

Enhanced CKKS Bootstrapping with Generalized Polynomial Composites Approximation

Seonhong Min
Seoul National University
Seoul, Republic of Korea
minsh@snu.ac.kr

Joon-Woo Lee*
Chung-Ang University
Seoul, Republic of Korea
jwlee2815@cau.ac.kr

Yongsoo Song*
Seoul National University
Seoul, Republic of Korea
y.song@snu.ac.kr

Abstract

Bootstrapping in approximate homomorphic encryption involves evaluating the modular reduction function. Traditional methods decompose the modular reduction function into three components: scaled cosine, double-angle formula, and inverse sine. While these approaches offer a strong trade-off between computational cost and level consumption, they lack flexibility in parameterization.

In this work, we propose a new method to decompose the modular reduction function with improved parameterization, generalizing prior trigonometric approaches. Numerical experiments demonstrate that our method achieves near-optimal approximation errors. Additionally, we introduce a technique that integrates the rescaling operation into matrix operations during bootstrapping, further reducing computational overhead.

CCS Concepts

• **Security and privacy** → **Public key encryption**.

Keywords

Homomorphic Encryption, Bootstrapping, CKKS (Cheon-Kim-Kim-Song)

ACM Reference Format:
Seonhong Min, Joon-Woo Lee, and Yongsoo Song. 2025. Enhanced CKKS Bootstrapping with Generalized Polynomial Composites Approximation. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '25), August 25–29, 2025, Hanoi, Vietnam.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3708821.3710824

1 Introduction

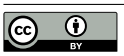
Homomorphic Encryption (HE) is an encryption scheme that allows an unlimited number of arbitrary function evaluations on encrypted data. Since Gentry's groundbreaking construction [12], numerous HE constructions have been developed based on the Learning with Errors (LWE) problem [24] and its ring variant, the Ring Learning with Errors (RLWE) problem [22]. The most practical HE schemes to date include BGV [4], BFV [11], GSW [13], TFHE/FHEW [9, 10], and CKKS [8].

Among these schemes, CKKS scheme possesses an intrinsic functionality for supporting fixed-point arithmetic in a single-instruction-multiple-data (SIMD) manner for real and complex data. However, as a leveled HE scheme, it can only evaluate circuits of limited depth, as each multiplication consumes one level. To enable further computation, a homomorphic evaluation of the decryption circuit, known as *bootstrapping*, is required to refresh the ciphertext in the lowest level. However, the bootstrapping process can consumes a significant amount of modulus. Consequently, substantial efforts have been made to reduce the overall modulus overhead of the bootstrapping. The primary bottleneck in modulus consumption lies in the polynomial approximation of the modulo function. Therefore, reducing the level consumption during the modulo function evaluation while optimizing the bootstrapping process is critical.

Let Q_0 denote the ciphertext modulus at level zero, Δ the scaling factor, and h the Hamming weight of the secret. When the modulus of the level-zero ciphertext is raised, the plaintext of the raised ciphertext becomes $m/Q_0 + I$, where m is a message satisfying $\|m\|_\infty \leq \Delta$ and $\|I\|_\infty < K = O(\sqrt{h})$. In this process, an auxiliary integer polynomial is added to the small scaled message coefficient after raising the modulus. During bootstrapping, the integer part of each coefficient must be removed to restore the message coefficient in the encrypted state. To achieve this, the CoeffToSlot operation is performed to convert the plaintext's coefficients into slots, then the function $t(x) = x - \lfloor x \rceil$ is homomorphically computed. Finally, the SlotToCoeff operation converts the slots back into coefficients. The $t(x)$ operation is non-arithmetic and must be approximated using a polynomial. This approximation significantly impacts bootstrapping precision, runtime, and total depth. Two approaches are commonly used for this approximation. One direction is to write $t(x)$ as a function related to trigonometric functions, then approximating it with polynomials. The other is to directly approximate $t(x)$ as a polynomial.

Approximation of $t(x)$ Many studies on CKKS bootstrapping [2, 15, 20] have decomposed $t(x)$ as a composition of trigonometric functions to reduce the number of the key-switching operations. For example, Han and Ki [15] approximated $t(x)$ by composing $\cos(\frac{2\pi}{2^\ell}(x-\frac{1}{4}))$ with ℓ double-angle formula iterations. This method derives the scaled sine function by dividing it by 2π, resulting in a function sufficiently close to $t(x)$ in regions near the integers. Building on this, Lee et al. [20] introduced an additional step involving the inverse sine function after the sine evaluation. This additional step completely removes the difference between $t(x)$ and the scaled sine function.

*Corresponding authors



This work is licensed under a Creative Commons Attribution 4.0 International License.

ASIA CCS '25, Hanoi, Vietnam

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1410-8/25/08
https://doi.org/10.1145/3708821.3710824

Alternatively, Lee et al. [21] proposed directly approximating $t(x)$ with a single polynomial to reduce the depth of bootstrapping. They also introduced an efficient polynomial evaluation algorithm called the *lazy-BSGS algorithm*, which minimizes the time-consuming key-switching operations, particularly for odd polynomials. Since $t(x)$ is an odd function, its direct approximation can be computed more efficiently. However, due to the high degree of the polynomial, this approach still requires a significant number of key-switching operations.

Although these methods effectively reduces homomorphic operations and time complexity, leveraging the decomposition of the target function $t(x)$ and the oddness of $t(x)$ for each, it remains unclear whether those two optimizations can be employed simultaneously. Therefore, combining both techniques could be a promising direction to further reduce time complexity.

Rescaling Operation As an independent issue in bootstrapping, several rescaling processes are required due to the multiple homomorphic multiplications involved. However, since the rescaling operation necessitates several NTT/INTT operations, it is beneficial to minimize the number of these operations during the rescaling process. In previous HE research, hoisting techniques have been proposed in various forms to reduce the number of operations in each homomorphic computation. The hoisting technique is a general approach that rearranges the order of sub-operation and merges operations with similar functionalities. Recently, to reduce the number of operations in bootstrapping, Bossuat et al. [2] proposed the double hoisting technique, which decreases the number of Decomp and ModDown processes during the CoeffToSlot and SlotToCoeff operations. Building on this, it would be valuable to devise another hoisting technique specifically aimed at reducing the number of operations in the rescaling process, particularly when additional homomorphic operations are applied just prior to the rescaling step.

1.1 Our Contribution

New Approximation Method for $t(x)$ In this paper, we generalize the sine-based approximation method to provide a more refined parameterization for CKKS bootstrapping. Our generalized approach decomposes the target function $t(x)$ into two functions: f_1, which performs the role of a scaled cosine function, and f_2, which incorporates both the double angle formula and inverse sine.

More specifically, we define f_1 as a function that maps the input intervals into r distinct intervals for a given parameter r. This heuristic is inspired by the scaled cosine function in previous sine-based methods, which also maps input intervals into multiple distinct intervals. While there are many possible choices for f_1 and f_2, we select f_1 as the (shifted) sine function, $f_1 = \sin(2\pi x/r)$, and f_2 as a composition of the r-th angle formula and the inverse sine function.

This generalization allows a more versatile choice of r compared to the previous methods, where r was restricted to power of two. Consequently, this flexibility enables a more adaptable selection of polynomial degrees in the EvalMod step. Additionally, the oddity and sparsity of the approximations of our chosen f_1 and f_2 reduce computational costs during the polynomial evaluation.

Furthermore, we numerically determine the lower bound of the approximation error for f_1 and demonstrate that our choice of f_1 is nearly optimal. Among the functions that map input intervals into r distinct intervals, there exists one with the minimal approximation error for a fixed polynomial degree. We identify a critical relationship between this 'most periodic function' and its best approximation polynomial. Using this relation, we compute the best periodic function and its corresponding approximation polynomial via a least-squares optimization problem in polynomial time, ultimately obtaining the lower bound of the approximation error.

Improved Linear Transformation We also improve the matrix-vector multiplication algorithm for leveled HE schemes. The main bottleneck in bootstrapping is CoeffToSlot, which involves a linear transformation, making the acceleration of matrix-vector multiplication critical for achieving better efficiency during bootstrapping. In prior works, the primary improvements focused on reducing the number of unit operations. For instance, the *hoisting* technique minimizes the number of gadget decompositions required when computing rotations of a single ciphertext [14], and the double hoisting baby-step giant-step (BSGS) matrix multiplication algorithm optimizes the number of divisions by special modulus [2].

Our approach introduces a new improvement by reducing the complexity of the unit operation itself. In our improved double hoisting BSGS algorithm, the rescaling operation is integrated into the matrix multiplication process, rather than being performed at the end of the algorithm. This modification exploits the efficiency of the NTT at the lower levels. The proposed method improves the theoretical complexity compared to prior work by a factor of approximately $(\mathcal{L}+\mathcal{N})/\mathcal{L}$, where $\mathcal{L}$ represents the number of the prime factors of the modulus at the current level, and $\mathcal{N}$ denotes the number of the prime factors divided during the algorithm.

1.2 Related Works

There have been a few attempts to reduce modulus consumption during bootstrapping that do not rely on polynomial approximation. Kim et al. [19] proposed using a smaller scaling factor for CoeffToSlot, based on the idea that it is sufficient to correctly extract the I term for bootstrapping. Bae et al. [1] introduced a high-precision bootstrapping method with shallow depth by iteratively bootstrapping from the highest to the lowest bits of the message through low-precision bootstrapping.

Meanwhile, the cumbersome key-switching operation in the CKKS scheme has been accelerated via algorithmic techniques. Kim et al. [18] proposed a space-time complexity tradeoff method that leverages the smallness of the gadget decomposition. This method reduces the cost of NTT-based computations from quadratic to linear, thereby accelerating key-switching at higher levels. On the other hand, the key-switching at lower levels has been improved by introducing the unused upper bits of the modulus into the special modulus without requiring additional key-switching keys [16], achieved through a simple modification to the gadget decomposition.

We remark that all these improvements are orthogonal to our approach and can be integrated with our proposed method as well.

2 Background

2.1 Notations

Let N be a power of two and Q be an integer. $R = \mathbb{Z}[X]/\Phi_{2N}(X) = \mathbb{Z}[X]/(X^N + 1)$ denotes the $2N$-th cyclotomic polynomial ring. $\mathcal{F}(a, b)$ denotes a set of arbitrary functions defined over the interval $[a, b]$. For a set S, we denote the cardinality of the set by $|S|$. We denote a uniform distribution over a set S by $\mathcal{U}(S)$. If S is a finite interval $[a, b]$, we write $\mathcal{U}(a, b)$ to denote a uniform distribution over $[a, b]$. Rotation of a vector $\mathbf{v}$ by index i is denoted by $\mathbf{v}^{(i)}$. i-times of iterative composition of a function f is denoted by $f^{\circ i}$, *i.e.*, $f^{\circ i} = f \circ f \circ \cdots \circ f$.

2.2 Homomorphic Encryption for Arithmetic of Approximate Numbers (CKKS)

The Cheon-Kim-Kim-Song (CKKS) homomorphic encryption scheme is a homomorphic encryption scheme that supports arithmetic operations on real and complex numbers [8]. Since the CKKS scheme operates on data with approximation noise, it does not remove the noise added for security during decryption of the ciphertext.

The CKKS scheme supports single-instruction-multiple-data (SIMD) operations. In other words, a vector containing multiple data elements is encrypted into a single ciphertext, and each homomorphic operation is applied to all encrypted data in the ciphertext simultaneously. Each element in the vector within the ciphertext is referred to as a slot. The maximum number of slots is $N/2$, where N is the degree of the base ring $R_q = R/qR$ in the ciphertext. If the number of required data elements is smaller than $N/2$, a smaller number of slots can be used to achieve more efficient bootstrapping, and this number is set to a power-of-two integer. We denote the number of slots as n.

To encode a vector of real or complex approximate numbers, we first convert the vector into a polynomial with real coefficients and multiply it by a large constant then round to integer. For the $2N$-th root of unity ζ, we find a ring polynomial $m(X) \in \mathbb{R}[X]/(X^N + 1)$ such that $m(\zeta^{5^j}) = v_{[j]_n}$ and $m(\overline{\zeta^{5^j}}) = \overline{v_{[j]_n}}$ for $(0 \le j < N/2)$, given a vector $\mathbf{v} = (v_i)_{i\in[n]}$. The polynomial $m(X)$ is then scaled by a sufficiently large constant and rounded to an integer polynomial, which resides in R and is ready to be added to an empty ciphertext. This large scaling factor determines the precision of the data and is denoted as Δ. We note that the encoded vector can be 'rotated' slotwise using automorphisms $\psi_i : X \mapsto X^{5^i}$.

One of the core subroutines in homomorphic operations is the key-switching operation. Assume the input is a ring element c, and the old and new secret keys are s and s', respectively. The output of the key-switching operation is a pair of ring elements (b, a) such that $c \cdot s = b + a \cdot s' + e$, where e is a small noise. This operation does not use the secret keys directly but requires the key-switching key swk. The key-switching operation is essential for homomorphic multiplication between ciphertexts, rotations, and conjugations, and it constitutes the majority of the overall computational cost in homomorphic computations. Therefore, reducing the number of key-switching operations is crucial for improving the efficiency of homomorphic computations.

Now, we describe the core algorithms of CKKS cryptosystem below.

- $\mathsf{Ecd}(\mathbf{v} = (v_i)_{i\in[n]} \in \mathbb{C}^n, \Delta) \to m$: Compute $\tilde{m} \in \mathbb{R}[X]/(X^N + 1)$ such that $m(\zeta^{5^i}) = v_{[i]_n}$ and $m(\overline{\zeta^{5^i}}) = \overline{v_{[i]_n}}$ for $i \in [N]$. Then output $m = \lfloor \Delta \cdot \tilde{m} \rceil \in R$.
- $\mathsf{Dcd}(m \in R, \Delta) \to \mathbf{v}$: Compute $\tilde{m} = m/\Delta \in \mathbb{R}[X]/(X^N + 1)$ and output $\mathbf{v} = (\tilde{m}(\zeta^{5^i}))_{i\in[n]}$.
- $\mathsf{Enc}(m \in R, \mathsf{pk} = (b_{\mathsf{pub}}, a_{\mathsf{pub}}) \in R_{Q_L}, \ell) \to (b, a)$: Sample $v, e_0, e_1 \leftarrow \chi$ and compute $(b, a) = v \cdot ([b_{\mathsf{pub}}]_{Q_\ell}, [a_{\mathsf{pub}}]_{Q_\ell}) + (m + e_0, e_1) \in R_{Q_\ell}$ where χ is the error distribution over R. Output $\mathsf{ct} = (b', a') \in R^2_{Q_\ell}$.
- $\mathsf{Dec}(\mathsf{ct} = (b, a) \in R^2_{Q_\ell}, s, \ell) \to m$: Output $m = b + a \cdot s$.

2.3 Gadget Decomposition and Rotation Operation

Given an RLWE ciphertext $(b, a) \in R_Q$ under secret t, the key-switching is an operation which homomorphically computes $b + at$ using an encryption of the previous key t under new key s. This can be realized with two common toolkits for lattice-based cryptography, the *gadget decomposition* and the *special modulus* [4, 8, 9, 11]. We begin by explaining the gadget decomposition briefly. Given a modulus Q and a fixed vector $\mathbf{g} = (g_i)_{1\le i\le\beta} \in R^\beta$ called the *gadget vector*, the gadget decomposition is a function $h : R_Q \mapsto R^\beta_Q$ which satisfies the following conditions for any $a \in R_Q$, its decomposition $h(a) := \mathbf{a} = (a_i)_{1\le i\le\beta} \in R_Q$.

(1) $\langle \mathbf{a}, \mathbf{g} \rangle = \sum_{i=1}^{\beta} a_i \cdot g_i = a \pmod{Q}$
(2) $\|a_i\|_\infty \le B_{\mathbf{g}}$ for some small $B_{\mathbf{g}}$.

Using these two properties of gadget decomposition, we can define the key-switching key $\mathsf{ksk}_{t\to s}$ and the key-switching operation as follows, for a *special modulus* P.

- $\mathsf{ksk}_{t\to s} = (\mathbf{u}_0, \mathbf{u}_1)$ where $\mathbf{u}_0 = -s\cdot\mathbf{u}_1 + P\cdot t\cdot\mathbf{g} + \mathbf{e} \in R^\beta_{PQ}$, $\mathbf{u}_1 \leftarrow \mathcal{U}(R_{PQ})^\beta$ and $\mathbf{e} \leftarrow \chi^\beta$ for some small error distribution χ.
- $\mathsf{KeySwitch}((b, a), \mathsf{ksk}_{t\to s}) \to (b', a')$: Compute $(b', a') = (b + \lfloor P^{-1} \cdot v_0 \rceil, \lfloor P^{-1} \cdot v_1 \rceil)$ for $(v_0, v_1) = h(a)^\top \cdot \mathsf{ksk}_{t\to s}$.

It can be easily checked that this key-switching algorithm is valid. In the following equations, e_{rnd} denotes the rounding error.

$$\begin{aligned} b' + a' \cdot s &= b + \frac{1}{P}\langle h(a), \mathbf{u}_0 + s \cdot \mathbf{u}_1 \rangle + e_{rnd} \pmod{Q} \\ &= b + \frac{1}{P}\langle h(a), P \cdot t \cdot \mathbf{g} + \mathbf{e} \rangle + e_{rnd} \pmod{Q} \\ &= b + a \cdot t + \frac{1}{P}\langle h(a), \mathbf{e} \rangle + e_{rnd} \pmod{Q} \end{aligned}$$

Therefore, the final error $e_{rnd} + \frac{1}{P}\langle h(a), \mathbf{e} \rangle$ is bounded by $\beta N B_{\mathbf{g}}/P \cdot \|\mathbf{e}\|_\infty + \|e_{rnd}\|_\infty$. Consequently, setting P sufficiently big, the key-switching operation can be executed with noise with only a size of the rounding error.

In CKKS, the homomorphic rotation operation can be realized with the key-switching operation. To be precise, given a ciphertext $(b, a) \in R_Q$ under secret t, the rotation by index i can be computed as follows:

$$\mathsf{KeySwitch}((\psi_i(b), \psi_i(a)), \mathsf{ksk}_{\psi_i(t)\to t}).$$

2.4 Bootstrapping of CKKS

The bootstrapping for CKKS scheme was first introduced in [6], which aims to increase the level of the ciphertext for the sake of further computations. Bootstrapping of CKKS scheme consists of five steps: ModRaise, SubSum, CoeffToSlot, EvalMod, SlotToCoeff.

Modulus Raising (ModRaise**):** To bootstrap a CKKS encryption $\mathsf{ct} = (b, a)$ of a message m, we first ensure the ciphertext is at level 0 with ciphertext modulus Q_0. Since $b + a \cdot s \approx m \pmod{Q_0}$, where s denotes the secret, $b + as \approx m + Q_0 I(X)$ for some polynomial $I(X) \in R$. It is worth noting that $\|I(X)\|_\infty < K$ for some integer K, due to the smallness of the secret s. When the ciphertext level is raised to L with ciphertext modulus $Q_L \gg Q_0$, the resulting ciphertext ct encrypts $m + Q_0 I$. In the subsequent bootstrapping steps, this additional $Q_0 I(X)$ factor is removed by evaluating the modulo function homomorphically.

Trace Evaluation (SubSum**):** This step is necessary only when the ciphertext is sparsely packed. In such cases, the plaintext resides in the subring $\mathbb{Z}[Y]/(Y^{2n}+1)$, where n is the number of the slots, and $Y = X^{N/2n}$. However, $Q_0 I$ factor is not an element of the subring. To address this, the $(N/2n)i$-th coefficients, for $0 \le i < 2n$, must be extracted homomorphically. This can be achieved efficiently by evaluating a trace-like map. After this step, we obtain $m(Y) + Q_0 \tilde{I}(Y) \in \mathbb{Z}[Y]/(Y^{2n}+1)$.

Linear Transformation (CoeffToSlot**):** Observe that the additional $Q_0 I$ part is added to the coefficient, not the message itself. Therefore, to evaluate modulo function in SIMD manner, the coefficients needs to be moved to the slots. This step involves a homomorphic evaluation of the encoding algorithm, which is a matrix multiplication between the message vector and the encoding matrix.

Modulo Function Evaulation (EvalMod**):** After the linear transformation, we obtain one or two ciphertexts encrypting the coefficients of the polynomial $m/Q_0 + I$. As described earlier, the goal is to evaluate the modulo function $t(x) = x - \lfloor x \rceil$ homomorphically over the interval $[-K, K]$. Since $t(x)$ is neither a polynomial nor an analytic function, it must be approximated using a polynomial. For better efficiency, we assume the message m is sufficiently close to the origin, i.e. the ratio between m and Q_0 is small. This assumption allows the modulo function to be approximated with a lower-degree polynomial, improving time complexity.

Inverse Linear Transformation (SlotToCoeff**):** After evaluating the modulo function, the ciphertext(s) encrypt the coefficients of m with a few remaining levels. To complete the bootstrapping, the coefficients are moved back from the slots to the coefficient representation. This step involves multiplying the decoding matrix to the ciphertext(s), mirroring the CoeffToSlot step.

The most challenging aspect of CKKS bootstrapping is the approximation of modulo function. Two primary methods are used:direct approximation [21] and sine-based approximation [3, 5, 6, 15, 17, 20]. Let K represent the probabilistic bound of $\|I\|_\infty$ from ModRaise and ε denote the ratio between the bound of the message and the modulus Q_0 at the lowest level. In direct approximation method, the target function $t(x) = x - \lfloor x \rceil$ is directly approximated over the $2K-1$ intervals $(i-\varepsilon, i+\varepsilon)$ for $-K < i < K$, using least-squares approximation. While this approach achieves low approximation error, it requires a high-degree polynomial, resulting in significant computational complexity. The sine approximation method reduces computational complexity by sacrificing some multiplicative levels. In sine method, the target function $t(x)$ is decomposed into three functions: a scaled cosine, double-angle formulae and (optionally) an inverse sine function. Specifically, the scale cosine $\cos(\frac{2\pi(x-0.25)}{r})$ is evaluated first, where r is a power-of-two. Then the double-angle formula is applied to the result $\log r$ times to obtain $\sin(2\pi x)$. Finally, $\frac{1}{2\pi}\arcsin$ is evaluated to obtain $t(x) = x - \lfloor x \rceil$. Each polynomial in this decomposition plays a distinct role. Firstly, the scaled cosine maps the $2K-1$ intervals into r disjoint intervals. Then, the double-angle formulae maps these r intervals into a single interval. Finally, the inverse sine compensates for the difference between the sine function and modulo function.

3 New Approach to Approximate Polynomial for EvalMod

In this section, we propose a new approximation method for decomposing the target function $t(x)$ in CKKS bootstrapping, generalizing prior methods. Our approach enables improved parameterization and benefits from the efficient evaluation of odd functions, as highlighted in [21]. Additionally, we numerically demonstrate that the approximation error of our method is near-optimal among approaches with a similar pipeline.

3.1 Our Method

The primary objective of EvalMod is to approximate the modular reduction function $t(x)$ with polynomials over the input interval $\cup_{-K<i<K}(i-\varepsilon, i+\varepsilon)$, where ε is the message bound, and K is the probabilistic bound. In previous approximation methods using trigonometry functions, the target function $t(x)$ is decomposed into three functions: scaled cosine, double-angle formula and inverse sine. More precisely, we can write $t(x) = t_3 \circ t_2^{\circ(\log r)} \circ t_1(x)$ where $t_3(x) = \frac{1}{2\pi}\sin^{-1}(x)$, $t_2(x) = 2x^2 - 1$ and $t_1(x) = \cos\left(\frac{2\pi(x-1/4)}{r}\right)$, for some small power-of-two $r > 0$. Generally, as r increases, the scaled cosine function $\cos(2\pi x/r)$ can be approximated with lower-degree polynomials [1]. Consequently, larger values of r reduce the arithmetic operations required to compute $t(x)$. However, to avoid wasting multiplicative levels when computing the r-th angle formula, r must be a power-of-two. Similarly, the inverse sine function is often over-approximated due to its level consumption. These constraints limit the flexibility of parameterization in this approach.

To address these limitations, we decompose the modular reduction function $t(x)$ into two functions, *i.e.,* $t(x) = f_2(f_1(x))$. In our method, $f_1(x)$ serves a similar purpose to the scaled cosine function in the sine method: it maps the approximation intervals into r disjoint intervals. Subsequently, $f_2(x)$ maps these r intervals into a single interval, ensuring that $f_2 \circ f_1(x) = t(x)$. Notably, $f_2(x)$ effectively combines the roles of the double angle formula and the inverse sine function. While there are numerous potential choices for f_1 and f_2, we select trigonometric functions to exploit their simplicity and suitability for this task. Below, we detail our specific choices for f_1 and f_2, taking into the oddity of r.

Odd r. If r is odd, we choose $f_1(x) = \sin(2\pi x/r)$. It is straightforward to show that f_1 is an odd function since $\sin(-2\pi x/r) = -\sin(2\pi x/r)$. Next, we demonstrate that f_1 maps the input approximation intervals into r intervals. As f_1 is periodic with a period of r, it suffices to show that $f_1([-\varepsilon, \varepsilon]), \ldots, f_1([r-1-\varepsilon, r-1+\varepsilon])$ do not intersect. Given the smallness of ε, we only need to verify that there is no repeating values in the sequence $\sin(2\pi \cdot 0/r), \sin(2\pi \cdot 1/r), \ldots, \sin(2\pi \cdot (r-1)/r)$. Suppose $\sin(2\pi \cdot i/r) = \sin(2\pi \cdot j/r)$ for some $0 \le i \ne j < r$. This implies that $2\pi \cdot j/r + 2\pi \cdot i/r$ is either π or 3π. Simplifying this relation, we find that $(i+j)/r = \frac{1}{2}$ or $\frac{3}{2}$. However, due to the oddness of r, such integers i and j cannot exist, proving our claim.

For f_2, we choose it as a composition of the r-th angle formula for sine and the inverse sine, scaled by $\frac{1}{2\pi}$. Applying the r-th angle formula to f_1 results in $\sin(2\pi x)$. Evaluating the inverse sine then returns $[2\pi x]_{2\pi}$, and scaling it by $\frac{1}{2\pi}$ gives $t(x)$. Notably, f_2 is an odd function since both the r-th angle formula and the inverse sine are odd functions.

Even r. When r is even, $f_1(x) = \sin(2\pi x/r)$ cannot be used directly as in the odd r case, because the output intervals of f_1 would intersect. Instead, we shift the input by $\frac{1}{4}$, choosing $f_1(x) = \sin(2\pi(x+0.25)/r)$. As with the odd r case, this function is periodic with a period of r. To prove that there are no repeating values in $\sin(2\pi \cdot 0.25/r), \sin(2\pi \cdot 1.25/r), \ldots \sin(2\pi(r-0.75)/r)$. suppose that for $0 \le i \ne j < r$, $\sin(2\pi \cdot (i+0.25)/r) = \sin(2\pi \cdot (j+0.25)/r)$. Analogously to the odd r case, this implies that $2\pi \cdot (i+0.25)/r + 2\pi(j+0.25)/r$ is either π or 3π. This simplifies to $2(i+j)+1 = r$ or $3r$. Since r is even, such integers i and j cannot exist, proving that the output intervals do not intersect.

For f_2, we define it as a composition of the r-th angle formula for cosine and inverse sine, scaled by $\pm\frac{1}{2\pi}$. Using the identity

$$f_1(x) = \sin\left(2\pi\frac{x+\frac{1}{4}}{r}\right) = \cos\left(2\pi\frac{x-\frac{1}{4}(r-1)}{r}\right),$$

applying the r-th angle formula for cosine to f_1 results in

$$\cos(2\pi x - \frac{\pi}{2}(r-1)) = (-1)^{\frac{r+1}{2}}\sin(2\pi x).$$

Applying the inverse sine then gives $\pm 2\pi \cdot t(x)$, and scaling by $\pm\frac{1}{2\pi}$ yields $t(x)$. Since the r-th angle formula is even, f_2 is an even function for even r.

To apply these functions in bootstrapping, we approximate f_1 and f_2 using the multi-interval Remez algorithm proposed by Lee et al. [20]. Note that any other approximation methods can be utilized. Specifically, we approximate f_1 over the multi-interval $\cup_{-K<i<K}(i-\epsilon, i+\epsilon)$, and f_2 over the range of f_1.

For odd r, the approximation polynomials of f_1 and f_2 derived from the multi-interval Remez algorithm are odd polynomials, due to the symmetry of the input intervals and the oddity of the functions. Conversely, for even r, the best approximation polynomial of f_1 may not be odd, as the input intervals are not necessarily symmetric about the origin. While using only odd polynomials as a basis to approximate f_1 can yield odd polynomials, it may increase the approximation error. This trade-off can be advantageous for certain parameters. On the other hand, the best approximation polynomial for f_2 is always an even polynomial.

3.2 Optimizations in the Bootstrapping

Now we discuss how our choice of f_1 and f_2 can be applied and optimized during the bootstrapping process.

3.2.1 Evaluation of f_2. We describe an efficient approach to evaluating f_2. f_2 can be decomposed similar to previous sine-based methods [15, 20], and moreover its approximation polynomial is sparse. Recall that f_2 is essentially a composition of the r-th angle formula and the inverse sine. If $r = 2^k \cdot r'$, where r' is odd, the approximation polynomial p_2 of f_2 can be expressed as:

$$p_2 = q \circ (-1)^{\frac{r'-1}{2}} T_{r'} \circ T_2^{\circ k},$$

where q is the approximation polynomial of the inverse sine. This decomposition allows p_2 to be computed by iteratively applying T_2 to the input k times, followed by evaluating $q \circ (-1)^{\frac{r'-1}{2}} T_{r'}$, without consuming any additional multiplicative levels.

Furthermore, we emphasize that $q \circ (-1)^{\frac{r'-1}{2}} T_{r'}$ is an odd sparse polynomial, which reduces the required ring arithmetic compared to arbitrary polynomials of the same degree. Specifically, let q, and odd polynomial of degree d, be expressed in the Chebyshev basis as $q = \sum_{i=1}^{(d+1)/2} q_i \cdot T_{2i-1}$. Then, $q \circ (-1)^{\frac{r'-1}{2}} T_{r'}$ can be written as

$$(-1)^{\frac{r'-1}{2}} \cdot \sum_{i=1}^{(d+1)/2} q_i \cdot T_{r'(2i-1)}$$

using the properties of Chebyshev polynomials. As a result, instead of generating all $d \cdot r'$ basis functions for computation, only d basis functions are needed,significantly reducing computational overhead.

3.2.2 Scaling. For high-degree polynomial evaluations in CKKS, input data must be confined within the range $[-1, 1]$ to avoid overflow. To achieve this, the input ciphertext of EvalMod is scaled by a constant during the Coeffs2Slots step, ensuring that the message values lie within $[-1, 1]$. In prior works, the input ciphertext was scaled by a factor of $1/K$, assuming the message values were confined within $[-K+1-\varepsilon, K-1+\varepsilon]$ with overwhelming probability. We propose a refinement by using the scailng factor $1/(K-0.5)$. First, observe that the input data is confined to the intervals $[-K+1-\varepsilon, K-1+\varepsilon]$ when r is odd, and $[-K+\frac{5}{4}-\varepsilon, K-\frac{3}{4}+\varepsilon]$ when r is even. Since $\varepsilon \ll \frac{1}{4}$ in typical scenarios, the input data for our algorithm is always within $[-K+0.5, K-0.5]$. Thus, scaling by $1/K-0.5$ guarantees that input data for EvalMod remains within the desired range $[-1, 1]$.

3.3 Measuring Goodness of New Method

In this section, we measure the effectiveness of our method numerically by estimating the lower bound of the approximation noise. Recall that f_1 is chosen as a function that maps $2K-1$ intervals into r intervals and is periodic when restricted to subsets of the input intervals. Specifically, for some disjoint subsets $I_1, \ldots, I_r$ of $\{-K+1, \ldots, K-1\}$ such that $\cup_{1 \le i \le r} I_i = \{-K+1, \ldots, K-1\}$, f_1 is periodic when restricted to each interval $\cup_{j \in I_i}(j-\varepsilon, j+\varepsilon)$. Now, suppose that the subsets $I_1, \ldots, I_r$ are fixed. Recall that the degree of the approximation polynomial for f_1 is typically higher than that of f_2. Thus, it is crucial to select f_1 such that it can be approximated with a polynomial with the smallest possible degree among all candidates for f_1. In the following, we prove that it is possible to

identify f_1 with the minimal approximation error in the L2 norm for a given polynomial degree. To achieve this, we first define the mean and error variance for the polynomial approximation of a periodic function over a given subset.

Definition 3.1. For $\varepsilon > 0$ and a continuous function $f \in \mathcal{F}(-\varepsilon, \varepsilon)$, a polynomial p with degree d and a subset $\mathcal{S}$ of $\mathbb{Z}$, we define

$$E(t, p; \mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} p(t - i)$$

$$\mathsf{Var}(f, p; \mathcal{S}) = \sum_{i \in \mathcal{S}} \int_{-\varepsilon}^{\varepsilon} (f(t) - p(t - i))^2 dt.$$

Given the subsets $I_1, \ldots, I_r$ and the polynomial degree d, our goal is to find functions g_i and a degree d polynomial p that minimize $\sum_{1 \le i \le r} \mathsf{Var}(g_i, p; I_i)$. Then, f_1 can be defined as a piecewise function $f(x) = g_i(x)$ for $x \in [j - \varepsilon, j + \varepsilon]$ with $j \in I_i$. However, directly minimizing $\sum_{1 \le i \le r} \mathsf{Var}(g_i, p; I_i)$ results in a trivial solution where $p = g_1 = \ldots g_r = b$ for some constant b, ensuring $\mathsf{Var}(g_i, p; I_i) = 0$ for all $1 \le i \le r$. To avoid this trivial solution, we fix two points of the polynomial p to constrain the minimization problem. Fixing any two points with distinct y-axis values is sufficient to guarantee a correct result, as the solution differs only by scale. Thus, we set $E(\pm\varepsilon, p; I) = \pm\varepsilon$ and scale the polynomial after approximation. This constraint eliminates two degrees of freedom in the polynomial approximation without fixing specific evaluation points, ensuring an effective approximation.

Under this constraint, there exist functions $g^*_{d,i}$ and a polynomial $p^*_{d,r}$ that minimize $\sum_{1 \le i \le r} \mathsf{Var}(g_i, p; I)$ when $g_i = g^*_{d,i}$ and $p = p^*_{d,r}$. Specifically,

$$\min_{\substack{E(\pm\varepsilon, p; I_1) = \pm\varepsilon \\ deg(p) = d}} \sum_{1 \le i \le r} \mathsf{Var}(g_i, p; I_i) = \sum_{1 \le i \le r} \mathsf{Var}(g^*_{d,i}, p^*_{d,r}; I_i). \quad (1)$$

Next, we show that such $g^*_{d,i}$'s are polynomials and they can be computed as a simple least-squares optimization problem. We begin by proving the following lemma, which establishes the relationship between $g^*_{d,i}$ and $p^*_{d,r}$.

Lemma 3.2. *For any positive integer $d > 0$ and $\varepsilon \in (0, 1/2)$, let $g^*_{d,i}$ $(1 \le i \le r)$ and $p^*_{d,r}$ be the solution of the minimization problem 1. Then, the following equation holds for any $1 \le i \le r$:*

$$g^*_{d,i} = E(x, p^*_{d,r}; I_i), \quad x \in [-\varepsilon, \varepsilon].$$

Proof. Suppose we are given the approximation polynomial $p^*_{d,r}$. Begin by considering the following optimization problem over a uniform partition $(x_1, \ldots, x_n)$ of $[-\varepsilon, \varepsilon]$ where the interval width is $\delta = x_2 - x_1 = \cdots = x_n - x_{n-1}$.

$$\min_{\substack{g_{1,1}, \ldots, g_{1,n}, \ldots, \\ g_{r,1}, \ldots, g_{r,n}}} \sum_{i=1}^{r} \sum_{j \in I_i} \sum_{k=1}^{n} (g_{i,k} - p^*_{d,r}(x_k - j))^2 \cdot \delta$$

Since this is a convex problem, the gradient of the target function at the solution is a zero vector. The partial derivative of the target function with respect to $g_{i,k}$ is

$$\delta \left(|I_i| g_{i,k} - \sum_{i \in I} p^*_{d,r}(x_k - j) \right).$$

Thus, the minimizer $(\tilde{g}_{i,k})_{i \in [r], k \in [n]}$ satisfies

$$\tilde{g}_{i,k} = \frac{1}{|I_i|} \sum_{j \in I} p^*_{d,r}(x_k - j).$$

This minimizer $\tilde{g}_{i,k}$ is essentially the mean of the polynomial $p^*_{d,r}$ over the set I_i. Next, consider the following optimization problem in the continuous setting:

$$\min_{g_1, \ldots, g_r} \sum_{i=1}^{r} \mathsf{Var}(g_i, p^*_{d,r}; I_i) = \sum_{i=1}^{r} \sum_{j \in I_i} \int_{-\varepsilon}^{\varepsilon} (g_i(t) - p^*_{d,r}(t - j))^2 dt.$$

Let $(\tilde{g}_{d,i})_{i \in [r]}$ represent the minimizer of this continuous fitting problem. By letting $n \to \infty$ in the discrete problem 3.3, $g_{d,i}(x)$ converges to the mean of the data, *i.e.*, $\tilde{g}_{d,i}(x) = E(x, p^*_{d,r}, I_i)$ for $1 \le i \le r$. Then, using the optimality of both solutions, we infer:

$$\sum_{i=1}^{r} \mathsf{Var}(\tilde{g}_{d,i}, p^*_{d,r}; I_i) \le \sum_{i=1}^{r} \mathsf{Var}(g^*_{d,i}, p^*_{d,r}; I_i) \le \sum_{i=1}^{r} \mathsf{Var}(\tilde{g}_{d,i}, p^*_{d,r}; I_i).$$

The left-hand inequality follows from the optimality of $(\tilde{g}_{d,i})_{i \in [r]}$, and the right-hand inequality follows from the optimality of $(g^*_{d,r})_{i \in [r]}$. Since the given problem is a convex problem, the universal optimizer is unique, and therefore $g^*_{d,i}(x) = \tilde{g}_{d,i}(x) = E(x, p^*_{d,r}; I_i)$ for $1 \le i \le r$. □

Using the result of this lemma, we show that the degree-d polynomial $p^*_{d,r}$ with the least approximation error can be computed efficiently. Let φ_i $(0 \le i \le d)$ denote the polynomial basis, and let $\overline{\varphi}^i_j$ $(0 \le j \le d)$ denote the average of φ_j over the interval I_i. *i.e.*, $\overline{\varphi}^i_j(x) = E(x, \varphi_j; I_i)$ $(0 \le j \le d)$. Now, if $p^*_{d,r} = \sum_{i=0}^{d} c_i \cdot \varphi_i$ for coefficients $c_i \in \mathbb{R}$, then

$$g^*_{d,i} = \frac{1}{|I_i|} \sum_{k \in I_i} \sum_{j=0}^{d} c_j \cdot \varphi_j = \sum_{j=0}^{d} c_j \cdot \overline{\varphi}^i_j$$

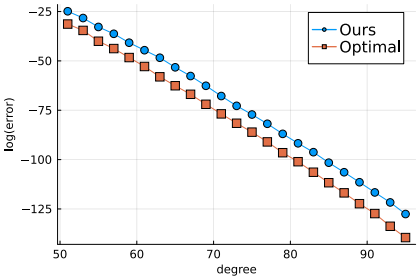
for all $1 \le i \le r$. Substituting this into the original minimization problem reduces into a problem of finding the coefficients $c_0, \ldots, c_d \in \mathbb{R}$ which minimize

$$\sum_{i=1}^{r} \mathsf{Var}\Big(\sum_{j=0}^{d} c_j \overline{\varphi}^i_j, \sum_{j=0}^{d} c_j \varphi_j; I_i \Big)$$

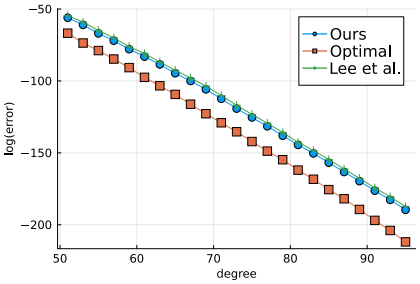
with a constraint $\sum_{i=0}^{d} c_i \overline{\varphi}^1_i(\pm\varepsilon) = \pm\varepsilon$. We remark that the solution to this problem can be obtained numerically by minimizing

$$g(\mathbf{c}) = \sum_{i=1}^{r} \mathsf{Var}\Big(\sum_{j=0}^{d} c_j \overline{\varphi}^i_j, \sum_{j=0}^{d} c_j \varphi_j; I_i \Big) + \lambda \Big[\Big(\sum_{i=0}^{d} c_i \overline{\varphi}^1_i(\varepsilon) - \varepsilon \Big)^2 + \Big(\sum_{i=0}^{d} c_i \overline{\varphi}^1_i(-\varepsilon) + \varepsilon \Big)^2 \Big]$$

for sufficiently big $\lambda > 0$. Since g is a convex function, the partial derivative $\frac{\partial}{\partial c_i} g(\mathbf{c}^*) = 0$ $(0 \le i \le d)$ for the minimizer $\mathbf{c}^*$. Using this fact, we can formulate a linear system of $(\sum_{i=1}^{r} \mathbf{T}_i + \lambda \mathbf{U}) \cdot \mathbf{c}^* = \lambda \mathbf{v}$ to



(a) Approximation error of our method, and the optimal error when $r = 5$



(b) Approximation error of Lee et al. [20], our method and the optimal error when $r = 8$

Figure 1: Comparison of the approximation error between our method, the best approximation method [20] and the optimal error.

solve the given minimization problem in polynomial time, where

$$\mathbf{T}_m = \left[\sum_{k \in I_m} \int_{-\varepsilon}^{\varepsilon} (\overline{\varphi}_i^m(t) - \varphi_i(t-k))(\overline{\varphi}_j^m(t) - \varphi_j(t-k))dt \right]_{i,j},$$

$$\mathbf{U} = \left[\overline{\varphi}_i^1(\varepsilon) \cdot \overline{\varphi}_j^1(\varepsilon) + \overline{\varphi}_i^1(-\varepsilon) \cdot \overline{\varphi}_j^1(-\varepsilon) \right]_{i,j} \quad \text{and}$$

$$\mathbf{v} = [\varepsilon \cdot (\overline{\varphi}_i^1(\varepsilon) - \overline{\varphi}_i^1(-\varepsilon))]_i.$$

We evaluate the effectiveness of our method by comparing it to the approximation lower bound obtained from the least-squares problem discussed earlier. For this experiment, we fixed $K = 26$ and $\epsilon = 2^{-8}$. Fig. 1a displays the L2-norm approximation error of our method alongside the lower bound of the error from Sec. 3.3, using the same sets $\{I_1, \ldots, I_r\}$ as in our method. As shown in the graph, our method achieves a similar approximation error while consuming only a few additional degrees, demonstrating near-optimality. Similarly, the L2-norm approximation errors for $r = 8$, comparing to our method, Lee et al.'s method [20], and the lower bound, are shown in Fig. 1b. Also for $r = 5$ case, our method achieves comparable approximation errors with only a few additional degrees, alongside the results of the prior method. Notably, our method displays a slightly smaller approximation error, by a few bits, compared to [20].

We also present possible parameter sets for our method and Lee et al.'s [20], achieving the same precision. To ensure a fair comparison, we optimized the polynomial degrees in both methods. Specifically, we first fixed the degree of the inverse sine approximation polynomial. Then, since the final precision cannot exceed the precision of the inverse sine approximation, we then determine the smallest degree approximation polynomials for f_1 and the scaled cosine (Lee et al.'s method). Using ℓ_2 (the number of levels consumed during f_2 evaluation in our method) and the number of levels consumed during the double angle formula and inverse sine evaluation in Lee et al.'s method, we selected the optimal r following the heuristic in Sec. 3.2.

Precision	ℓ_2	Ours		[20]	
		r	degree set	r	degree set
36 bits	2	1	(183, 3)	1	(183, 3)
	3	2	(107, 2, 3)	2	(107, 2, 3)
	4	5	(55, 15)	4	(65, 2, 2, 3)
	5	10	(37, 2, 15)	8	(43, 2, 2, 2, 3)
50 bits	3	1	(195, 5)	1	(195, 5)
	4	3	(87, 15)	2	(117, 2, 5)
	5	6	(57, 2, 15)	4	(73, 2, 2, 5)
	6	12	(39, 2, 2, 15)	8	(49, 2, 2, 2, 5)

Table 1: Approximation polynomial degrees for our method and the previous method [20].

Table 1 presents the parameters for both 36-bit and 50-bit precision. As expected, the degree of the approximation polynomials is reduced both cases. Furthermore, our method benefits from the efficient evaluation of odd/even polynomials, unlike Lee et al. [20]'s work, making polynomial evaluation even more efficient. Additionally, when $r = 2$, our method saves one level during polynomial evaluation compared to the prior method.

4 Improved Linear Transformation

In this section, we improve the time complexity of the matrix evaluation algorithm. We observe that the rescaling can be fused into the outer loop of the BSGS algorithm. Subsequently, the number of (i)NTTs during the linear transformation is reduced. In the following subsections, the number of the slots and the ring dimension are denoted by n and N, respectively.

4.1 Previous Methods

We commence by briefly explaining the underlying idea of the previous matrix-vector multiplication algorithms. In the context of SIMD arithmetic, the matrix multiplication can be realized with a 'diagonal packing' of the matrix, and rotations of the ciphertext. Suppose that we multiply an $n \times n$ matrix $\mathbf{M} = (m_{i,j})_{i,j \in [n]}$ to an encryption of vector $\mathbf{x} = (x)_{i \in [n]}$. Then, we can write the matrix multiplication as the sum of the point-wise multiplications between diagonal vectors $\mathbf{M}_j = (m_{j,[n-i+j]_n})_{j \in [n]}$ of $\mathbf{M}$ and the rotations $\mathbf{x}^{(i)} = (x_{[n-i+j]_n})_{j \in [n]}$ of $\mathbf{x}$.

$$\begin{bmatrix} m_{0,0} & m_{0,1} & \cdots & m_{0,n-1} \\ m_{1,0} & m_{1,1} & \cdots & m_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n-1,0} & m_{n-1,1} & \cdots & m_{n-1,n-1} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}$$
$$= \sum_{i=1}^{n} \begin{bmatrix} m_{0,n-i} \\ m_{1,n-i+1} \\ \vdots \\ m_{n-1,n-1-i} \end{bmatrix} \odot \begin{bmatrix} x_{n-i} \\ x_{n-i+1} \\ \vdots \\ x_{n-1-i} \end{bmatrix}$$
$$= \sum_{i=1}^{n} \mathbf{M}_i \odot \mathbf{x}^{(i)}.$$

Based on this observation, matrix multiplication can be realized with n rotations over the input ciphertext. A naïve approach would involve computation of n rotations of the encryptions of $\mathbf{x}$. Considering that the gadget decomposition operates on the coefficients of the polynomial, each key-switching would require $\beta + 3$ (i)NTTs. This is because we need to perform 1 iNTT before the gadget decomposition in order to make the ring element into coefficient form, and β NTTs after the gadget decomposition for the following polynomial multiplications. Finally, to divide the resulting ciphertext with the special modulus P, it is necessary to transform the ciphertext to the coefficient form. As a result, it requires two (i)NTTs before and after the division by special modulus. Consequently, this approach demands $(n-1)(\beta+3)$ (i)NTTs total.

This naïve method was improved in [14] using two methods, *hoisting* and the *Baby-step Giant-step* (BSGS) algorithm. Observe that the automorphism essentially rearranges the coefficients, and therefore it can be performed over the gadget decomposition of a ring element, thereby yielding a gadget decomposition of a rotated ring element. Additionally, as automorphisms can also be applied to the NTT form as well, only one gadget decomposition is needed to perform all the key-switchings. However, division by the special modulus is still required for each rotation and $2(n-1)$ more (i)NTTs are needed. This technique effectively reduces the number of the (i)NTTs to $\beta + 2n - 1$, however, this method necessitates to store all n key-switching keys.

The BSGS matrix multiplication algorithm reduces the space overhead from n to $O(\sqrt{n})$, sacrificing the time complexity. For $n_1, n_2 = O(\sqrt{n})$ such that $n = n_1 \cdot n_2$, the BSGS matrix-vector multiplication algorithm is as follows.

(1) Compute $\mathsf{ct}^{(0)} = \mathsf{ct}, \mathsf{ct}^{(1)}, \ldots, \mathsf{ct}^{(n_1-1)}$, the $0, \ldots, n_1 - 1$-th rotations of ct, the encryption of $\mathbf{x}$.
(2) Generate n_2 'rotated' partial sums $\mathsf{psum}_j := \sum_{i=0}^{n_1-1} \mathbf{M}_{n_1 j+i}^{(-n_1 j)} \cdot \mathsf{ct}^{(i)}$ $(0 \le j < n_2)$ using the pre-rotated encodings $\mathbf{M}_{n_1 j+i}^{(-n_1 j)}$ $(1 \le i \le n_1, 0 \le j < n_2)$.
(3) Return $\sum_{j=0}^{n_2-1} \mathsf{psum}_j^{(n_1 j)}$.

In the first step of this algorithm, only $\beta + 2n_1 - 1$ (i)NTTs are required to compute all ciphertext rotations using the hoisting technique. Subsequently, the third step requires $(n_2 - 1)(\beta + 3)$ (i)NTTs. In total, $n_2(\beta + 3) + 2n_1 - 4$ (i)NTTs are needed for the BSGS algorithm.

This algorithm was further refined in a subsequent study [2], driven by two observations: 1. Rotations over ring elements can be reduced using pre-rotated rotation keys, 2. Plaintext multiplications and additions in the second step can be performed without requiring division by the special modulus.

The first improvement leverages the following equalities:

$$\begin{aligned} \langle h(\psi_i(a)), \mathsf{rtk}_i \rangle &= \langle \psi_i(h(a)), \mathsf{rtk}_i \rangle \\ &= \psi_i(\langle h(a), \psi_{-i}(\mathsf{rtk}_i) \rangle) \end{aligned}$$

where h is the gadget decomposition, $a \in R_Q$ is a ring element, and rtk_i is the i-th rotation key. By introducing pre-rotated rotation keys $\widetilde{\mathsf{rtk}}_i := \psi_{-i}(\mathsf{rtk}_i)$, a single rotation can be instantiated with only one automorphism instead of β automorphisms, where β is the length of the gadget decomposition. This technique, termed the *double-hoisting*, builds on the hoisting technique introduced in [14]. It reduces the number of automorphisms by a factor of β.

On the other hand, for an encryption $\mathsf{ct} = (b, a) \in R_Q$ of plaintext m under secret t, and a key switching key $\mathsf{ksk}_{t \to s}$, a *lazy key-switching* computes $(P \cdot b, 0) + \langle h(a), \mathsf{ksk}_{t \to s} \rangle \in R_{PQ}^2$, resulting in an erroneous encryption of $P \cdot m$ under secret s. Here, the plaintext is only multiplied by the special modulus P, allowing ciphertext addition and plaintext multiplication to proceed naturally.

Building on these idea, the computation of $\mathsf{ct}^{(i)}$ and the rotation of partial sums can be performed with lazy key-switching, removing the need for division by the special modulus. Subsequently, the generation and the summation of the partial sums are instantiated over modulus PQ. Moreover, since the key-switching operation is only needed to be applied to the second index a for an input ciphertext (b, a), we can keep the first index for all the partial sums and then the division by the special modulus P can be performed at the end of the algorithm. Therefore, only $\beta + 1$ (i)NTTs are needed before and after the gadget decomposition of input ciphertext ct. Then, after the computation of the partial sum psum_i, it requires $\beta + 1$ (i)NTTs for every iteration for the division by special modulus P and the gadget decomposition. Finally, four (i)NTTs are needed for the division by special modulus of the final ciphertext. To sum up it all, it requires $n_2(\beta+1)+4$ (i)NTTs total. The precise algorithm is described in Alg. 1.

4.2 Improved matrix-vector multiplication algorithm

As discussed in the previous section, the time complexity of matrix-vector multiplication algorithms is primarily determined by the number of iNTT and NTT operations. Consequently, prior works have focused on minimizing the number of the (i)NTTs. However, we emphasize that the algorithm can be accelerated by reducing the time complexity of each unit (i)NTT operations. Our improvement is motivated from two key observations: 1. (i)NTT operations are more efficient at lower levels. 2. Rescaling and division by P can be performed at once with a cost of one.

To explain the rationale behind our approach, we first briefly outline the software implementation of CKKS scheme. A common and efficient method for implementing CKKS is to use the *Residue Number System* (RNS) [7]. In the RNS-CKKS variant, each ring polynomial is represented as a tuple of polynomials with word-sized prime moduli, where the product of these moduli forms the

Algorithm 1 Double-hoisting BSGS matrix-vector multiplication [2]

Input: A CKKS ciphertext $\mathsf{ct} \in R^2_{Q_\ell}$, $\mathbf{M} \in R_{PQ_\ell}$, the pre-rotated encoding of matrix, $n_1 n_2 = n$, $\widetilde{\mathsf{rtk}}_i = (\widetilde{\mathsf{rtk}}_i^0, \widetilde{\mathsf{rtk}}_i^1) \in R^2_{PQ_\ell}$, the set of pre-rotated rotation keys.
Output: CKKS ciphertext $\mathsf{ct}' \in R^2_{Q_{\ell-1}}$

$\mathbf{d} \leftarrow \left[[c_1]_{q_{\alpha_0 \le i < \beta}}\right]_{PQ_\ell}$
$(a_0, b_0) \leftarrow (P \cdot c_0, P \cdot c_1)$
for $i = 1; i < n_1; i = i + 1$ **do**
  $a_i \leftarrow \psi_i\left(a_0 + \langle \mathbf{d}, \widetilde{\mathsf{rtk}}_i^0 \rangle\right)$
  $b_i \leftarrow \psi_i\left(\langle \mathbf{d}, \widetilde{\mathsf{rtk}}_i^1 \rangle\right)$
end for
$(c'_0, c'_1) \leftarrow (0, 0)$
for $j = 0; j < n_2; j = j + 1$ **do**
  $(u_0, u_1) \leftarrow (0, 0)$
  for $i = 0; i < n_1; i = i + 1$ **do**
    $(u_0, u_1) \leftarrow (u_0, u_1) + (a_i, b_i) \cdot \mathbf{M}^{(n_1 \cdot j + i)}$
  end for
  $u_1 \leftarrow \lfloor P^{-1} \cdot u_1 \rceil$
  $\mathbf{d} \leftarrow \left[[u_1]_{q_{\alpha_0 \le i < \beta}}\right]_{PQ_\ell}$
  $c'_0 \leftarrow c'_0 + \psi_{n_1 \cdot j}(u_0 + \langle \mathbf{d}, \widetilde{\mathsf{rtk}}^0_{n_1 \cdot j} \rangle)$
  $c'_1 \leftarrow c'_1 + \psi_{n_1 \cdot j}(\langle \mathbf{d}, \widetilde{\mathsf{rtk}}^1_{n_1 \cdot j} \rangle)$
end for
$(c'_0, c'_1) \leftarrow \left(\lfloor P^{-1} \cdot c'_0 \rceil, \lfloor P^{-1} \cdot c'_1 \rceil\right)$
$\mathsf{ct}' \leftarrow \left(\lfloor Q_{\ell-1}/Q_\ell \cdot c'_0 \rceil, \lfloor Q_{\ell-1}/Q_\ell \cdot c'_1 \rceil\right)$
return ct'

Algorithm 2 Improved double-hoisting BSGS matrix-vector multiplication

Input: A CKKS ciphertext $\mathsf{ct} \in R^2_{Q_\ell}$, $\mathbf{M} \in R_{PQ_\ell}$, the pre-rotated encoding of matrix, $n_1 n_2 = n$, $\widetilde{\mathsf{rtk}}_i = (\widetilde{\mathsf{rtk}}_i^0, \widetilde{\mathsf{rtk}}_i^1) \in R^2_{PQ_\ell}$, the set of pre-rotated rotation keys.
Output: CKKS ciphertext $\mathsf{ct}' \in R^2_{Q_{\ell-1}}$.

$\mathbf{d} \leftarrow \left[[c_1]_{q_{\alpha_0 \le i < \beta}}\right]_{PQ_\ell}$
$(a_0, b_0) \leftarrow (P \cdot c_0, P \cdot c_1)$
for $i = 1; i < n_1; i = i + 1$ **do**
  $a_i \leftarrow \psi_i\left(a_0 + \langle \mathbf{d}, \widetilde{\mathsf{rtk}}_i^0 \rangle\right)$
  $b_i \leftarrow \psi_i\left(\langle \mathbf{d}, \widetilde{\mathsf{rtk}}_i^1 \rangle\right)$
end for
$(c'_0, c'_1) \leftarrow (0, 0)$
$u \leftarrow 0$
for $j = 0; j < n_2; j = j + 1$ **do**
  $(u_0, u_1) \leftarrow (0, 0)$
  for $i = 0; i < n_1; i = i + 1$ **do**
    $(u_0, u_1) \leftarrow (u_0, u_1) + (a_i, b_i) \cdot \mathbf{M}^{(n_1 \cdot j + i)}$
  end for
  $u \leftarrow u + \psi_{n_1 \cdot j}(u_0)$
  $u_1 \leftarrow \lfloor (PQ_\ell/Q_{\ell-1})^{-1} \cdot u_1 \rceil$
  $\mathbf{d} \leftarrow \left[[u_1]_{q_{\alpha_0 \le i < \beta}}\right]_{PQ_{\ell-1}}$
  $(c'_0, c'_1) \leftarrow (c'_0, c'_1) + \psi_{n_1 \cdot j}\left(\left(\langle \mathbf{d}, \widetilde{\mathsf{rtk}}^0_{n_1 \cdot j} \rangle, \langle \mathbf{d}, \widetilde{\mathsf{rtk}}^1_{n_1 \cdot j} \rangle\right)\right)$
end for
$(c'_0, c'_1) \leftarrow (u + Q_\ell/Q_{\ell-1} \cdot c'_0, Q_\ell/Q_{\ell-1} \cdot c'_1)$
$\mathsf{ct}' \leftarrow \left(\lfloor (PQ_\ell/Q_{\ell-1})^{-1} \cdot c'_0 \rceil, \lfloor (PQ_\ell/Q_{\ell-1})^{-1} \cdot c'_1 \rceil\right)$
return ct'

ciphertext, employing the *Chinese Remainder Theorem* (CRT). Consequently, the (i)NTT for a ring polynomial is performed as (i)NTT over each prime modulus. Therefore, the time complexity of (i)NTT operations is directly proportional to the number of prime factors. As a result, (i)NTTs at lower levels (with fewer prime factors) are inherently faster than those at higher levels. This motivates our approach: if the (i)NTTs during matrix-vector multiplication can be performed at lower levels, the overall time complexity can be reduced.

We propose integrating the rescaling operation into the matrix-vector multiplication algorithm to reduce the level during computation, thereby achieving lower time complexity. Rescaling can be applied to the partial sum concurrently with the division by the special modulus without introducing any additional unit (i)NTT operations. Our proposed algorithm is described in detail in Alg. 2.

Let us elaborate our algorithm in detail. The generation of pre-rotated ciphertext and the partial sums is executed over the ring R_{PQ_ℓ}. Note that only the second element of the partial sum is gadget decomposed, we lazily divide the first element by the special modulus and then rescale. Consequently, we only divide the second index by $PQ_\ell/Q_{\ell-1}$. Then, the base ring is converted to $R_{Q_{\ell-1}}$ and NTT operation on the gadget decomposition is executed over the ring $R_{PQ_{\ell-1}}$ with a reduced time complexity. Then after the outer loop, we divide the accumulator with the special modulus P and the accumulator u for the first index with $PQ_\ell/Q_{\ell-1}$.

4.2.1 Comparison. We compare the matrix-vector multiplication algorithm from prior work [2] with our refined approach by evaluating the number of the word-sized (i)NTTs. Let $\mathcal{L}, \mathcal{L}', \mathcal{L}_P$ and β denote the number of the prime factors for the moduli of the input and output ciphertexts, and the length of the gadget decomposition, respectively.

In Alg. 1, the pre-computations of rotations for the input ciphertext ct requires $(\beta+1)\mathcal{L} + \beta\mathcal{L}_P$ (i)NTTs. Subsequently, during each iteration of the outer loop, $\mathcal{L} + \mathcal{L}_P$ iNTTs are performed for the division by the special modulus for the accumulator u_1, and $\beta(\mathcal{L}+\mathcal{L}_P)$ NTTs are executed after the decomposition of u_1. At the end of the algorithm, $2(\mathcal{L} - \mathcal{L}' + \mathcal{L}_P)$ iNTTs are performed for the division by P and rescaling, while $2\mathcal{L}'$ NTTs are performed for the transformation into the evalution form. Thus, the total number of (i)NTTs for the double-hoisting BSGS matrix-vector multiplication is given by

$$(\beta+1)\mathcal{L} + \beta\mathcal{L}_P + n_2(\beta+1)(\mathcal{L}+\mathcal{L}_P) + 2(\mathcal{L}+\mathcal{L}_P).$$

In Alg. 2, our refined approach, the pre-computation of rotations for the input ciphertext ct similarly requires $(\beta+1)\mathcal{L}+\beta\mathcal{L}_P$ (i)NTTs. For each iteration of the outer loop, $\mathcal{L}+\mathcal{L}_P$ iNTTs are performed for

division by $PQ_\ell/Q_{\ell-1}$ for the accumulator u_1, and only $\beta(\mathcal{L}'+\mathcal{L}_P)$ NTTs are needed for the transformation of u_1's decomposition due to its reduced level. Following the outer loop, dividing and rounding c_0' and c_1' require $2(\mathcal{L}-\mathcal{L}'+\mathcal{L}_P)$ iNTTs and $2\mathcal{L}'$ NTTs. Therefore, the number of the (i)NTTs required for our algorithm is

$$(\beta+1)\mathcal{L}+\beta\mathcal{L}_P+n_2\left((\beta+1)(\mathcal{L}+\mathcal{L}_P')-\beta(\mathcal{L}-\mathcal{L}')\right)+2(\mathcal{L}+\mathcal{L}_P).$$

Consequently, our proposed algorithm reduces the number of (i)NTTs by $\beta n_2(\mathcal{L}-\mathcal{L}')$.

4.2.2 Discussion. Below, we present the experimental result for our improved linear transformation method. In the table, radix refers to the number of sparse matrices into which the encoding/decoding matrix is decomposed. For the radix 1 case, we used $N = 2^{15}$ instead of $N = 2^{16}$, due to the overwhelming size of the rotation keys.

radix	Method	[2]	Ours
1	CoeffToSlot	20.33s*	19.71s*
	SlotToCoeff	7.07s*	6.88s*
2	CoeffToSlot	14.39s	13.50s
	SlotToCoeff	3.47s	3.34s
3	CoeffToSlot	10.76s	10.57s
	SlotToCoeff	2.08s	2.03s

Table 2: Performance comparison with prior work [2], with parameter set II [2]. We used $N = 2^{15}$ for experimental results with asterisk.

Our method demonstrates better efficiency when the radix is smaller, *i.e.,* when the matrix is denser, particularly in the higher level (e.g., during the CoeffToSlot step). This efficiency arises because the number of the (i)NTTs reduced in our algorithm is $O(\beta\sqrt{n})$. It is worth noting that bootstrapping is typically performed at higher levels, so the performance improvements are more pronounced when matrix multiplication is executed at lower levels. This algorithmic improvement is not limited to the CKKS scheme but is also applicable to other leveled SIMD HE schemes where the ciphertext modulus is reduced during computation, such as BGV [4] or leveled BFV [11].

5 Experimental Results and Parameters

In this section, we provide an heuristic to find the best parameters, and the experimental results. We implement our method using the Lattigo library [23], and also provide a recommended parameter set which achieves the same precision and security to the previous methods [2, 21]. All experiments were conducted on a machine equipped with an Intel(R) Xeon(R) Platinum 8268 CPU @ 2.90GHz and 192GB RAM, running Ubuntu 20.04.2 LTS.

5.1 Parameter Selection

We propose a heuristic to choose the optimal r and the degrees of the approximation polynomials for f_1 and f_2. Generally, the approximation error decreases as the degree of the (best) approximation polynomial increases. However, higher-degree polynomials consume more multiplicative levels in CKKS. Thus, it is crucial to set adequate approximation polynomial degrees for f_1 and f_2 to optimize the EvalMod step.

Suppose we aim to evaluate $t(x)$ with ε-bits of precision during EvalMod. In this case, the approximation polynomials for f_1 and f_2 must achieve approximation errors no greater than ε-bits. Using this requirement, we can devise a heuristic to select r, f_1 and f_2. Since the degree of f_1 is generally larger than f_2 for practical parameters, we focus on optimizing f_2 first. Recall that f_2 is essentially a composition of the r-th angle formula, which is polynomial, and the inverse sine. Thus, its approximation error is identical to the error of inverse sine. For example, Fig. 2 shows the error of the best approximation polynomials of the inverse sine and f_2 when $r = 3$, computed using the multi-interval Remez algorithm [20]. The error of a degree-d approximation polynomial of f_2 matches the error of a degree $\lfloor d/r \rfloor$ best approximation polynomial of the inverse sine.

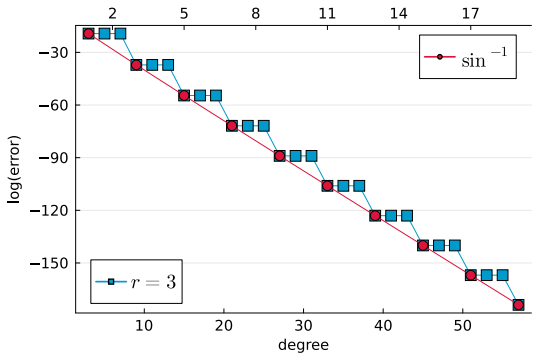


Figure 2: Approximation error of inverse sine and f_2, when $r = 3$.

Based on this observation, we can first determine d regardless of r, selecting d as the smallest odd integer such that a degree-d (best) approximation polynomial of the inverse sine achieves ε-bits of approximation error. Let ℓ_1 and ℓ_2 denote the number of levels consumed during the evaluation of f_1 and f_2, respectively. When ℓ_2 is fixed, we choose the largest r possible such that $rd < 2^{\ell_2}$, *i.e.,* $r = \lfloor \frac{\log \ell_2}{d} \rfloor$. A larger r allows the use of a lower-degree approximation polynomial for f_1. Finally, with r settled, we compute the approximation polynomial for f_1, ensuring its approximation error does not exceed ε-bits.

5.2 Experimental Results

We compare our proposed method with state-of-the-art bootstrapping methods [20, 21]. In prior works [2, 20], the parameters ϵ and K were deliberately chosen to minimize level consumption and maximize bootstrapping precision, leaving little room for further optimization. Therefore, we provide a wider range of parameters to demonstrate the flexibility and scalability of our method. The parameters for ϵ, K, and the Coeffs2Slots and Slots2Coeffs steps are summarized in Table 3. We note that the parameters for K are specified for a secret key with Hamming weight $h = 192$. Specifically, $K = 16$ when the key-encapsulation technique [3] is applied, and $K = 23$ otherwise.

Set	ε	K	Coeffs2Slots	Slots2Coeffs	$\log P$
I	2^{-6}	23	58×4	42×3	61×4
II		16			

Table 3: Bootstrapping parameters used in the experiments.

The recommended parameter set and experimental results are presented in Table. 4. We select $r > 2$ since most practical parameter sets use $r > 2$ for performance reasons. During the evaluation, the lazy-BSGS polynomial evaluation algorithm [21] was applied to all bootstrapping methods. The remaining levels after bootstrapping are fixed for our proposed method and the prior works for consistency. For the experiments, the messages in the slots were sampled from a distribution where the real and imaginary parts were drawn uniformly from $[-1, 1]$, consistent with the assumptions in prior works [2, 21].

Set	Method	r	Degree Set	$\log PQ$	Level	avg prec.	# relin	BTS
I	Ours	5	(53, 25)	1547	11	27.12	20	24.9s
		6	(47, 2, 15)	1547	11	27.40	19	25.1s
		12	(31, 2, 2, 15)	1547	11	26.42	17	24.6s
	[2, 20]	4	(61, 2, 2, 5)	1547	11	27.43	20	25.5s
		8	(39, 2, 2, 2, 5)	1562	12	26.54	19	25.0s
	[21]	-	(511)	1517	9	29.38	31	40.0s
II	Ours	3	(57, 15)	1532	10	28.70	20	24.8s
		6	(39, 2, 15)	1547	11	28.76	18	24.4s
		11	(29, 55)	1547	11	28.75	18	24.9s
		12	(27, 2, 2, 15)	1547	11	28.76	17	23.8s
	[2, 20]	4	(49, 2, 2, 5)	1547	11	28.77	19	24.6s
		8	(33, 2, 2, 2, 5)	1562	12	28.74	18	24.0s
	[21]	-	(301)	1517	9	29.74	25	35.0s

Table 4: Comparison of the degree set, ciphertext modulus, level consumption, average bootstrapping precision, the number of relinearizations, and the bootstrapping timing for our approximation methods and the previous ones [2, 20, 21].

In the table, we compare our proposed method, which approximates the (generalized) cosine function and its inverse function, to state-of-the-art methods. These include previous works that either use only power-of-two r [2, 20], or directly approximate the target function $t(x)$ [21]. As anticipated, our optimized composition method consumes fewer multiplicative levels than the original sine method. For instance, in parameter set I, our optimized composition method consumes only 11 levels across all recommended parameters, whereas the sine method [2, 20] has parameters consuming 12 levels. Similarly, in parameter set II, our method consumes 10 to 11 levels, while the prior works [2, 20] consume 11 to 12 levels. Furthermore, in both parameter sets I and II, our method offers parameters with the least number of the relinearizations. Specifically, our parameter set requires only 17 key-switching operations for some parameters, whereas the prior works [2, 20] require at least 18 relinearizations. Additionally, the elapsed time for bootstrapping with our method is faster than both the sine method [2, 20] and the direct approximation method [21]. This improvement is primarily due to the reduced key-switching operations and the enhanced linear transformation algorithm. Consequently, our optimized composite function method offers a more advantageous parameters in both the level consumption and time complexity, making it a more efficient choice for practical applications.

Acknowledgments

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2023-00211649) and the Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. 2022-0-01047, Development of statistical analysis algorithm and module using homomorphic encryption based on real number operation & No. RS-2024-00398360, Development of a user-friendly and efficiency-optimized real-time homomorphic statistical analysis processing platform).

References

[1] Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. 2022. META-BTS: Bootstrapping Precision Beyond the Limit. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 223–234. doi:10.1145/3548606.3560696

[2] Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2021. Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys. In *Advances in Cryptology – EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I* (Zagreb, Croatia). Springer-Verlag, Berlin, Heidelberg, 587–617. doi:10.1007/978-3-030-77870-5_21

[3] Jean-Philippe Bossuat, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2022. Bootstrapping for Approximate Homomorphic Encryption with Negligible Failure-Probability by Using Sparse-Secret Encapsulation. In *Applied Cryptography and Network Security: 20th International Conference, ACNS 2022, Rome, Italy, June 20–23, 2022, Proceedings* (Rome, Italy). Springer-Verlag, Berlin, Heidelberg, 521–541. doi:10.1007/978-3-031-09234-3_26

[4] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference* (Cambridge, Massachusetts) *(ITCS '12)*. Association for Computing Machinery, New York, NY, USA, 309–325. doi:10.1145/2090236.2090262

[5] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2019. Improved Bootstrapping for Approximate Homomorphic Encryption. In *Advances in Cryptology – EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part II* (Darmstadt, Germany). Springer-Verlag, Berlin, Heidelberg, 34–54. doi:10.1007/978-3-030-17656-3_2

[6] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for approximate homomorphic encryption. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part I 37*. Springer, Springer Cham, Berlin, Heidelberg, 360–384. doi:10.1007/978-3-319-78381-9_14

[7] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2019. A Full RNS Variant of Approximate Homomorphic Encryption. In *Selected Areas in Cryptography – SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers* (Calgary, AB, Canada). Springer-Verlag, Berlin, Heidelberg, 347–368. doi:10.1007/978-3-030-10970-7_16

[8] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, Springer Cham, Berlin, Heidelberg, 409–437. doi:10.1007/978-3-319-70694-8_15

[9] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *J. Cryptol.* 33, 1 (Jan. 2020), 34–91. doi:10.1007/s00145-019-09319-x

[10] Léo Ducas and Daniele Micciancio. 2015. FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, Springer Cham, Berlin, Heidelberg, 617–640. doi:10.1007/978-3-662-46800-5_24

[11] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2012/144. https://eprint.iacr.org/2012/144

[12] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing* (Bethesda, MD, USA) *(STOC '09)*. Association for Computing Machinery, New York, NY, USA, 169–178. doi:10.1145/1536414.1536440

[13] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*. Springer, Springer Cham, Berlin, Heidelberg, 75–92. doi:10.1007/978-3-642-40041-4_5

[14] Shai Halevi and Victor Shoup. 2018. Faster Homomorphic Linear Transformations in HElib. In *Advances in Cryptology – CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I* (Santa Barbara, CA, USA). Springer-Verlag, Berlin, Heidelberg, 93–120. doi:10.1007/978-3-319-96884-1_4

[15] Kyoohyung Han and Dohyeong Ki. 2020. Better Bootstrapping for Approximate Homomorphic Encryption. In *Topics in Cryptology – CT-RSA 2020: The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24–28, 2020, Proceedings* (San Francisco, CA, USA). Springer-Verlag, Berlin, Heidelberg, 364–390. doi:10.1007/978-3-030-40186-3_16

[16] Intak Hwang, Jinyeong Seo, and Yongsoo Song. 2023. Optimizing HE operations via Level-aware Key-switching Framework. In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography* (Copenhagen, Denmark) *(WAHC '23)*. Association for Computing Machinery, New York, NY, USA, 59–67. doi:10.1145/3605759.3625263

[17] Charanjit S Jutla and Nathan Manohar. 2022. Sine series approximation of the mod function for bootstrapping of approximate HE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Springer Cham, Berlin, Heidelberg, 491–520. doi:10.1007/978-3-031-06944-4_17

[18] Miran Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. 2023. Accelerating HE Operations from Key Decomposition Technique. In *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part IV* (Santa Barbara, CA, USA). Springer-Verlag, Berlin, Heidelberg, 70–92. doi:10.1007/978-3-031-38551-3_3

[19] Seonghak Kim, Minji Park, Jaehyung Kim, Taekyung Kim, and Chohong Min. 2023. EvalRound Algorithm in CKKS Bootstrapping. In *Advances in Cryptology – ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part II* (Taipei, Taiwan). Springer-Verlag, Berlin, Heidelberg, 161–187. doi:10.1007/978-3-031-22966-4_6

[20] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. 2021. High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Function. In *Advances in Cryptology – EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I* (Zagreb, Croatia). Springer-Verlag, Berlin, Heidelberg, 618–647. doi:10.1007/978-3-030-77870-5_22

[21] Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and HyungChul Kang. 2022. High-Precision Bootstrapping for Approximate Homomorphic Encryption by Error Variance Minimization. In *Advances in Cryptology – EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 – June 3, 2022, Proceedings, Part I* (Trondheim, Norway). Springer-Verlag, Berlin, Heidelberg, 551–580. doi:10.1007/978-3-031-06944-4_19

[22] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2013. On Ideal Lattices and Learning with Errors over Rings. *J. ACM* 60, 6, Article 43 (Nov. 2013), 35 pages. doi:10.1145/2535925

[23] Christian Vincent Mouchet, Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2020. Lattigo: A multiparty homomorphic encryption library in go. In *Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography*. Association for Computing Machinery, New York, NY, USA, 64–70.

[24] Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM* 56, 6, Article 34 (Sept. 2009), 40 pages. doi:10.1145/1568318.1568324