a ciphertext modulus used by an \(n\)-bit bootstrapping is \((d_{\text{EvalMod}} + d_{\text{rest}}) \cdot O(n + \log \sqrt{Nh})\)-bit where \(\sqrt{Nh}\) is added because of a rescale error. Therefore, the total size of a ciphertext modulus is \((d_{\text{EvalMod}} + d_{\text{rest}}) \cdot O(n + \log \sqrt{Nh}) + \log(Q_{rem})\)-bit.

Also, we define \(C_n\) as the the number of multiplications used by an \(n\)-bit precision bootstrapping with fixed \(N, h\). Since higher precision needs a higher degree of polynomial approximation, \(C_n\) is an incremental function for \(n\).

Lemma 4.4. The time complexity of an \(n\)-bit precision bootstrapping algorithm is at least \(C_n \cdot O(Q_n \log Q_n)\) where the \(Q_n\) is \((d_{\text{EvalMod}} + d_{\text{rest}}) \cdot O(n + \log \sqrt{Nh}) + \log(Q_{rem})\).

Proof. The size of a ciphertext modulus is \((d_{\text{EvalMod}} + d_{\text{rest}}) \cdot O(n + \log \sqrt{Nh}) + \log(Q_{rem})\). Let \(Q_n\) be \((d_{\text{EvalMod}} + d_{\text{rest}}) \cdot O(n + \log \sqrt{Nh}) + \log(Q_{rem})\). In the bootstrapping, since multiplication of \(Q_n\)-bit integer is performed \(C_n\) times, the time complexity is at least \(C_n \cdot O(Q_n \log Q_n)\).

Furthermore, \(d_{\text{EvalMod}}\) increase with precision because higher precision needs a higher degree of polynomial approximation. We consider only the cosine function among approximations of the trigonometric functions in EvalMod. The relationship between degree and bit precision (i.e. \(\log_2(||p(x) - \cos(mx)||_\infty)\)) is investigated when several cosine functions are approximated by minimax approximation \(p(x)\) using the Remez algorithm.

Figure 4: Precision and Degree of the Minimax Approximation of \(\cos(mx)\) by Remez algorithm.

As shown in Fig.4, the degree of approximate polynomial as proportional to the square root of bit precision. Therefore, we assume that the minimum polynomial degree to achieve \(n\)-bit precision
is approximately $O(\sqrt{n})$. Using the BSGS algorithm [2], a polynomial approximation can be computed only by square root of the polynomial degree multiplications. Thus, the number of ciphertext multiplication $d_{\text{EvalMod}}$ is roughly $O(n^{1/4})$. To sum up, in the $n$-bit precision bootstrapping, the total size of the ciphertext modulus is $O(n^{1/4}(n + \log \sqrt{Nh})) + d_{\text{rest}} \cdot O(n + \log \sqrt{Nh}) \log(Q_{\text{rem}})$.

**Lemma 4.5.** (Heuristic). If $n$ is large enough, the time complexity of an $n$-bit precision bootstrapping algorithm is at least $C_n \cdot O(n^{5/4} \log n)$.

**Proof.** Since the number of ciphertext multiplication $d_{\text{EvalMod}}$ is roughly $O(n^{1/4})$, the ciphertext modulus is approximated by $O(n^{1/4}(n + \log \sqrt{Nh})) + d_{\text{rest}} \cdot O(n + \log \sqrt{Nh}) \log(Q_{\text{rem}})$. If $n$ is large enough to ignore other constant values, the ciphertext modulus can be simplified to $O(n^{5/4})$. Therefore, the time complexity of an $n$-bit precision bootstrapping algorithm is at least $C_n \cdot O(n^{5/4} \log n)$. \hfill \square

Using our Meta-BTS algorithm, an $n$-bit precision bootstrapping can be constructed by repeating a $n/k$-bit precision bootstrapping $k$ times for some $k$. If $n$ is large enough, the time complexity of an $n$-bit precision bootstrapping is superlinear for $n$ due to the time complexity of multiplication. By Theorem 3.2, our Meta-BTS algorithm linearly increases the bootstrapping time with the number of iterations $k$. Therefore, Our Meta-BTS algorithm can improve the time complexity of high precision bootstrapping.

**Theorem 4.6.** (Heuristic). Given a parameter set of FHE. Assume that $n$ is large enough to ignore other parameters and the time complexity of $n$-bit integer multiplication is $O(n \log n)$.

When making an $n$-bit precision bootstrapping by repeating $n/k$-bit precision bootstrapping $k$ times using Meta-BTS, the time complexity of the $n$-bit precision bootstrapping is $C_{n/k} \cdot O\left(\left(\frac{n}{k}\right)^{5/4} \log \left(\frac{n}{k}\right)\right)$.

**Proof.** For the iteration number $k$, which $n/k$ is high enough, the time complexity of $n/k$-bit precision bootstrapping algorithm is $C_{n/k} \cdot O\left(\left(\frac{n}{k}\right)^{5/4} \log \left(\frac{n}{k}\right)\right)$ by Lemma 4.5.

When making an $n$-bit precision bootstrapping algorithm by repeating $n/k$-bit precision bootstrapping algorithm $k$ times, the time complexity of Meta-BTS is $k$ times the time complexity of $n/k$-bit precision bootstrapping algorithm by Theorem 3.2.

Therefore, using Meta-BTS, the time complexity of the $n$-bit precision bootstrapping decrease from $C_n \cdot O(n^{5/4} \log n)$ to $C_{n/k} \cdot O\left(\left(\frac{n}{k}\right)^{5/4} \log \left(\frac{n}{k}\right)\right)$. \hfill \square

5 CONCRETE PARAMETERS AND EXPERIMENTS
We provide a proof-of-concept implementation to show the performance of Meta-BTS. Our source code is developed in C++ with our HEaNN library. We summarize our optimization techniques, recommended parameter sets and some experimental results in this section. All experiments are conducted single-threaded on an Intel Xeon Gold 6242 at 2.8 GHz with 502GB of RAM running Linux. Our parameter sets satisfy at least 128-bit security according to Lattice-estimator [1] and known best attacks [10, 25].

Using Meta-BTS, one can repeat the existing bootstrapping algorithm to increase the bootstrapping precision without having to increase the size of primes for bootstrapping algorithm. Therefore, since we only need to increase the prime used in a ciphertext multiplication, we can obtain a higher precision than the existing algorithms on the same parameter.

5.1 A basic example
In this subsection, we provide a basic example of Meta-BTS. We use a parameter set which maintains an 8-bit precision as described in Table 2. In this parameter, the bootstrapping algorithm uses maximum 41-bit primes and the ciphertext multiplication uses 28-bit primes.

<table>
<thead>
<tr>
<th>log N</th>
<th>log QP</th>
<th>h</th>
<th>log q₁</th>
<th>depth</th>
<th>bit prec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>762</td>
<td>192</td>
<td>28-41</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

We apply Meta-BTS to this parameter and observe that the precision of a ciphertext increases with the number of iterations. All the parameters satisfy 128-bit security.

Table 3: Experimental result of Meta-BTS. $\log q_{\text{Mult}}$ denotes the bit lengths of individual primes which uses a ciphertext multiplication. $\text{depth}$ denotes the number of possible multiplications after the bootstrapping algorithm.

<table>
<thead>
<tr>
<th>iter</th>
<th>$\log q_{\text{Mult}}$</th>
<th>$\text{depth}$</th>
<th>bit prec.</th>
<th>boot time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28</td>
<td>6</td>
<td>9</td>
<td>7.86s</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>4</td>
<td>17</td>
<td>14.2s</td>
</tr>
<tr>
<td>3</td>
<td>44</td>
<td>3</td>
<td>25</td>
<td>19.2s</td>
</tr>
<tr>
<td>4</td>
<td>52</td>
<td>2</td>
<td>34</td>
<td>23.8s</td>
</tr>
</tbody>
</table>

Theoretically, the bootstrapping time for the $k$th iteration should be roughly $k$ times the initial bootstrapping time. However, it is actually smaller than the expected value, since we should reduce the number of multiplication primes in order to satisfy the same security level.

**Remark 1.** Even when parameter modification is not allowed, Meta-BTS could still be applied to achieve higher precision. We can group several multiplication primes together and consider them as a rescaling unit, enabling a high precision multiplication.

5.2 Achieving the maximum precision
In Table 1, we described the maximum precision using Meta-BTS. In this section, we propose the parameter sets obtaining the maximum precision.
Table 4: Overview of the base parameter for \( N = 2^{17} \). \( \log QP \), \( \log(q_i) \), and \( \log(p_j) \) denote the bit lengths of the largest RLWE modulus, individual RNS primes, and temporary primes for Modulus switching, respectively. \( \Delta_{Ecd} \) denotes the encoding scaling factor. \( L \) denotes the maximum ciphertext level, and \( h \) denotes the hamming weight of a secret key. Base, StC, Mult, Sine, and CTS denote slotoCoeff, multiplication, evalMod, and coeffToSlot primes, respectively. The left and right operands of the dot product denotes the number and the bit lengths of primes, respectively.

<table>
<thead>
<tr>
<th>( \log QP )</th>
<th>( \Delta_{Ecd} )</th>
<th>( L )</th>
<th>( h )</th>
<th>( \lambda )</th>
<th>boot time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2783</td>
<td>2(^{34})</td>
<td>35</td>
<td>128</td>
<td>&gt; 128</td>
<td>51.4s</td>
</tr>
</tbody>
</table>

\[
\begin{array}{|c|c|c|c|}
\hline
log(q_i) & log(p_j) \\
\hline
\text{Base} & \text{StC} & \text{Mult} & \text{Sine} & \text{CTS} \\
61 & 3 \cdot 54 & 16 \cdot 54 & 13 \cdot 61 & 3 \cdot 57 & 12 \cdot 61 \\
\hline
\end{array}
\]

5.2.1 Implementation for \( N = 2^{17} \). We construct a parameter for \( N = 2^{17} \), and achieve the maximum precision with actual implementation. First, we introduce the base parameter in Table 4.

We iterated Meta-BTS until we achieve the maximum precision described in theorem 4.3. Table 5 describes the implementation result including parameter constructions, ordered by number of iterations.

Table 5: Experimental result of Meta-BTS, achieving the maximum possible precision. Div denotes the primes reserved for dividing by \( 2^8 \).

<table>
<thead>
<tr>
<th>( \text{iter} )</th>
<th>( \log QP )</th>
<th>( \Delta_{Ecd} )</th>
<th>( \log(q_i) )</th>
<th>( \log(p_j) )</th>
<th>boot time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2783</td>
<td>2(^{34})</td>
<td>Div (^1)</td>
<td>16 \cdot 54</td>
<td>12 \cdot 61</td>
</tr>
<tr>
<td>2</td>
<td>2897</td>
<td>2(^{34})</td>
<td>1 \cdot 30</td>
<td>10 \cdot (2 \cdot 42)</td>
<td>14 \cdot 60</td>
</tr>
<tr>
<td>3</td>
<td>2777</td>
<td>2(^{114})</td>
<td>2 \cdot 30</td>
<td>7 \cdot (2 \cdot 57)</td>
<td>12 \cdot 61</td>
</tr>
<tr>
<td>4</td>
<td>2790</td>
<td>2(^{144})</td>
<td>3 \cdot 30</td>
<td>5 \cdot (3 \cdot 48)</td>
<td>13 \cdot 61</td>
</tr>
<tr>
<td>5</td>
<td>2735</td>
<td>2(^{174})</td>
<td>4 \cdot 30</td>
<td>4 \cdot (3 \cdot 58)</td>
<td>12 \cdot 61</td>
</tr>
<tr>
<td>6</td>
<td>2742</td>
<td>2(^{204})</td>
<td>5 \cdot 30</td>
<td>3 \cdot (4 \cdot 51)</td>
<td>13 \cdot 61</td>
</tr>
<tr>
<td>7</td>
<td>2868</td>
<td>2(^{234})</td>
<td>6 \cdot 30</td>
<td>3 \cdot (4 \cdot 59)</td>
<td>13 \cdot 61</td>
</tr>
<tr>
<td>8</td>
<td>2720</td>
<td>2(^{264})</td>
<td>7 \cdot 30</td>
<td>2 \cdot (5 \cdot 53)</td>
<td>13 \cdot 61</td>
</tr>
<tr>
<td>9</td>
<td>2810</td>
<td>2(^{294})</td>
<td>8 \cdot 30</td>
<td>2 \cdot (5 \cdot 59)</td>
<td>13 \cdot 61</td>
</tr>
<tr>
<td>14</td>
<td>2881</td>
<td>2(^{344})</td>
<td>13 \cdot 30</td>
<td>1 \cdot (9 \cdot 50)</td>
<td>14 \cdot 61</td>
</tr>
</tbody>
</table>

Let \( k \) be the number of iterations. According to theorem 4.3, 
\[
k \leq \left\lfloor \frac{\log(Qrem/q) - \log(\sqrt{nh}) + n}{2n} \right\rfloor = \left\lfloor \frac{16 \cdot 54 - 12 + 30}{60} \right\rfloor = 14,
\]
and the maximum possible precision is \( nk = 420 \) bits. The last parameter shown in Table 5 achieves the maximum precision. The actual bootstrapping time for this parameter is 903s, which is greater than the expected 51.4s \( \cdot 14 = 719.6s \). Due to the implementation issues, the maximum ciphertext level \( L \) increased, so the bootstrapping time increased more. When \( \text{iter} \) is 3 or 5, the bootstrapping time, 156s, 259s, is almost same as expected, 51.4s \( \cdot 3 = 154.2s \), 51.4s \( \cdot 5 = 257s \).

5.2.2 Construction for \( N = 2^{15}, 2^{16} \). We construct parameters for \( N = 2^{15}, 2^{16} \), that achieve the maximum precision. The Table 6, 7 describes the base parameters for the construction.

Table 6: Overview of the base parameter for \( N = 2^{15} \).

<table>
<thead>
<tr>
<th>( \log QP )</th>
<th>( \Delta_{Ecd} )</th>
<th>( L )</th>
<th>( h )</th>
<th>( \lambda )</th>
<th>boot time</th>
</tr>
</thead>
<tbody>
<tr>
<td>759</td>
<td>2(^{36})</td>
<td>14</td>
<td>192</td>
<td>&gt;128</td>
<td>6.43s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \log(q_i) )</th>
<th>( \log(p_j) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Base}</td>
<td>\text{StC}</td>
</tr>
<tr>
<td>49</td>
<td>2 \cdot 33</td>
</tr>
</tbody>
</table>

Table 7: Overview of the base parameter for \( N = 2^{16} \).

<table>
<thead>
<tr>
<th>( \log QP )</th>
<th>( \Delta_{Ecd} )</th>
<th>( L )</th>
<th>( h )</th>
<th>( \lambda )</th>
<th>boot time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1549</td>
<td>2(^{36})</td>
<td>29</td>
<td>192</td>
<td>128</td>
<td>16.3s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \log(q_i) )</th>
<th>( \log(p_j) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Base}</td>
<td>\text{StC}</td>
</tr>
<tr>
<td>49</td>
<td>3 \cdot 36</td>
</tr>
</tbody>
</table>

Table 8, 9 describes the maximum precision parameters constructed from the base parameters achieving 48, 255 bit precision, respectively.

Table 8: Overview of the high precision parameter for \( N = 2^{15} \).

<table>
<thead>
<tr>
<th>( \log QP )</th>
<th>( \Delta_{Ecd} )</th>
<th>( L )</th>
<th>( h )</th>
<th>( \lambda )</th>
</tr>
</thead>
<tbody>
<tr>
<td>755</td>
<td>2(^{58}) = 2(^{32}) \cdot 2(^{26})</td>
<td>15</td>
<td>192</td>
<td>&gt;128</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \log(q_i) )</th>
<th>( \log(p_j) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Base}</td>
<td>\text{StC}</td>
</tr>
<tr>
<td>49</td>
<td>2 \cdot 33</td>
</tr>
</tbody>
</table>

Table 9: Overview of the high precision parameter for \( N = 2^{16} \).

<table>
<thead>
<tr>
<th>( \log QP )</th>
<th>( \Delta_{Ecd} )</th>
<th>( L )</th>
<th>( h )</th>
<th>( \lambda )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1570</td>
<td>2(^{276}) = 2(^{240}) \cdot 2(^{36})</td>
<td>24</td>
<td>192</td>
<td>&gt;128</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \log(q_i) )</th>
<th>( \log(p_j) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Base}</td>
<td>\text{StC}</td>
</tr>
<tr>
<td>49</td>
<td>3 \cdot 36</td>
</tr>
</tbody>
</table>
5.3 An improved parameter set based on a set in [2]

Table 10 describes the overview of the parameter sets I, II and III suggested in [2]. Here $N$ denotes the dimension of the polynomial space, $\Delta_{EED}$ denotes the scaling factor of the input ciphertext, $\log(Q_{L-k}/q_0)$ denotes the remaining coefficient modulus excluding the base prime after bootstrapping, and $e$ denotes the precision of the bootstrapping. As you can see, Set III leaves twice as much remaining coefficient modulus as Set II, while providing more than half a precision compared to that of Set II and maintaining faster bootstrapping time.

<table>
<thead>
<tr>
<th>Set</th>
<th>$N$</th>
<th>$\Delta_{EED}$</th>
<th>boot time$^2$</th>
<th>$\log(Q_{L-k}/q_0)$</th>
<th>$\log(e^{-1})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>$2^{16}$</td>
<td>$2^{40}$</td>
<td>23.0s</td>
<td>360</td>
<td>25.7\mbox{(*mean)}</td>
</tr>
<tr>
<td>II</td>
<td>$2^{16}$</td>
<td>$2^{45}$</td>
<td>23.4s</td>
<td>225</td>
<td>31.5\mbox{(*mean)}</td>
</tr>
<tr>
<td>III</td>
<td>$2^{16}$</td>
<td>$2^{30}$</td>
<td>18.1s</td>
<td>450</td>
<td>19.1\mbox{(*mean)}</td>
</tr>
</tbody>
</table>

We can construct a new parameter set based on Set III, by taking Meta-BTS as its default bootstrapping method. The new set is more precise in terms of bootstrapping precision and faster in terms of amortized multiplication time, compared with Set II. Our new parameter set \textit{III’} is given as follows. Notations in Table 11 follow from Table 5 in [2].

<table>
<thead>
<tr>
<th>Table 11: Suggested parameter Set III’</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\log(QP)$</td>
</tr>
<tr>
<td>1545</td>
</tr>
<tr>
<td>$\log(q_1)$</td>
</tr>
<tr>
<td>Base + Div + Mult</td>
</tr>
<tr>
<td>55 + (13 + 43) + 9 · 43</td>
</tr>
</tbody>
</table>

Let us compare the performance of Set III’ with other parameter sets given in Table 10. Homomorphic operations except bootstrapping under Set III’ should be as fast as Set I since they have the same parameter structure. If we take Meta-BTS using \textit{Theorem 3.1} as the default bootstrapping strategy for Set III’, then we see that the running time for bootstrapping would be about 46 seconds which is twice of the bootstrapping running time of Set I. Since Set III’ has multiplicative depth of 10, which means we can do twice as much multiplications with Set III’ as with Set II, we see that the amortized multiplication of Set III’ is approximately the same as that of Set II.

6 CONCLUSION

In this paper, we presented the Meta-BTS algorithm for obtaining the high precision bootstrapping algorithm in the CKKS scheme. We provided the experimental results of our algorithms by implementing existing bootstrapping algorithm. The main contribution is to propose much higher precision after bootstrapping algorithm with fixed $N$ compared to prior works [2, 6, 7, 9, 19, 20, 22–24]. Our technique supported high-precision operations by repeating the bootstrapping algorithm even on a fixed parameter set. Finally, we proposed the parameter sets that can obtain higher precision according to $N$ than existing one, respectively. It is an open problem that applies the technique of obtaining high precision by repeating the bootstrapping algorithm in the BGV or BFV scheme with similar properties to the CKKS scheme.

REFERENCES


