Low-Latency Bootstrapping for CKKS using Roots of Unity

Jean-Sébastien Coron and Robin Köstler

University of Luxembourg

Abstract. We introduce a new bootstrapping equation for the CKKS homomorphic encryption scheme of approximate numbers. The original bootstrapping approach for CKKS consists in homomorphically evaluating a polynomial that approximates the modular reduction modulo q. In contrast, our new bootstrapping equation directly embeds the additive group modulo q into the complex roots of unity, which can be evaluated natively in the CKKS scheme. Due to its reduced multiplicative depth, our new bootstrapping equation achieves a 7x latency improvement for a single slot compared to the original CKKS bootstrapping, though it scales less efficiently when applied to a larger number of slots.

1 Introduction

Fully Homomorphic Encryption. Homomorphic encryption (HE) enables to perform operations on encrypted data without knowing the decryption key. *Fully homomorphic encryption* (FHE) extends this capability, enabling the evaluation of arbitrary circuits on encrypted data. FHE has a wide range of applications, including secure cloud computing, multi-party computation, and secure machine learning.

Since Gentry's invention of FHE in 2009 [Gen09], numerous homomorphic encryption schemes have been developed, leading to significant improvements in FHE performance. Gentry's key innovation was the introduction of *bootstrapping*, a ciphertext refreshing procedure based on homomorphically evaluating the decryption circuit. Building on this breakthrough, schemes such as BGV [BGV11] and BFV [Bra12,FV12] further enhanced FHE by basing their security on the hardness of the ring learning with errors (RLWE) problem [SSTX09,LPR10]. These schemes can achieve high computational throughput by utilizing Single Instruction, Multiple Data (SIMD) operations [GHS12a,SV14].

The FHEW scheme, introduced in [DM15], reduced the time for bootstrapping a single-bit encryption to under one second, building on the approach from [AP14] and the matrix-based scheme of [GSW13]. This was further improved in [CGGI16] with the torus-FHE (TFHE) scheme, achieving a bootstrapping time of less than 0.1 second.

The CKKS scheme. In [CKKS17], the authors presented a construction for homomorphic encryption supporting approximate arithmetic, based on the RLWE problem. The core idea involves adding noise to the plaintext to ensure security, with this noise being treated as part of the error inherent in approximate computations. Given a ciphertext ct, for a secret key sk, the decryption equation $\langle \mathsf{ct}, \mathsf{sk} \rangle \pmod q$ outputs an approximation $m + e$ of the original message m, for a small error e. The authors introduced a rescaling procedure to control the magnitude of plaintexts, enabling the construction of a leveled homomorphic encryption scheme, where the ciphertext modulus grows linearly with the depth of the evaluated circuit. Through rescaling, the scheme can emulate fixed-point addition and multiplication on encrypted messages. Additionally, they described a packing method that allows the encryption of up to $N/2$ complex numbers in a single ciphertext to perform SIMD operations.

CKKS bootstrapping. In [CHK+18a], the authors introduced a novel ciphertext refreshing procedure for CKKS, extending the leveled homomorphic encryption scheme to fully homomorphic encryption based on Gentry's bootstrapping technique. Specifically, the initial leveled scheme from [CKKS17] can only evaluate circuits of a fixed depth, as each homomorphic multiplication reduces the ciphertext modulus until it becomes too small to support further computation.

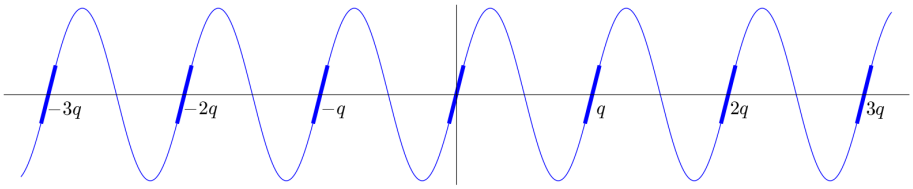


Fig. 1. Approximating modular reduction via a scaled sine function.

As illustrated in Figure 1, CKKS bootstrapping relies on approximating the modular reduction modulo q using a scaled sine function. Since the sine function is 2π-periodic, the decryption function can be expressed as:

$$[\langle \mathsf{ct}, \mathsf{sk} \rangle]_q \approx \frac{q}{2\pi} \cdot \sin\left(\frac{2\pi}{q} \cdot \langle \mathsf{ct}, \mathsf{sk} \rangle\right).$$

which is periodic modulo q and closely approximates $\langle \mathsf{ct}, \mathsf{sk} \rangle$ near 0. Given that $m \ll q$, the polynomial approximation only has to be good for small values modulo q. The bootstrapping procedure homomorphically evaluates this polynomial approximation, producing a refreshed ciphertext that encrypts the same message but under a larger modulus, thereby enabling further computation. The error resulting from bootstrapping is kept sufficiently small to preserve plaintext precision.

The authors also demonstrated how to refresh packed ciphertexts containing n slots with $\mathcal{O}(n)$ complexity. In particular, they use a linear transformation that moves the polynomial coefficients into the plaintext slots via the *coefficient-to-slot* (CoeffToSlot) procedure, and reverses the latter using the *slot-to-coefficient* (SlotToCoeff) operation. The complexity of these operations was further reduced to $\mathcal{O}(\log n)$ in [CHH18,CCS19]. We recall the original CKKS bootstrapping procedure in more details in Appendix C, first for a single slot, and then for multiple slots, using the CoeffToSlot and SlotToCoeff operations.

Our new bootstrapping algorithm. Our main contribution is a new bootstrapping algorithm for CKKS. It consists in embedding the additive group $\mathbb{Z}_q$ into the complex roots of unity, using the homomorphism:

$$f : \mathbb{Z}_q \longrightarrow \mathbb{C}, \qquad x \longmapsto \exp(2i\pi \cdot x/q).$$

Considering for simplicity an LWE ciphertext $\boldsymbol{c}$, we can compute the decryption equation $m = \langle \boldsymbol{s}, \boldsymbol{c} \rangle = \sum_{k=0}^{N-1} s_k c_k \pmod q$ over this multiplicative group, using $s_k \in \{0, 1\}$ as a selector:

$$\exp(2i\pi \cdot \langle \boldsymbol{s}, \boldsymbol{c} \rangle / q) = \prod_{k=0}^{N-1} \exp(2i\pi \cdot c_k s_k / q) = \prod_{k=0}^{N-1} \left(1 + (\exp(2i\pi \cdot c_k/q) - 1) \cdot s_k\right).$$

Moreover, considering a message $m \in \mathbb{Z}$ such that $m = [\langle \boldsymbol{c}, \boldsymbol{s} \rangle]_q \ll q$, we can use the approximation $\exp(x) \simeq 1 + x$ for small x, which yields:

$$1 + m \cdot \frac{2i\pi}{q} \simeq \prod_{k=0}^{N-1} (1 + (\exp(2i\pi \cdot c_k/q) - 1) \cdot s_k) \tag{1}$$

We claim that this provides a new bootstrapping equation for CKKS, as it can be homomorphically evaluated to provide a refreshed ciphertext. Specifically, it suffices to encrypt each s_k, and following (1), homomorphically multiply it by $\exp(2i\pi \cdot c_k/q) - 1$. The full product is then computed homomorphically using CKKS multiplications. Eventually, by extracting the imaginary part and applying appropriate scaling, we obtain a new ciphertext encrypting the same $m \in \mathbb{Z}$, but under a larger modulus $q \cdot p$ than the original modulus q, enabling further computation.

This bootstrapping equation differs significantly from the original CKKS bootstrapping algorithm, as we avoid using a polynomial approximation of the sine function. Instead, the modular reduction modulo q is achieved by directly embedding the ciphertext coefficients $c_k \in \mathbb{Z}$ into the complex circle group, where computations are performed using CKKS' homomorphic operations. Notably, when bootstrapping multiple slots, the *coefficient-to-slot* procedure from the CKKS bootstrapping is dispensable, as the ciphertext is already embedded within the complex slots. However, the *slot-to-coefficient* (STC) operation is still needed to revert to the coefficient representation.

Our approach shares similarities with the blind rotation algorithms used in FHEW/TFHE bootstrapping [DM15,CGGI16], where the additive group $\mathbb{Z}_q$ is embedded into the set Y^i of roots of unity of $X^N + 1$, for $N = k \cdot q$ and $Y = X^{2k}$. In both cases, the computation involves a chain of products with factors selected homomorphically over the encrypted bits of the secret key. However, unlike FHEW/TFHE bootstrapping which can bootstrap only a single small precision message (from Boolean to 5-bit integers), our bootstrapping equation can bootstrap multiple large precision messages in parallel. This is possible because CKKS allows embedding $N/2$ copies of the additive group $\mathbb{Z}_q$ into the $N/2$ complex slots of a single ciphertext.

The goal of this paper is therefore to explore this new bootstrapping approach, prove a corresponding bootstrapping theorem, and introduce several optimizations that make it practical. Finally, we present concrete parameter sets and benchmark results, comparing the performance of our bootstrapping method with that of the original CKKS bootstrapping.

First optimization: packing the secret-key bits. We introduce a first optimization for the homomorphic evaluation of our bootstrapping equation (1), where the N secret-key bits s_i are packed into the $N/2$ slots of CKKS ciphertexts, requiring only two CKKS ciphertexts. The product is then computed using a product operator (corresponding to the standard field norm, see Sec. 3.5), reducing the number of operations from $\mathcal{O}(N)$ to $\mathcal{O}(\log N)$. Additionally, the bootstrapping key size is reduced from N ciphertexts to just two. This packing method is particularly efficient for bootstrapping, as CKKS natively supports complex numbers in the slots, enabling an efficient embedding of $\mathbb{Z}_q$ into the complex roots of unity.

Second optimization: sparse block secret-key. We introduce a second optimization, where the secret key is structured into h blocks, each of size $B = N/h$, with exactly one non-zero bit per block. This technique was previously explored in the context of TFHE bootstrapping [LMSS23]. Leveraging this structure, we can rewrite the bootstrapping equation (1) using partial sums over

each block instead of products. For a packed ciphertext, these sums are efficiently computed using the trace operator, while the remaining products are still handled by the product operator. To apply the trace operator, we re-index the secret-key coordinates such that each block corresponds to a congruence class of indices, over which the trace operator computes the sum. Since most homomorphic multiplications are replaced by additions, this approach reduces the multiplicative depth from $\log_2 N + 1$ to $\log_2 h + 1$. Additionally, the number of homomorphic multiplications decreases from $\mathcal{O}(\log N)$ to $\mathcal{O}(\log h)$, further improving efficiency. We formally describe in Section 4.4 our bootstrapping algorithm for a single slot by combining the two previous optimizations.

Bootstrapping any number of slots. Finally, we extend our bootstrapping algorithm from a single slot to multiple slots, leveraging the Single Instruction Multiple Data (SIMD) capabilities of CKKS. We first generalize the approach to n slots for $n \leq B/2$, where B is the block size in the secret key. To achieve this, we treat each coefficient of the polynomial message as the decryption of an independent LWE ciphertext. The key bits and the corresponding embeddings of the LWE ciphertext coefficients are then packed into the $N/2$ slots of $2n$ CKKS ciphertexts. As before, the sum is computed in two steps: first, by summing across $2n$ distinct ciphertexts, and then by applying the trace operator to the result. Finally, the product is again evaluated using the product operator. For ciphertexts with up to $n \leq N/2$ slots, we apply the above bootstrapping procedure sequentially over ciphertexts containing $B/2$ slots.

Asymptotically, our method has a complexity of $\mathcal{O}(n + \log N)$ homomorphic operations, whereas the original CKKS bootstrapping scales as $\mathcal{O}(\log n + \log N)$. Consequently, the original CKKS bootstrapping is more efficient for large n. However, for a small number of slots, our approach is significantly more efficient in practice due to its lower multiplicative depth. Additionally, our bootstrapping method is highly parallelizable; with n processors, it achieves the same $\mathcal{O}(\log N)$ complexity as the original CKKS bootstrapping.

Concrete parameters and implementation. We provide concrete parameter sets for both the original CKKS bootstrapping and our new bootstrapping, ensuring the same level of security. To determine secure parameters, we use the standard Lattice Estimator [APS15], taking into account the best-known attacks, including the hybrid lattice attack that exploits the sparse key distribution. Additionally, we derive explicit formulas for the number of homomorphic multiplications required in both bootstrapping schemes. These formulas demonstrate that for a small number of slots ($n \leq 16$), our bootstrapping is significantly more efficient.

We also present timing results from an implementation of both bootstrapping methods. Our implementation is written in Python using the SageMath library, which calls the C++ NTL library for polynomial arithmetic, with only a small portion of the runtime spent in the SageMath wrapper. Our benchmarks confirm the theoretical formulas for homomorphic operation counts. In practice, thanks to its lower multiplicative depth, our bootstrapping achieves a $7\times$ speedup for a small number of slots.

Related work. A significant portion of CKKS ' runtime is attributed to its linear transformations, specifically SlotToCoeff and CoeffToSlot, which perform homomorphic encoding and decoding. These transformations can be executed using matrix multiplication, requiring $\mathcal{O}(n)$ homomorphic operations [CHK+18a]. This was later optimized to $\mathcal{O}(\log n)$ by employing a homomorphic Discrete Fourier Transform (DFT) [CHH18,CCS19]. In this paper, we utilize the $\mathcal{O}(\log n)$ algorithm.

Efficient implementations of CKKS often use a *residue number system* (RNS), which involves decomposing the ciphertext modulus into several smaller moduli. This allows operations to be performed on native 64-bit word-sized integers, providing significant computational advantages. The first RNS-CKKS scheme was introduced in [CHK+18b], achieving an order of magnitude improvement in efficiency, by adapting the double-CRT representation in the BGV scheme [GHS12b]. Further optimizations were described [KPP22], and the most recent advancements were presented in [CCKK24]. Another optimization involves GPU-accelerated implementations, as described in [SYL+23] and [JKA+21]. Additionally, the authors of [CCKS23] proposed a new technique for performing a ciphertext multiplication that consumes fewer bits of the ciphertext modulus. All these optimizations are fully compatible with our new bootstrapping algorithm.

The CKKS approximation of the modular reduction function, specifically the EvalMod algorithm, has been optimized in [KPK+23] and [LLK+22] to consume fewer bits of the ciphertext modulus. The approximation of the sine function has been improved in [LLL+21] and [JM22], improving the bootstrapping precision.

In [KDE+21], the authors introduced a new bootstrapping approach for BGV, BFV and CKKS, based on the blind rotation technique used in FHEW/TFHE bootstrapping. For CKKS, their approach enables high-precision bootstrapping with a relatively small ring dimension of $N = 2^{13}$. Since FHEW/TFHE can only bootstrap a single slot, the ciphertext is split according to the number of slots. Then, each part is bootstrapped independently, and the results are recombined into a single ciphertext. However, this approach requires $\mathcal{O}(N)$ homomorphic operations for a single slot, which could render it impractical for CKKS. The authors do not provide timing results for their implementation.

In [BCC+22], the authors introduced a method for achieving high-precision bootstrapping by combining multiple low-precision bootstrapping steps. As a result, it is possible to keep the same parameters (in particular, the same modulus size) for bootstrapping while achieving higher precision, albeit with the trade-off of requiring more iterations. Finally, it was recently shown that CKKS can be used in a black-box manner to perform computations on binary data [DMPS24]. This was further improved in [BCKS24] with a CKKS bootstrapping algorithm designed specifically for ciphertexts encoding binary data.

2 Preliminaries

Basic notations. For a real number r, we denote by $\lfloor r \rceil$ the nearest integer to r, rounding upwards. We denote vectors in bold, and $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ denotes the scalar product of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. For an integer q, we use $\mathbb{Z} \cap (-q/2, q/2]$ as a representative of $\mathbb{Z}_q$, and use $[z]_q$ or $z \bmod q$ to denote the central reduction of the integer z modulo q into that interval. For a finite set S, we denote by $v \leftarrow S$ the sampling of v uniformly at random from S. By $v \leftarrow \mathcal{D}$ we also denote sampling v from a distribution $\mathcal{D}$. We utilize the same notations for sampling coefficients of vectors/polynomials independently and identically from a set/distribution. For an integer $\kappa > 0$, we let $v \leftarrow \chi_\kappa$ be sampled from the *centered binomial distribution* χ_κ. It is defined as outputting $v := \sum_{j=1}^{\kappa}(a_i - b_i)$, where $\boldsymbol{a}, \boldsymbol{b} \leftarrow \{0,1\}^\kappa$. In this case, it holds that $|v| \leq \kappa$. We denote by λ the security parameter; all known attacks should take $\Omega(2^\lambda)$ bit operations.

Cyclotomic rings. For a power-of-two N, we set $\mathcal{R} := \mathbb{Z}[X]/(X^N + 1)$ as the $2N$-th cyclotomic ring. We also write $\mathcal{R}_q := \mathbb{Z}_q[X]/(X^N + 1)$ for the residue ring of $\mathcal{R}$ modulo q. As previously, coefficients of elements of $\mathcal{R}_q$ are represented as integers in $(-q/2, q/2]$. We represent an arbitrary

element of the cyclotomic ring $\mathcal{S} = \mathbb{R}[X]/(X^N + 1)$ as a polynomial $a(X) = \sum_{j=0}^{N-1} a_j X^j$ of degree strictly less than N, and identify it with its coefficient vector $(a_0, \ldots, a_{N-1}) \in \mathbb{R}^N$. We consider the norms $\|a\|_\infty = \max_{i \in \{0,\ldots,N-1\}} |a_i|$ and $\|a\|_1 = \sum_{i=0}^{N-1} |a_i|$ on the coefficient vector of a.

3 The CKKS scheme

3.1 Basic RLWE-based encryption

In [CKKS17], the authors described a fully homomorphic encryption scheme designed for approximate arithmetic. Given ciphertexts encrypting m_1 and m_2, the scheme enables secure computations of encrypted approximate values of $m_1 + m_2$ and $m_1 \cdot m_2$, with a predefined level of precision. We recall the concrete instantiation based on the BGV scheme [BGV11], using the multiplication from [GHS12b] based on raising the ciphertext modulus. Note that the CKKS scheme differs from BGV in the sense that the plaintext space is the polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, instead of $\mathbb{Z}_t[X]/(X^N + 1)$ for a small t.

Let q be an integer and let s be a secret key with small components in $\mathcal{R}$. A CKKS ciphertext $\mathsf{ct} = (c_0, c_1)$ for a plaintext $m \in \mathcal{R}$ satisfies:

$$\langle \mathsf{ct}, \mathsf{sk} \rangle = c_0 + c_1 \cdot s = m + e \pmod{(q, X^N + 1)}$$

for some small error $e \in \mathcal{R}$. This implies that during decryption, the message m is not recovered exactly, as the low-order bits of the coefficients of $m(X)$ are masked by the error $e(X)$. Such an error e is inserted to guarantee the security of the hardness assumption of the ring learning with error (RLWE) problem. During homomorphic operations, the coefficients of the plaintext polynomial $m(X)$ should remain sufficiently small compared to the ciphertext modulus q.

Definition 1 (RLWE). *For an integer q, a ring dimension N and distributions $\mathcal{D}_s$ and $\mathcal{D}_e$, the decisional* RLWE *problem consists, for $s \leftarrow \mathcal{D}_s$, in distinguishing polynomially many samples $(a, a \cdot s + e) \in \mathcal{R}_q^2$ from uniform in $\mathcal{R}_q^2$, where $a \leftarrow \mathcal{R}_q$ and $e \leftarrow \mathcal{D}_e$.*

We now formally describe the scheme, following [CHK+18a], beginning with the construction of a *leveled* homomorphic encryption scheme for approximate arithmetic. The plaintext space is $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, while the ciphertext space is $\mathcal{R}_q^2$. The scheme defines various layers of moduli, *i.e.* $q_\ell = q_0 \cdot \Delta^\ell$ for $1 \leq \ell \leq L$ and a base Δ. For the error sampling, for simplicity we use the centered binomial distribution χ_κ with parameter κ, rather than a discrete Gaussian distribution. For $h \in \mathbb{N}$, we let $\mathcal{HW}(h)$ be the set of ternary vectors in $\{0, \pm 1\}^N$ whose Hamming weight is exactly h. For a real $0 \leq \rho \leq 1$, let $\mathcal{ZO}(\rho)$ denote the distribution that outputs a vector in $\{0, \pm 1\}^N$, where each entry is sampled independently, with probability $\rho/2$ for both -1 and 1, and probability $1 - \rho$ for 0.

- $\mathsf{KeyGen}(1^\lambda)$:
  - For a base Δ, a base modulus q_0 and an integer L, let $q_\ell = q_0 \cdot \Delta^\ell$ for $\ell = 1, \ldots, L$. Given the security parameter λ, choose a power-of-two N, an integer P, and an integer $\kappa > 0$ for an RLWE problem that achieves a security level of λ-bits, for a modulus $P \cdot q_L$ and a ring degree N.
  - Sample $s \leftarrow \mathcal{HW}(h)$, $a \leftarrow \mathcal{R}_{q_L}$ and $e \leftarrow \chi_\kappa$. Set the secret key as $\mathsf{sk} \leftarrow (1, s)$ and the public key as $\mathsf{pk} \leftarrow (b, a) \in \mathcal{R}_{q_L}^2$ where $b = -as + e \pmod{q_L}$.

- $\mathsf{KSGen}_{\mathsf{sk}}(s')$: for $s' \in \mathcal{R}$, sample $a \leftarrow \mathcal{R}_{P \cdot q_L}$ and $e' \leftarrow \chi_\kappa$. Return the switching key as $\mathsf{swk} \leftarrow (b', a') \in \mathcal{R}_{P \cdot q_L}$ where $b' = -a's + e' + Ps' \pmod{P \cdot q_L}$.
  - Set the evaluation key as $\mathsf{evk} \leftarrow \mathsf{KSGen}_{\mathsf{sk}}(s^2)$.
- $\mathsf{Enc}_{\mathsf{pk}}(m)$: for $m \in \mathcal{R}$, sample $v \leftarrow \mathcal{ZO}(1/2)$ and $e_0, e_1 \leftarrow \chi_\kappa$. Output $\mathsf{ct} = v \cdot \mathsf{pk} + (m + e_0, e_1) \pmod{q_L}$.
- $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$: for $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_{q_\ell}^2$, output $m = c_0 + c_1 \cdot s \pmod{q_\ell}$.

We now define the homomorphic operations $\mathsf{Add}(\mathsf{ct}_1, \mathsf{ct}_2)$ and $\mathsf{Mult}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$. We also define the simpler external multiplication $\mathsf{ExtMult}$, which multiplies a ciphertext by a plaintext $v \in \mathcal{R}$.

- $\mathsf{Add}(\mathsf{ct}_1, \mathsf{ct}_2)$: for $\mathsf{ct}_1, \mathsf{ct}_2 \in \mathcal{R}_{q_\ell}^2$, output $\mathsf{ct}_{\mathsf{add}} = \mathsf{ct}_1 + \mathsf{ct}_2 \pmod{q_\ell}$.
- $\mathsf{Mult}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$: for $\mathsf{ct}_1 = (b_1, a_1)$, $\mathsf{ct}_2 = (b_2, a_2) \in \mathcal{R}_{q_\ell}^2$, let $(d_0, d_1, d_2) = (b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2) \pmod{q_\ell}$. Output $\mathsf{ct}_{\mathsf{mult}} \leftarrow (d_0, d_1) + \lfloor P^{-1} \cdot d_2 \cdot \mathsf{evk} \rceil \pmod{q_\ell}$.
- $\mathsf{ExtMult}(\mathsf{ct}, v)$: for $\mathsf{ct} = (b, a) \in \mathcal{R}_{q_\ell}^2$ and $v \in \mathcal{R}$, output $\mathsf{ct}_{\mathsf{extmult}} = (b \cdot v, a \cdot v) \pmod{q_\ell}$.

The rescaling procedure below transforms an encryption of m modulo q into an encryption of m/Δ, while also scaling its inherent noise e to around e/Δ. As will be seen in Section 3.2, the composition of homomorphic multiplication and rescaling enables to mimic fixed-point arithmetic, while managing the error growth.

- $\mathsf{RS}_{\ell \to \ell'}(\mathsf{ct})$: for a ciphertext $\mathsf{ct} \in \mathcal{R}_{q_\ell}^2$ at level ℓ, output $\mathsf{ct}' = \left\lfloor \Delta^{\ell' - \ell} \cdot \mathsf{ct} \right\rceil \pmod{q_{\ell'}}$. We omit the subscript $\ell \to \ell'$ when $\ell' = \ell - 1$.

In [CKKS17], the authors state some lemmata for estimating the noise, although their noise estimation is only heuristic. Our lemmata below are with rigorous proofs, though with larger upper bounds; we refer to Appendix A for the proofs.

Lemma 1. *Let* $\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$ *be an encryption of* $m \in \mathcal{R}$. *Then* $\langle \mathsf{ct}, \mathsf{sk} \rangle = m + e_{\mathsf{clean}} \pmod{q_L}$ *for some clean encryption error* $e_{\mathsf{clean}} \in \mathcal{R}$ *satisfying* $\|e_{\mathsf{clean}}\|_\infty \leq B_{\mathsf{clean}} := 3N\kappa$.

Lemma 2. *Let* $\mathsf{ct}' \leftarrow \mathsf{RS}_{\ell \to \ell'}(\mathsf{ct})$ *for a ciphertext* $\mathsf{ct} \in \mathcal{R}_{q_\ell}^2$. *Then* $\langle \mathsf{ct}', \mathsf{sk} \rangle = \lfloor \langle \mathsf{ct}, \mathsf{sk} \rangle \cdot q_{\ell'} / q_\ell \rceil + e_{\mathsf{rs}} \pmod{q_{\ell'}}$ *for some rescaling error* $e_{\mathsf{rs}} \in \mathcal{R}$ *satisfying* $\|e_{\mathsf{rs}}\|_\infty \leq B_{\mathsf{rs}} := 2N$.

Lemma 3. *Let* $\mathsf{ct}_{\mathsf{mult}} \leftarrow \mathsf{Mult}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$ *for two ciphertexts* $\mathsf{ct}_1, \mathsf{ct}_2 \in \mathcal{R}_{q_\ell}^2$. *If* $P \geq N q_\ell \kappa$, *then* $\langle \mathsf{ct}_{\mathsf{mult}}, \mathsf{sk} \rangle = \langle \mathsf{ct}_1, \mathsf{sk} \rangle \cdot \langle \mathsf{ct}_2, \mathsf{sk} \rangle + e_{\mathsf{mult}} \pmod{q_\ell}$ *for some* e_{mult} *satisfying* $\|e_{\mathsf{mult}}\|_\infty \leq B_{\mathsf{mult}} := 2N$.

Writing $\langle \mathsf{ct}_1, \mathsf{sk} \rangle = m_1 + e_1 \pmod{q_\ell}$ and $\langle \mathsf{ct}_2, \mathsf{sk} \rangle = m_2 + e_2 \pmod{q_\ell}$, the previous lemma shows that only the high-order bits of $m_1 m_2$ can be recovered, as we get $\langle \mathsf{ct}_{\mathsf{mult}}, \mathsf{sk} \rangle = (m_1 + e_1) \cdot (m_2 + e_2) + e_{\mathsf{mult}} = m_1 m_2 + e^\star \pmod{q_\ell}$, and the low-order bits of $m_1 m_2$ are masked by the quite large error $e^\star = e_1 m_2 + e_2 m_1 + e_1 e_2 + e_{\mathsf{mult}}$. Applying Lemma 2, one can then perform a modulus switching to rescale the product, which then gives:

$$\langle \mathsf{ct}'_{\mathsf{mult}}, \mathsf{sk} \rangle = \lfloor m_1 m_2 / \Delta \rceil + e'_{\mathsf{mult}} \pmod{q_{\ell-1}}.$$

Therefore, one obtains an encryption of the scaled product $\lfloor m_1 m_2 / \Delta \rceil$.

3.2 Encrypting single real numbers

For simplicity, we first recall the CKKS encryption of single real numbers. We then recall the encryption of $N/2$ complex numbers in parallel in Section 3.3.

Specifically, the CKKS scheme can encrypt a real value $x \in \mathbb{R}$ with a specified precision, by rounding to an integer after scaling. This enables to define (approximate) homomorphic operations directly over $\mathbb{R}$. More precisely, to encode a real value $x \in \mathbb{R}$ with precision Δ, we use

$$\mathsf{Ecd}(x) = \lfloor x \cdot \Delta \rceil. \tag{2}$$

Therefore, to encrypt $x \in \mathbb{R}$, we first encode x into $m = \mathsf{Ecd}(x)$, and then encrypt m into $\mathsf{ct} = \mathsf{Enc}_{\mathsf{pk}}(m)$, which gives $\langle \mathsf{ct}, \mathsf{sk} \rangle = \mathsf{Ecd}(x) + e \pmod{q_L}$. Since $\mathsf{Ecd}(x) \in \mathbb{Z}$, we are using only a single coefficient of the plaintext space $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$. Formally, we define the following encryption procedure for real numbers, and the homomorphic multiplication of two ciphertexts of real numbers.

- $\mathsf{EncR}_{\mathsf{pk}}(x)$: for $x \in \mathbb{R}$, return $\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathsf{Ecd}(x))$.
- $\mathsf{MultR}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$: return $\mathsf{RS}(\mathsf{Mult}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2))$
- $\mathsf{ExtMultR}(\mathsf{ct}, v)$: return $\mathsf{RS}(\mathsf{ExtMult}(\mathsf{ct}, v))$

Thanks to the rescaling by a factor Δ, given as input two ciphertexts ct_1, ct_2 encrypting $x_1, x_2 \in \mathbb{R}$ under a modulus q_ℓ, we can obtain a new ciphertext $\mathsf{ct} = \mathsf{MultR}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$ encrypting $\mathsf{Ecd}(x_1) \cdot \mathsf{Ecd}(x_2)/\Delta \simeq \mathsf{Ecd}(x_1 \cdot x_2)$, therefore an encryption of the product $x_1 \cdot x_2 \in \mathbb{R}$. However, the new encryption is under a smaller modulus $q_{\ell-1} = q_\ell/\Delta$. This is captured in the following lemma; see Fig. 2 for an illustration.

Lemma 4. *Let* $\mathsf{ct}_{\mathsf{multR}} \leftarrow \mathsf{MultR}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$ *for two ciphertexts* $\mathsf{ct}_1, \mathsf{ct}_2 \in \mathcal{R}_{q_\ell}^2$ *such that* $\langle \mathsf{ct}_i, \mathsf{sk} \rangle = \mathsf{Ecd}(x_i) + e_i \pmod{q_\ell}$ *for* $i = 1, 2$*, for* $x_i \in \mathbb{R}$ *with* $|x_i| \le \nu < N$*, and* $\|e_i\|_\infty \le E$*, where* $E^2 \le \Delta$*. Then* $\langle \mathsf{ct}_{\mathsf{mult}}, \mathsf{sk} \rangle = \mathsf{Ecd}(x_1 \cdot x_2) + e_{\mathsf{multR}} \pmod{q_{\ell-1}}$ *for some* e_{multR} *satisfying* $\|e_{\mathsf{multR}}\|_\infty \le 2\nu E + 8N$*.*

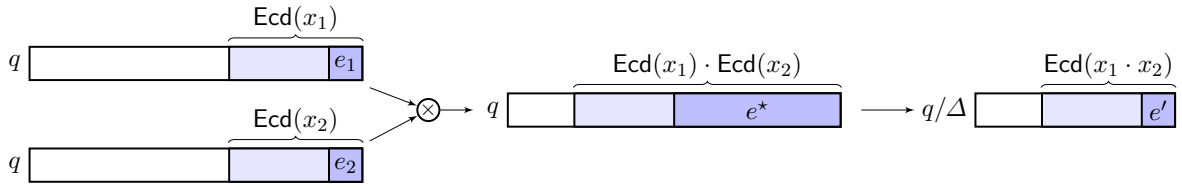


Fig. 2. Homomorphic multiplication of real numbers in CKKS: given two encryptions of $x_1, x_2 \in \mathbb{R}$, we obtain an encryption of $x_1 \cdot x_2 \in \mathbb{R}$, albeit for a smaller modulus q/Δ.

3.3 Packing method for parallel encryption of complex numbers

In [CKKS17,CHK+18a], the authors described a packing method to encrypt multiple messages in a single ciphertext. Through a ring isomorphism, the set $\mathcal{S} = \mathbb{R}[X]/(X^N + 1)$ can be identified with the complex coordinate space $\mathbb{C}^{N/2}$. This enables to encrypt $N/2$ complex numbers is a single ciphertext, and perform parallel operations in a SIMD manner.

More precisely, the authors considered a variant of the Fourier transform

$$\mathsf{DFT} : \mathbb{R}[X]/(X^N+1) \to \mathbb{C}^{N/2}, \qquad p \mapsto \left(p\left(\zeta^{5^0}\right), p\left(\zeta^{5^1}\right), \ldots, p\left(\zeta^{5^{N/2-1}}\right)\right)$$

where ζ is a primitive $(2N)$-th root of unity. We denote by $\mathsf{iDFT} : \mathbb{C}^{N/2} \to \mathcal{S}$ the inverse transform. The DFT function is a ring isomorphism from the ring $\mathcal{S}$ (the coefficients space) to the ring of complex vectors in $\mathbb{C}^{N/2}$ (the slots space). Its purpose is therefore to map polynomial addition/multiplication in the ring $\mathcal{S}$ to component-wise addition/multiplication in $\mathbb{C}^{N/2}$.

Encoding. For RLWE-type schemes, we must encode our $N/2$ complex values into polynomials with integer coefficients in the plaintext ring $\mathcal{R} = \mathbb{Z}[X]/(X^N+1)$, instead of $\mathbb{R}[X]/(X^N+1)$. Therefore, the encoding map $\mathsf{Ecd} : \mathbb{C}^{N/2} \to \mathcal{R}$ uses an integer approximation of iDFT, with a scaling factor Δ:

$$\forall \boldsymbol{z} \in \mathbb{C}^{N/2} : \mathsf{Ecd}(\boldsymbol{z}) = \lfloor \Delta \cdot \mathsf{iDFT}(\boldsymbol{z}) \rceil$$

and the corresponding decoding map $\mathsf{Dcd} : \mathcal{R} \to \mathbb{C}^{N/2}$ is defined as $\forall p \in \mathcal{R} : \mathsf{Dcd}(p) = \frac{1}{\Delta} \cdot \mathsf{DFT}(p)$. We denote by $\times$ the scaled multiplication over $\mathcal{R}$, *i.e.* $p_1 \times p_2 = \lfloor p_1 \cdot p_2/\Delta \rceil$ for any $p_1, p_2 \in \mathcal{R}$. We then have the approximate homomorphic properties:

$$\mathsf{Ecd}(\boldsymbol{z}_1 + \boldsymbol{z}_2) \simeq \mathsf{Ecd}(\boldsymbol{z}_1) + \mathsf{Ecd}(\boldsymbol{z}_2)$$
$$\mathsf{Ecd}(\boldsymbol{z}_1 \odot \boldsymbol{z}_2) \simeq \mathsf{Ecd}(\boldsymbol{z}_1) \times \mathsf{Ecd}(\boldsymbol{z}_2)$$

where the vectors $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ are added and multiplied component-wise. Therefore, to perform parallel addition/multiplication on $N/2$ complex values, we can first encode them as a polynomial $p(X)$ using Ecd, and then perform ring addition / multiplication in $\mathcal{R}$. Eventually, we can decode the polynomial into $N/2$ complex values using Dcd; see Fig. 3 for an illustration.

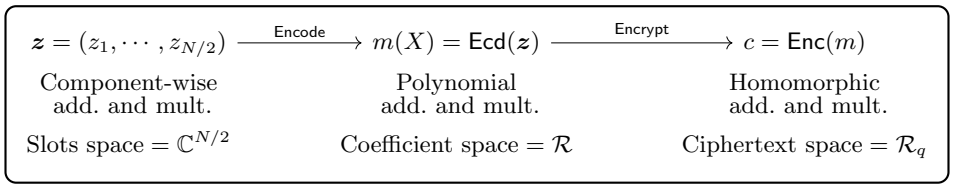


Fig. 3. Packing method for CKKS.

Sparse packing. For efficiency reasons, it can be advantageous to encode fewer than $N/2$ components. We consider a power-of-two n dividing $N/2$. The sub-ring $\mathbb{R}[X^{N/(2n)}]/(X^N+1) \simeq \mathbb{R}[Y]/(Y^{2n}+1)$ is isomorphic to $\mathbb{C}^n$. In particular, if $p \in \mathbb{R}[X^{N/(2n)}]/(X^N+1)$, then $\mathsf{DFT}(p)$ contains $N/(2n)$ repetitions of the same n components. We can therefore extend the encoding map Ecd to encode vectors of n complex values instead of $N/2$, as follows:

$$\forall \boldsymbol{z} \in \mathbb{C}^n : \mathsf{Ecd}(\boldsymbol{z}) = \lfloor \Delta \cdot \mathsf{iDFT}(\underbrace{\boldsymbol{z}, \ldots\ldots, \boldsymbol{z}}_{N/(2n) \text{ rep.}}) \rceil \in \mathbb{Z}[X^{N/(2n)}]/(X^N+1). \tag{3}$$

In particular, for $n = 1$, we have $\mathsf{Ecd}(z) = \lfloor \Delta \cdot \mathsf{Re}(z) \rceil + X^{N/2} \cdot \lfloor \Delta \cdot \mathsf{Im}(z) \rceil$ for any $z \in \mathbb{C}$. Therefore, the encoding map Ecd defined in (3) is a generalization of the encoding function defined in (2) for encoding $x \in \mathbb{R}$.

Message packing into a ciphertext. Thanks to this generalized encoding function for any $\boldsymbol{z} \in \mathbb{C}^n$, we can extend the encryption procedure $\mathsf{EncR}_{\mathsf{pk}}(x)$ from Section 3.2 to any $\boldsymbol{z} \in \mathbb{C}^n$, for any power-of-two $1 \leq n \leq N/2$:

- $\mathsf{EncR}_{\mathsf{pk}}(\boldsymbol{z})$: for $\boldsymbol{z} \in \mathbb{C}^n$, return $\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathsf{Ecd}(\boldsymbol{z}))$.

To perform a homomorphic multiplication, we employ the same rescaled multiplication procedure $\mathsf{MultR}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$ as in the previous section. Given ciphertexts $\mathsf{ct}_1 \leftarrow \mathsf{EncR}_{\mathsf{pk}}(\boldsymbol{z}_1)$ and $\mathsf{ct}_2 \leftarrow \mathsf{EncR}_{\mathsf{pk}}(\boldsymbol{z}_2)$, when applying $\mathsf{Add}(\mathsf{ct}_1, \mathsf{ct}_2)$ and $\mathsf{MultR}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$, the vectors $\boldsymbol{z}_1, \boldsymbol{z}_2 \in \mathbb{C}^n$ are homomorphically added and multiplied component-wise; see Figure 3 for an illustration.

3.4 Rotation and conjugation of the slots

As demonstrated in [CKKS17], rotation and conjugation of slots can be performed homomorphically by applying an automorphism Ψ to both components of a ciphertext. However, Ψ is also applied to the secret key, meaning the new ciphertext only decrypts correctly with the modified key $s' = \Psi(s)$. Therefore, a key-switching procedure is required to revert to the original key s.

Key switching. The goal of key switching is to convert a ciphertext under a secret key $\mathsf{sk}' = (1, s')$ into a ciphertext for the same message with respect to another secret key $\mathsf{sk} = (1, s)$. The switching key is generated by the procedure $\mathsf{swk} \leftarrow \mathsf{KSGen}_{\mathsf{sk}}(s')$. Formally, the key switching proceeds as follows:

- $\mathsf{KS}_{\mathsf{swk}}(\mathsf{ct})$: for $\mathsf{ct} = (c_0, c_1)$, return $\mathsf{ct}' \leftarrow (c_0, 0) + \lfloor P^{-1} \cdot c_1 \cdot \mathsf{swk} \rceil \pmod{q_\ell}$.

The noise growth from key switching is captured in the following lemma.

Lemma 5 (Key switching). *Let $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_{q_\ell}^2$ be a ciphertext with respect to a secret key $\mathsf{sk}' = (1, s')$ and let $\mathsf{swk} \leftarrow \mathsf{KSGen}_{\mathsf{sk}}(s')$. If $P \geq N q_\ell \kappa$, then $\mathsf{ct}' \leftarrow \mathsf{KS}_{\mathsf{swk}}(\mathsf{ct})$ satisfies $\langle \mathsf{ct}', \mathsf{sk} \rangle = \langle \mathsf{ct}, \mathsf{sk}' \rangle + e_{\mathsf{ks}} \pmod{q_\ell}$ for some $e_{\mathsf{ks}} \in \mathcal{R}$ with $\|e_{\mathsf{ks}}\|_\infty \leq 2N$.*

Rotation. For $\mathcal{S} = \mathbb{R}[X]/(X^N + 1)$, the automorphism $\Psi_r : \mathcal{S} \to \mathcal{S}$, given $m(X) \in \mathcal{S}$, returns $m(X^{5^r}) \pmod{X^N + 1}$. It induces a rotation by r positions to the left on the vector of encoded values. Namely for any $m(X) \in \mathcal{S}$, letting $\mathsf{DFT}(m) = (z_j)_{0 \leq j < N/2}$, we get:

$$\begin{aligned}\mathsf{DFT}(\Psi_r(m)(X)) &= (\Psi_r(m)(\zeta^{5^j}))_{0 \leq j < N/2} = (m((\zeta^{5^j})^{5^r}))_{0 \leq j < N/2} \\ &= (m(\zeta^{5^{r+j}}))_{0 \leq j < N/2} = (z_{r+j \bmod N/2})_{0 \leq j < N/2}.\end{aligned}$$

Conjugation. We also consider the automorphism $\mathsf{conj} : \mathcal{S} \to \mathcal{S}$ defined as $\mathsf{conj}(m)(X) = m(X^{-1}) \pmod{X^N + 1}$. It enables to compute the conjugate of the values encoded into a polynomial, namely, we have that $\mathsf{DFT}(\mathsf{conj}(m)) = \overline{\mathsf{DFT}(m)}$. We also denote by $\mathsf{Im2} : \mathcal{S} \to \mathcal{S}$ the operator

$$\mathsf{Im2}(m) = -X^{N/2} \cdot (m - \mathsf{conj}(m)).$$

Since $-\Delta \cdot X^{N/2} = \mathsf{Ecd}(1/i)$, given any $\boldsymbol{z} \in \mathbb{C}^{N/2}$, we have $\mathsf{DFT}(\mathsf{Im2}(\mathsf{iDFT}(\boldsymbol{z}))) = 2 \cdot \mathsf{Im}(\boldsymbol{z})$. Therefore, the $\mathsf{Im2}$ operator enables to extract the imaginary part of the slots, up to a factor 2. Similarly, we define the $\mathsf{Re2}$ operator with $\mathsf{Re2}(m) = \mathsf{conj}(m) + m$, so that $\mathsf{DFT}(\mathsf{Re2}(\mathsf{iDFT}(\boldsymbol{z}))) = 2 \cdot \mathsf{Re}(\boldsymbol{z})$.

Homomorphic rotation/conjugation of the slots. We summarize below the procedures for homomorphic rotation/conjugation of the slots for a ciphertext $\mathsf{ct} = (c_0, c_1)$:

- $\mathsf{GenRot}_{\mathsf{sk}}(r)$: generate the rotation key $\mathsf{rk}_r \leftarrow \mathsf{KSGen}_{\mathsf{sk}}(\Psi_r(s))$.
- $\mathsf{Rot}(\mathsf{ct}, r)$: return $\mathsf{KS}_{\mathsf{rk}_r}((\Psi_r(c_0), \Psi_r(c_1)))$.
- $\mathsf{GenConj}_{\mathsf{sk}}$: generate the conjugation key $\mathsf{ck} \leftarrow \mathsf{KSGen}_{\mathsf{sk}}(\mathsf{conj}(s))$.
- $\mathsf{Conj}(\mathsf{ct})$: return $\mathsf{KS}_{\mathsf{ck}}((\mathsf{conj}(c_0), \mathsf{conj}(c_1)))$.
- $\mathsf{Im2}(\mathsf{ct})$: return the ciphertext $\mathsf{ExtMult}(\mathsf{Conj}(\mathsf{ct}) - \mathsf{ct}, X^{N/2})$.

3.5 The trace and product operators

Summing the slots. We recall how to homomorphically compute partial or full summation of the slots. The sum of all slots could be simply obtained by repeatedly rotating the slots by one position and then accumulating the result by summing. However, this would require $N/2 - 1$ rotations for $N/2$ slots, which would be impractical. In the following, we recall the technique from [CHK+18a, Alg. 2], which uses only $\mathcal{O}(\log N)$ homomorphic operations. Summing all slots is algebraically known as the field trace operator [CDKS20].

Formally, for a power-of-two integer a dividing $N/2$, using the Ψ_r automorphism from the previous section, we define the operator $\mathsf{Tr}_{N/2\to a}$ as

$$\mathsf{Tr}_{N/2\to a} = (\mathsf{id} + \Psi_a) \circ (\mathsf{id} + \Psi_{2a}) \circ (\mathsf{id} + \Psi_{4a}) \circ \cdots \circ (\mathsf{id} + \Psi_{N/4}).$$

We show in Appendix B.1 that the $\mathsf{Tr}_{N/2\to a}$ operator computes partial sums of the slots sharing the same index modulo a:

$$\mathsf{Tr}_{N/2\to a}(\mathsf{iDFT}(z_0, \ldots, z_{N/2-1})) = \mathsf{iDFT}(\underbrace{\boldsymbol{t}, \ldots\ldots, \boldsymbol{t}}_{N/(2a)\text{ rep.}})$$

for $\boldsymbol{t} = (t_k)_{0\le k<a}$, with $t_k = z_k + z_{k+a} + \cdots + z_{k+(N/(2a)-1)\cdot a}$. Therefore, the $\mathsf{Tr}_{N/2\to a}$ operator outputs an encoding of the a slots $t_0, \ldots, t_{a-1}$, which are repeated $N/(2a)$ times to get $N/2$ slots. The $\mathsf{Tr}_{N/2\to a}$ operator can be homomorphically evaluated, requiring $\log_2(N/(2a))$ automorphisms and key switchings, while consuming no multiplicative levels; see Appendix B.1.

Multiplying the slots. Similarly to the trace operator, we can define the product operator to compute the partial or full product of all slots, simply by replacing sums by scaled products in the definition. For powers-of-two $b < a \le N/2$, we define the $\mathsf{Pr}_{a\to b}$ operator as

$$\mathsf{Pr}_{a\to b} := (\mathsf{id} \times \Psi_b) \circ (\mathsf{id} \times \Psi_{2b}) \circ (\mathsf{id} \times \Psi_{4b}) \circ \cdots \circ (\mathsf{id} \times \Psi_{a/2})$$

where as in Section 3.3 we denote by $\times$ the scaled multiplication over $\mathcal{R}$, with $p_1 \times p_2 = \lfloor p_1 \cdot p_2/\Delta \rceil$ for any $p_1, p_2 \in \mathcal{R}$. Letting $\boldsymbol{z} \in \mathbb{C}^a$, the $\mathsf{Pr}_{a\to b}$ operator computes partial products of the slots sharing the same index modulo b:

$$\mathsf{Pr}_{a\to b}(\mathsf{Ecd}(\boldsymbol{z})) \simeq \mathsf{Ecd}(\boldsymbol{t})$$

where $\boldsymbol{t} = (t_k)_{0\le k<b}$, with $t_k = z_k \cdot z_{k+b} \cdots z_{k+(a/b-1)\cdot b}$ for $0 \le k < b$. The $\mathsf{Pr}_{a\to b}$ operator can be homomorphically evaluated using $\log_2(a/b)$ automorphisms and multiplications; thus, it consumes $\log_2(a/b)$ levels; see Appendix B.2. Note that the product operator $\mathsf{Pr}_{a\to b}$ corresponds to the standard field norm of $\mathbb{Q}(\zeta_a)$ over $\mathbb{Q}(\zeta_b)$, where ζ_k denotes a primitive k-th root of unity.

3.6 From slots back to polynomial coefficients

Our bootstrapping algorithm for multiple slots requires putting the slots back to the coefficients of a polynomial; this is the SlotToCoeff operation. The original CKKS bootstrapping requires both CoeffToSlot and SlotToCoeff operations [CHK+18a], while our technique only requires SlotToCoeff, so we only describe the latter; the two operations are the inverse of each other.

Formally, $\mathsf{SlotToCoeff}_n$ takes as input a polynomial $v(X)$ encoding n slots $\boldsymbol{t} = (t_0, \ldots, t_{n-1}) \in \mathbb{R}^n$:

$$v(X) = \mathsf{Ecd}(t_0, \ldots, t_{n-1}) \in \mathbb{Z}[X^{N/(2n)}]/(X^N + 1)$$

and outputs a polynomial $m(X) \in \mathbb{Z}[X^{N/n}]/(X^N + 1)$ whose n coefficients are the t_j's, scaled by a factor $\Delta \in \mathbb{Z}$:

$$m(X) = \mathsf{SlotToCoeff}_n(v) = \sum_{j=0}^{n-1} \lfloor \Delta \cdot t_j \rceil \cdot X^{j \cdot N/n} \tag{4}$$

Since SlotToCoeff is a linear map, it can be homomorphically evaluated as a matrix-vector multiplication, requiring $\mathcal{O}(n)$ homomorphic operations [CHK+18a]. However, the homomorphic Discrete Fourier Transform (DFT), which is the primary computation within SlotToCoeff, can be performed more efficiently with only $\mathcal{O}(\log n)$ homomorphic operations [CHH18,CCS19]. In this case, the polynomial coefficients in (4) are recovered in a bit-reversed order. In this paper, we employ the fast DFT-based algorithm [CHH18,CCS19] with complexity $\mathcal{O}(\log n)$. To further reduce its multiplicative depth from $\log_2 n$, we use a radix-4 variant, resulting in a lower depth of $\ell_{\mathsf{CtS}} = \ell_{\mathsf{StC}} = \lfloor (\log_2 n)/2 \rfloor + 1$. We refer to Appendix D for a detailed description.

4 New bootstrapping equation for CKKS: single slot

We now introduce our new bootstrapping algorithm, which is quite different from the original CKKS bootstrapping recalled in the previous section. For simplicity, we first consider a CKKS ciphertext ct encrypting a single slot, that is $\langle \mathsf{ct}, \mathsf{sk} \rangle = m + e \pmod q$ for $m \in \mathbb{Z}$ and $e \in \mathcal{R}$. We will describe the bootstrapping of $n \geq 2$ slots in parallel in Section 5.

An RLWE-format ciphertext can easily be viewed as (multiple) LWE-format ciphertexts. Therefore, we first convert $\mathsf{ct} = (b, a)$ into an LWE ciphertext $\boldsymbol{c} \in \mathbb{Z}_q^N$ for the same $m \in \mathbb{Z}$, *i.e.* we set $\boldsymbol{c} = (a_0 + b_0, -a_{N-1}, -a_{N-2}, \ldots, -a_1)$ and $\boldsymbol{s} = (s_0, \ldots, s_{N-1})$, assuming that $s_0 = 1$. LWE decryption now comprises computing the dot product $\langle \boldsymbol{c}, \boldsymbol{s} \rangle = m + e_0 \pmod q$ for $e_0 \in \mathbb{Z}$. Unlike the original CKKS bootstrapping, which uses sparse ternary secrets, we first assume that $\boldsymbol{s}$ is a random binary vector, not necessarily constrained to a low Hamming weight.

4.1 Our new bootstrapping equation

Our new bootstrapping equation consists in embedding the additive group $\mathbb{Z}_q$ into the multiplicative circle group of complex numbers, using the homomorphism:

$$f : \mathbb{Z}_q \to \mathbb{C}, \qquad x \longmapsto \exp(2i\pi \cdot x/q).$$

We can therefore compute the decryption equation $m = \langle \boldsymbol{s}, \boldsymbol{c} \rangle = \sum_{k=0}^{N-1} s_k c_k \pmod q$ over the multiplicative circle group in $\mathbb{C}$:

$$\exp(2i\pi \cdot m/q) = \prod_{k=0}^{N-1} \exp(2i\pi \cdot c_k s_k/q)$$

Moreover, since $s_k \in \{0,1\}$ works as a selector, we can write:

$$\exp(2i\pi \cdot m/q) = \prod_{k=0}^{N-1} \left(1 + (\exp(2i\pi \cdot c_k/q) - 1) \cdot s_k\right). \tag{5}$$

By isolating the imaginary part and assuming $m \ll q$, we obtain:

$$\frac{2\pi \cdot m}{q} \simeq \sin\left(\frac{2\pi \cdot m}{q}\right) = \mathsf{Im}\left(\prod_{k=0}^{N-1} \left(1 + (\exp(2i\pi \cdot c_k/q) - 1) \cdot s_k\right)\right). \tag{6}$$

We want to obtain an encryption of $m \in \mathbb{Z}$, which corresponds to an encoding of $m/\Delta \in \mathbb{R}$. Therefore, we scale the above equation by a factor $q/(2\pi\Delta)$, and we use the approximation:

$$\frac{q}{2\pi\Delta} \cdot \sin\left(\frac{2\pi m}{q}\right) = \frac{q}{2\pi\Delta} \cdot \left(\frac{2\pi m}{q} + \mathcal{O}\left(\frac{m^3}{q^3}\right)\right) = \frac{m}{\Delta} + \mathcal{O}\left(\frac{m^3}{q^2\Delta}\right). \tag{7}$$

Eventually, we obtain from (6):

$$\frac{m}{\Delta} \simeq \frac{q}{2\pi\Delta} \cdot \mathsf{Im}\left(\prod_{k=0}^{N-1} \left(1 + (\exp(2i\pi \cdot c_k/q) - 1) \cdot s_k\right)\right). \tag{8}$$

From (7), the above approximation requires $m = \mathcal{O}(q^{2/3})$, the same condition as in the original CKKS bootstrapping.

Bootstrapping. We claim that (8) provides a new bootstrapping equation for CKKS, as it can be homomorphically evaluated to provide a new ciphertext for the same $m \in \mathbb{Z}$, but under a larger modulus $q \cdot p$ than the original modulus q. For this, we stress that in the above equation, we consider all variables over $\mathbb{R}$, and since $\exp(2i\pi \cdot c_k/q) \in \mathbb{C}$, we treat its real and imaginary part separately. Each complex multiplication decomposes into 4 multiplications and 2 additions over $\mathbb{R}$. Since a complex addition also decomposes into two real additions, we can work entirely over $\mathbb{R}$. Similarly, we view the final scaling as a multiplication by $q/(2\pi\Delta) \in \mathbb{R}$, and the final output $m/\Delta \in \mathbb{R}$.

Correspondingly, for the homomorphic evaluation of (8), we consider all ciphertexts as encryptions of real numbers, using the encryption function $\mathsf{EncR}(x)$ for $x \in \mathbb{R}$ introduced in Section 3.2. Consequently, all homomorphic operations are performed on encryptions of reals. Eventually, since $m/\Delta \in \mathbb{R}$ is encoded as $m = \mathsf{Ecd}(m/\Delta) \in \mathbb{Z}$, the encryption of m/Δ will correspond to an encryption of the original plaintext $m \in \mathbb{Z}$, but under a larger modulus $q \cdot p$.

Therefore, for bootstrapping, we assume that we are given encryptions $S_k \leftarrow \mathsf{EncR}_{\mathsf{pk}}(s_k)$ of each bit $s_k \in \{0,1\}$ of the secret key $\boldsymbol{s}$. Here, although $s_k \in \{0,1\}$, we again view $s_k \in \mathbb{R}$. Then, by homomorphic multiplications (using $\mathsf{ExtMultR}$) and homomorphic additions, we obtain encryptions of $1 + (\exp(2i\pi \cdot c_k/q) - 1) \cdot s_k$ for each $0 \leq k < N$, while still encrypting the real/imaginary parts separately. The product of the N terms can then be homomorphically computed with a tree of multiplicative depth $\log_2 N$, such that, according to (5), we obtain an encryption of $\exp(2i\pi \cdot m/q)$. Eventually, we only keep the encryption of its imaginary part and scale by $q/(2\pi\Delta)$. According to (8), this provides an encryption of the real m/Δ, or equivalently, an encryption of the integer $m = \mathsf{Ecd}(m/\Delta)$.

The circuit corresponding to (8) has a total multiplicative depth of $\ell = \log_2(N) + 2$. Thus, starting with a modulus $Q = q \cdot p \cdot \Delta^\ell$ for the encryptions S_k, after ℓ rescalings, we end up with an encryption of m under the modulus $q \cdot p > q$. This achieves bootstrapping since there remains a level for a single homomorphic multiplication. For this last level, we can use a smaller scaling factor $p \simeq q^{2/3}$, because the plaintext message m must be encoded under the condition $m = \mathcal{O}(q^{2/3})$ for (7). On the other hand, we must use a scaling factor $\Delta = \mathcal{O}(q)$ for the main bootstrapping evaluation. We prove the following bootstrapping theorem summarizing the above in Appendix E.

Theorem 1 (Bootstrapping). *Given a ciphertext* ct *such that* $\|[\langle \mathsf{ct}, \mathsf{sk}\rangle]_q\|_\infty < q^{2/3}$ *and* $\langle \mathsf{ct}, \mathsf{sk}\rangle = m + e \pmod q$ *for* $m \in \mathbb{Z}$ *and* $e \in \mathcal{R}$*, the above evaluation procedure with* $\Delta \geq \max(4N^2\kappa q, (20N^3\kappa)^2)$ *outputs a new ciphertext* ct' *such that* $\langle \mathsf{ct}', \mathsf{sk}\rangle = m + e(0) + e_{\mathsf{bt}} \pmod{p \cdot q}$*, where* $e(0)$ *is the constant coefficient of* e*, and* $\|e_{\mathsf{bt}}\| \leq B_{\mathsf{bt}}$ *with* $B_{\mathsf{bt}} := 10N$*.*

As opposed to the original CKKS bootstrapping, our new bootstrapping equation (8) does not require a small Hamming weight secret key; however, we describe an optimization based on such a secret key in Section 4.3.

4.2 First optimization: packing the secret-key bits

For a ring degree N, the above bootstrapping algorithm requires $\mathcal{O}(N)$ homomorphic operations, which would be impractical. Recall that the CKKS scheme provides a packing method where $N/2$ plaintext slots can be packed into a single ciphertext. These $N/2$ slots can then be added and multiplied independently, by performing homomorphic operations on ciphertexts (see Fig. 3 in Section 3.3). Therefore, the first natural optimization consists of packing the N secret key bits s_k from (8) into the $N/2$ slots of two CKKS ciphertexts, instead of encrypting each bit separately. Similarly, we pack the terms $\exp(2i\pi \cdot c_k/q) - 1$ into two polynomial encodings. The products $(\exp(2i\pi \cdot c_k/q) - 1) \cdot s_k$ are then computed independently and in parallel in the slots, thus requiring only two homomorphic external multiplications (ExtMultR). Moreover, the bootstrapping key size is also reduced from N to only 2 ciphertexts.

Another advantage of the packing method is that CKKS supports complex numbers in the slots natively, so we do not need to encrypt the real/imaginary parts separately anymore. To compute the product in (8), we can apply the $\mathsf{Pr}_{N/2\to 1}$ operator from Section 3.5 homomorphically to compute the product of the $N/2$ slots in each ciphertext. As a consequence, in the entire procedure, the multiplicative depth remains $\ell = \log_2(N)+2$, but now the number of homomorphic multiplications is reduced to $\mathcal{O}(\log N)$ instead $\mathcal{O}(N)$. We will describe this optimization more formally in Section 4.4, when combined with the following second optimization.

4.3 Second optimization: using the trace operator

In order to be competitive with the original CKKS bootstrapping, we describe a second optimization that decreases the multiplicative depth from $\mathcal{O}(\log N)$ to $\mathcal{O}(\log h)$, with $h \ll N$, by replacing most homomorphic multiplications by additions, which further improves efficiency.

For this, we still consider a binary secret key s but with a small Hamming weight h, where h is a power-of-two, moreover with a special block structure (as in e.g. [LMSS23]). We assume that the coefficient vector $\boldsymbol{s} \in \{0,1\}^N$ can be split into h blocks, each of size $B = N/h$, where each block contains a single 1. Formally, we write $\boldsymbol{s} = (s_{b\cdot B+j})_{0\leq b<h, 0\leq j<B}$ where in the b-th block, $s_{b\cdot B+j} = 0$ for all $j \in \{0,\ldots,B-1\} \setminus \{j_b^\star\}$, and $s_{b\cdot B+j_b^\star} = 1$ for some $0 \leq j_b^\star < B$.

As previously, we consider an LWE ciphertext $\boldsymbol{c} \in \mathbb{Z}_q^N$ derived from the original RLWE ciphertext, with decryption equation:

$$m \simeq \langle \boldsymbol{s}, \boldsymbol{c} \rangle = \sum_{k=0}^{N-1} s_k c_k = \sum_{b=0}^{h-1} \sum_{j=0}^{B-1} s_{b \cdot B+j} \cdot c_{b \cdot B+j} \pmod{q}$$

and as previously we embed the above equation into the multiplicative circle group of complex numbers:

$$\exp(2i\pi \cdot \langle \boldsymbol{s}, \boldsymbol{c} \rangle / q) = \prod_{b=0}^{h-1} \prod_{j=0}^{B-1} \exp(2i\pi \cdot s_{b \cdot B+j} \cdot c_{b \cdot B+j}/q).$$

Since within a given block there is only a single $s_{b \cdot B+j}$ equal to 1 and the others are zero, we can replace the products within a block by a sum:

$$\exp(2i\pi \cdot \langle \boldsymbol{s}, \boldsymbol{c} \rangle / q) = \prod_{b=0}^{h-1} \sum_{j=0}^{B-1} s_{b \cdot B+j} \cdot \exp(2i\pi \cdot c_{b \cdot B+j}/q).$$

As previously, this enables to derive the following bootstrapping equation:

$$\frac{m}{\Delta} \simeq \frac{q}{2\pi\Delta} \cdot \mathsf{Im} \left(\prod_{b=0}^{h-1} \sum_{j=0}^{B-1} s_{b \cdot B+j} \cdot \exp(2i\pi \cdot c_{b \cdot B+j}/q) \right)$$

We can remove the last multiplication by $q/(2\pi\Delta)$ by scaling by a factor $\delta \in \mathbb{R}$ such that $\delta^h = q/(2\pi\Delta)$. This yields our final bootstrapping equation in the single slot case:

$$\frac{m}{\Delta} \simeq \mathsf{Im} \left(\prod_{b=0}^{h-1} \sum_{j=0}^{B-1} s_{b \cdot B+j} \cdot \exp(2i\pi \cdot c_{b \cdot B+j}/q) \cdot \delta \right). \tag{9}$$

The advantage of the above equation is that it has a smaller multiplicative depth than (8), since we are computing the product of h factors only instead of N. More precisely, the multiplicative depth is now $\ell = \log_2(h) + 1$ instead of $\ell = \log_2(N) + 2$. By lowering the multiplicative depth, we can decrease the size of the largest modulus $Q = q \cdot p \cdot \Delta^\ell$. After bootstrapping, the refreshed ciphertext is encrypted under the larger modulus $p \cdot q$ rather than the initial modulus q, enabling a single additional homomorphic multiplication between two bootstrapped ciphertexts. Similar to the original CKKS bootstrapping, further homomorphic operations after bootstrapping can be supported by choosing a larger Q.

4.4 Our bootstrapping algorithm for a single component

We now formally describe our bootstrapping algorithm for a single slot by combining the two previous optimizations. We consider the sparse block secret key $\boldsymbol{s}$ as in Section 4.3, split into h blocks, each of size $B = N/h$. In order to combine the two optimizations, we must rewrite equation (9) to support packed slots. Recall that we must encode N secret-key bits, but a polynomial over the plaintext space $\mathbb{Z}[X]/(X^N + 1)$ can only encode $N/2$ slots, so we need two polynomial encodings. To speed up computation, we split each block of the secret key in two halves. The first halves of each block will be placed in the first polynomial encoding, and the second halves in the second encoding; see Figure 4 for an illustration.

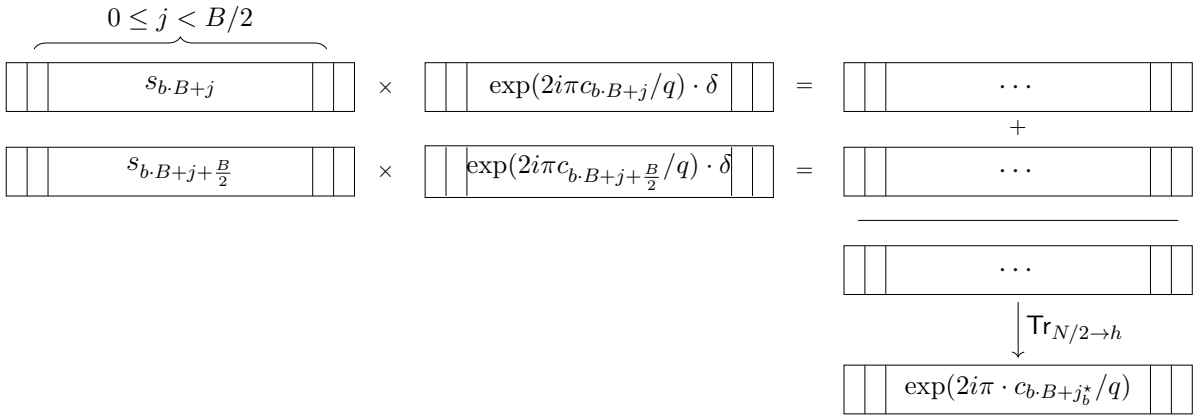


Fig. 4. Illustration of the computation within the half-blocks of size $B/2$. After the application of the $\mathsf{Tr}_{N/2\to h}$ operator, all $B/2$ slots contain the same value $\exp(2i\pi\cdot c_{b\cdot B+j_b^\star})$, where $j_b^\star$ is the index for which $s_{b\cdot B+j_b^\star}=1$. Note that each polynomial encoding contains h such half-blocks, for a total of $h\cdot B/2 = N/2$ slots.

After the external multiplications by the encoding of the $\exp(2i\pi\cdot c_{b\cdot B+j}/q)$ values, we compute the sum in (9) by first adding the two polynomial encodings, while the rest of the sum's evaluation within each block is handled by the $\mathsf{Tr}_{N/2\to h}$ operator applied to a single encoding.

However, we must be careful with the indexing of the slots when applying the $\mathsf{Tr}_{N/2\to h}$ operator. As illustrated in Appendix B.1, $\mathsf{Tr}_{N/2\to h}$ computes the partial sums of slots sharing the same index modulo h. This implies that we cannot encode the elements of the blocks contiguously in the slots, but only separated modulo h. Therefore, we utilize a "reversed" indexing, in which we put the j-th element of the b-th block at index $j\cdot h+b$ for all $0\le j<B/2$ for the first half, and similarly at index $(j-B/2)\cdot h+b$ for all $B/2\le j<B$ for the second half. More precisely, we denote by $\tilde{s}^0_\iota$ and $\tilde{s}^1_\iota$ the corresponding slots, for $0\le\iota<N/2$. According to the above indexing, we let $\tilde{s}^0_{j\cdot h+b}=s_{b\cdot B+j}$ for $0\le j<B/2$ (first block halves), and $\tilde{s}^1_{(j-B/2)\cdot h+b}=s_{b\cdot B+j}$ for $B/2<j<B$ (second block halves), for all $0\le b<h$. In total, the N secret-key bits s_k are encoded into the two polynomials:

$$S_0(X)=\mathsf{Ecd}((\tilde{s}^0_\iota)_{0\le\iota<N/2}),\qquad S_1(X)=\mathsf{Ecd}((\tilde{s}^1_\iota)_{0\le\iota<N/2}).$$

We encode the ciphertext's components following the same indexing. More precisely, we define the slots $(e^0_\iota)_{0\le\iota<N/2}$ and $(e^1_\iota)_{0\le\iota<N/2}$, with $e^0_{j\cdot h+b}=\exp(2i\pi\cdot c_{b\cdot B+j}/q)\cdot\delta$ for $0\le j<B/2$, and $e^1_{(j-B/2)\cdot h+b}=\exp(2i\pi\cdot c_{b\cdot B+j}/q)\cdot\delta$ for $B/2\le j<B$, for all $0\le b<h$. This yields the following polynomial encodings:

$$E_0(X)=\mathsf{Ecd}((e^0_\iota)_{0\le\iota<N/2}),\qquad E_1(X)=\mathsf{Ecd}((e^1_\iota)_{0\le\iota<N/2}).$$

Under this indexing, the bootstrapping equation (9) can be rewritten as:

$$\frac{m}{\Delta}\simeq\mathsf{Im}\left(\prod_{b=0}^{h-1}\sum_{j=0}^{B/2-1}\left(\tilde{s}^0_{j\cdot h+b}\cdot e^0_{j\cdot h+b}+\tilde{s}^1_{j\cdot h+b}\cdot e^1_{j\cdot h+b}\right)\right). \tag{10}$$

With this new indexing now compatible with the $\mathsf{Tr}_{N/2\to h}$ operator, we can perform the same operation homomorphically on polynomials. Specifically, we use $\mathsf{Tr}_{N/2\to h}$ to compute the sums

over each half-block and $\mathsf{Pr}_{h\to 1}$ to calculate the product of the h factors, after which the $N/2$ slots contain the same value. Finally, we extract the imaginary part using the $\mathsf{Im2}$ operator (see Section 3.4). Eventually, from (10), we obtain m/Δ in all $N/2$ slots, which corresponds to an encoding of $m = \mathsf{Ecd}(m/\Delta) \in \mathbb{Z}$. In total, we arrive at our bootstrapping equation for polynomial encodings:

$$m \simeq \mathsf{Im2}\left(\mathsf{Pr}_{h\to 1}\left(\mathsf{Tr}_{N/2\to h}\left(S_0(X) \times E_0(X) + S_1(X) \times E_1(X)\right)\right)\right). \tag{11}$$

Note that we must use a scaling factor $\delta \in \mathbb{R}$ such that $\delta^h = q/(4\pi\Delta)$ instead of $\delta^h = q/(2\pi\Delta)$, because of the factor 2 in the $\mathsf{Im2}$ operator.

Bootstrapping. As before, we claim that Equation (11) provides a decryption equation that is compatible with bootstrapping. Namely, all the previous polynomial operations can be applied homomorphically to ciphertexts.

More precisely, let cs denote the bootstrapping key consisting of CKKS encryptions $\mathsf{cs}_0, \mathsf{cs}_1$ of $S_0(X)$ and $S_1(X)$, modulo the largest modulus Q. The products $S_i(X) \times E_i(X)$ are then performed homomorphically on ciphertexts using $\mathsf{ExtMultR}(\mathsf{cs}_i, E_i)$, including rescaling. Similarly, the operators $\mathsf{Tr}_{N/2\to h}$, $\mathsf{Pr}_{h\to 1}$ and $\mathsf{Im2}$ are applied homomorphically on ciphertexts, as explained in sections 3.4 and 3.5. Eventually, we obtain an encryption of $m/\Delta \in \mathbb{R}$ over all slots, and equivalently an encryption of $m = \mathsf{Ecd}(m/\Delta) \in \mathbb{Z}$. This corresponds to a new CKKS ciphertext ct' such that $\langle \mathsf{ct}', \mathsf{sk}\rangle = m + e \pmod{pq}$, for a new modulus $pq > q$. For this, it suffices to set the largest modulus $Q := q \cdot p \cdot \Delta^\ell$, where $\ell = \log_2(h) + 1$ is the multiplicative depth. We provide a pseudo-code description of this bootstrapping algorithm in Appendix F.

The main advantage of the packing approach is that the total number of homomorphic operations is now $\mathcal{O}(\log N)$ instead of $\mathcal{O}(N)$, which makes our bootstrapping equation practical. We will consider the bootstrapping of $n \geq 2$ slots in parallel in Section 5, and concrete implementation results in Section 6.

4.5 Security analysis of block binary secret key

Our scheme uses a block binary secret key, divided into h blocks, each of size $B = N/h$, with exactly one '1' in each block. This distribution differs slightly from that used in the original CKKS bootstrapping, which employs a ternary secret key with Hamming weight h, but without the regular block structure. To ensure security against the best-known attacks, we provide a specific security analysis for this modified distribution and present corresponding parameter sets in Section 6. We follow the analysis from [LMSS23], which uses the same distribution in the context of TFHE.

We first consider the hybrid dual attack on LWE with a sparse secret [Alb17]. This attack, which combines a dual lattice attack with exhaustive search, is used in the Lattice Estimator [APS15] to estimate the security of LWE. In the first phase, the secret key $\boldsymbol{s}$ is partitioned as $\boldsymbol{s} = (\boldsymbol{s}_0, \boldsymbol{s}_1)$. Using lattice reduction, the LWE problem is reduced to a single LWE instance with a smaller dimension. The secret key $\boldsymbol{s}_0$ can then be recovered using combinatorial techniques such as exhaustive search or a meet-in-the-middle algorithm.

For the second, combinatorial phase, the authors of [LMSS23] considered two possible approaches. The original method involves guessing the positions of the zeros in the secret key. However, for the block binary distribution, the lattice dimension can be reduced by the block size $B = N/h$ by guessing the position of a '1' among B possible cases. While the latter approach seemed potentially more efficient, the authors provide an analysis showing that the original

approach of guessing the zero positions is, in fact, more efficient. Consequently, the block binary structure seems to offer no advantage for an attacker. Thus, the hybrid dual attack, as modeled in the Lattice Estimator, remains the best attack. Since, for convenience, we assume $s_0 = 1$, we must adjust the ring dimension to $N' = N \cdot (1 - 1/h)$ and the Hamming weight to $h' = h - 1$ in the estimator.

The authors of [LMSS23] also considered the recently improved meet-in-the-middle attack algorithm from May [May21]. It is based on recursively splitting the secret vector $\boldsymbol{s}$, which requires at least $\mathcal{S}^{0.25}$ time, where $\mathcal{S}$ is the size of the key space. In our case, the search space is $\mathcal{S} = (N/h)^{h-1}$. Therefore, the complexity of May's attack is at least $\mathcal{S} = (N/h)^{0.25(h-1)}$. As in [CHK+18a], we will take $h = 64$ in the concrete parameters. In that case, for $N \geq 2^{15}$, the complexity is at least 2^{141} operations.

Finally, we observe that the complexity of our bootstrapping algorithm grows only as $\mathcal{O}(\log h)$ homomorphic multiplications, making it relatively insensitive to increases in h. Even if the attacks exploiting the sparsity of the secret key were to improve significantly, increasing h would not result in a substantial performance penalty. For example, based on the operation count in Section 6, augmenting h from 64 to 128 would only increase the number of homomorphic operations in our bootstrapping by about 7%. Conversely, the complexity of May's attack would escalate from 2^{141} to 2^{254}. This originates from the double-exponential gap between the complexity of our bootstrapping and that of May's attack.

We conclude that, as in the TFHE case, the use of block binary secrets in CKKS does not affect the hardness of RLWE in the usual parameter setup.

5 Bootstrapping for multiple slots

In the previous section, we have described our new CKKS bootstrapping algorithm for a single slot only. That is, we considered a CKKS ciphertext ct such that $\langle \mathsf{ct}, \mathsf{sk} \rangle = m + e \pmod q$, for a message $m \in \mathbb{Z}$. In this section, we generalize our bootstrapping equation (9) to handle multiple slots in parallel.

5.1 Bootstrapping equation for multiple slots

For a power-of two $n \geq 2$, we consider the plaintext space $\mathcal{P}_n = \mathbb{Z}[X^{N/n}]/(X^N+1) \simeq \mathbb{Z}[Y]/(Y^n+1)$ for $Y = X^{N/n}$. Therefore, the plaintext space contains only n coefficients. We consider a ciphertext $\mathsf{ct} = (c_0, c_1)$ encrypting a message $m \in \mathcal{P}_n$.

Our first step is to extract the n corresponding LWE ciphertexts. For this, we consider the decryption equation $\langle \mathsf{ct}, \mathsf{sk} \rangle = c_0 + s \cdot c_1 = m + e \pmod q$, where

$$m(X) = \sum_{a=0}^{n-1} m_a X^{aN/n} \in \mathcal{P}_n,$$

and $e \in \mathbb{Z}[X]/(X^N+1)$. For ease of notation, in the decryption equation, we include the coefficients of $X^{aN/n}$ in the error $e(X)$ directly in $m(X)$. By the linearity of polynomial multiplication, we can rewrite the decryption equation as a vector-matrix multiplication, keeping only the coefficients in $X^{aN/n}$ for $0 \leq a < n$. By setting $s(X) = \sum_{i=0}^{N-1} s_i X^i$, the decryption equation can be written as:

$$\sum_{a=0}^{n-1} m_a X^{aN/n} + e = c_0 + c_1 \cdot \sum_{i=0}^{N-1} s_i X^i = c_0 + \sum_{i=0}^{N-1} s_i \cdot c_1 \cdot X^i \pmod q.$$

Therefore, we consider the vector $\boldsymbol{c}_0 \in \mathbb{Z}_q^n$ of coefficients of powers of $X^{N/n}$ of $c_0(X)$, and the matrix $\mathbf{C}_1 \in \mathbb{Z}_q^{N \times n}$ whose i-th row contains the n coefficients of powers of $X^{N/n}$ of $c_1(X) \cdot X^i$, for $0 \leq i < N$. Using this notation, we arrive at the equivalent decryption equation $\boldsymbol{m} = \boldsymbol{c_0} + \boldsymbol{s} \cdot \mathbf{C}_1 \pmod q$. Assuming that $s_0 = 1$, we can add the row vector $\boldsymbol{c_0}$ to the first row of $\mathbf{C}_1$ and obtain the decryption equation for $\boldsymbol{m} \in \mathbb{Z}^n$, $\boldsymbol{s} \in \mathbb{Z}^N$ and $\mathbf{C} \in \mathbb{Z}_q^{N \times n}$:

$$\boldsymbol{m} = \boldsymbol{s} \cdot \mathbf{C} \pmod q \tag{12}$$

We call $\mathbf{C}$ the *decryption matrix*, and by $\mathbf{C} \leftarrow \mathsf{DecMat}(\mathsf{ct})$ we denote the above algorithm extracting $\mathbf{C} \in \mathbb{Z}_q^{N \times n}$ from the ciphertext ct.

As in Section 4.3, we consider a binary secret key s with a small power-of-two Hamming weight h, such that when written as a coefficient vector $\boldsymbol{s} \in \{0,1\}^N$, it can be separated into h blocks, each of size $B = N/h$, where there is exactly a single 1 in each block. From (12), for each component m_a of $\boldsymbol{m} \in \mathbb{Z}^n$, we can write $m_a = \langle \boldsymbol{s}, \mathbf{C}_a \rangle \pmod q$ for the corresponding column vector $\mathbf{C}_a$ of $\mathbf{C}$. Therefore, each m_a is decrypted as an independent LWE ciphertext given by the column vector $\mathbf{C}_a$. This implies that Equation (9) generalizes to the following bootstrapping equation for each of the m_a for $0 \leq a < n$:

$$\frac{m_a}{\Delta} \simeq \mathsf{Im}\left(\prod_{b=0}^{h-1} \sum_{j=0}^{B-1} s_{b \cdot B + j} \cdot \exp(2i\pi \cdot C_{b \cdot B + j, a}/q) \cdot \delta\right). \tag{13}$$

In the next section, we show how to perform this computation efficiently in parallel using all available $N/2$ slots of polynomials in $\mathbb{Z}[X]/(X^N + 1)$, and then homomorphically over ciphertexts to achieve bootstrapping.

5.2 Bootstrapping algorithm

Our approach for evaluating (13) homomorphically is essentially the same as in Section 4.4. The difference is that we must compute (13) for the n components m_a in parallel instead of a single one. Since each polynomial encoding provides a maximum of $N/2$ slots, we only have $N/(2n)$ slots per component m_a at our disposal in each polynomial encoding. Recall that the binary secret key $\boldsymbol{s}$ is split into h blocks of size $B = N/h$, each containing a single 1. With only $N/(2n)$ slots at our disposal and h blocks, we will consider sub-blocks of size $N/(2n)/h = B/(2n)$. In other words, we split each block of B components into $2n$ sub-blocks with $B/(2n)$ components each, and we encode each of the $2n$ sub-blocks in a separate polynomial.

As illustrated in Figure 5, we first compute the products $s_{b \cdot B + j} \cdot \exp(2i\pi \cdot C_{b \cdot B + j, a}/q) \cdot \delta$ from (13) in parallel over each sub-block of size $B/(2n)$, for each of the $2n$ sub-blocks. To compute the sum in (13), we first compute the sum of the $2n$ corresponding encodings, such that we end up with a single sub-block with $B/(2n)$ components. Then we can apply the trace operator to finish the computation of the sum of each block. After the application of the trace operator, each of the $B/(2n)$ slots contains the same value $\exp(2i\pi \cdot C_{b \cdot B + j_b^\star, a}/q) \cdot \delta$. This computation is performed in parallel for each of the h blocks and for each of the n components m_a of the plaintext. In total, this corresponds to $h \cdot n$ independent slots; therefore, we must use the $\mathsf{Tr}_{N/2 \to hn}$ operator. Eventually we apply the $\mathsf{Pr}_{hn \to n}$ operator to compute the final product of the h elements, for each of the n components of the message in parallel.

As previously, we must use an indexing of the slots that is compatible with the $\mathsf{Tr}_{N/2 \to hn}$ operator, which computes partial sums of the slots sharing the same index modulo hn (see

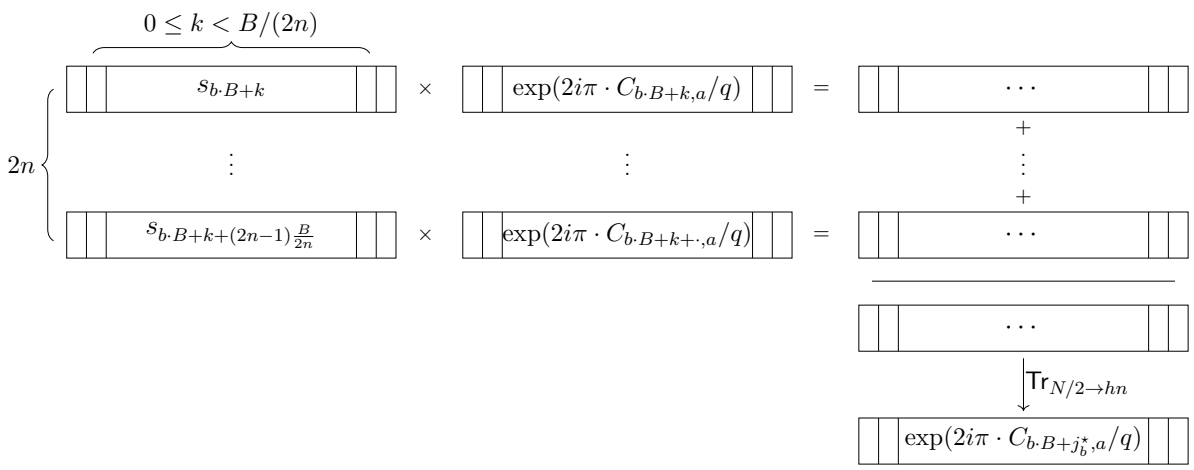


Fig. 5. Illustration of the computation within a given block. After application of the $\mathsf{Tr}_{N/2\to hn}$ operator, all $B/(2n)$ slots contain the same value $\exp(2i\pi \cdot C_{b\cdot B+j_b^\star,a}/q)$, where $j_b^\star$ is the index for which $s_{b\cdot B+j_b^\star} = 1$.

Appendix B.1 for an illustration). This implies that we must not encode the block's elements contiguously in the slots, but separated modulo nh. Therefore, we use a "reversed" indexing as in Section 4.4. In this reversed indexing, the index $0 \le j < B$ in (13) is decomposed as $j = u \cdot B/(2n) + k$ for $0 \le u < 2n$ and $0 \le k < B/(2n)$. Consequently, for an index $0 \le u < 2n$, we consider the slots $(\tilde{s}_\iota^u)_{0\le\iota<N/2}$, such that for all $0 \le k < B/(2n)$, $0 \le b < h$ and $0 \le a < n$, we have:

$$\tilde{s}^u_{k\cdot hn+b\cdot n+a} = s_{b\cdot B+u\cdot B/(2n)+k}.$$

We use the same reversed indexing for the ciphertext components, with the slots $(e_\iota^u)_{0\le\iota<N/2}$, such that for all $0 \le k < B/(2n)$, $0 \le b < h$ and $0 \le a < n$, we have:

$$e^u_{k\cdot hn+b\cdot n+a} = \exp(2i\pi \cdot C_{b\cdot B+u\cdot B/(2n)+k,a}/q) \cdot \delta.$$

Under this new indexing, we may rewrite the bootstrapping equation (13) as:

$$\frac{m_a}{\Delta} \simeq \mathsf{Im}\left(\prod_{b=0}^{h-1}\sum_{k=0}^{B/(2n)-1}\sum_{u=0}^{2n-1}\tilde{s}^u_{k\cdot hn+b\cdot n+a}\cdot e^u_{k\cdot hn+b\cdot n+a}\right)$$

Since the indexing is now compatible with the $\mathsf{Tr}_{N/2\to hn}$ operator, we can transfer the above equation to polynomial notation. Namely, we consider for an index $0 \le u < 2n$ the polynomials $S_u(X) = \mathsf{Ecd}((\tilde{s}_\iota^u)_{0\le\iota<N/2})$ and $E_u(X) = \mathsf{Ecd}((e_\iota^u)_{0\le\iota<N/2})$, and we obtain the equivalent equation:

$$\mathsf{Ecd}\left(\left(\frac{m_a}{\Delta}\right)_{0\le a<n}\right) \simeq \mathsf{Im2}\left(\mathsf{Pr}_{hn\to n}\left(\mathsf{Tr}_{N/2\to hn}\left(\sum_{u=0}^{2n-1} S_u(X)\times E_u(X)\right)\right)\right).$$

Note that in the above equation, we only have the decrypted values m_a/Δ in the slots, so we need to move the m_a's to the coefficient space, using the same $\mathsf{SlotToCoeff}$ procedure as in the

original CKKS bootstrapping procedure (see Section 3.6). This eventually gives us the decrypting equation:

$$\sum_{a=0}^{n-1} m_a X^{aN/n} \simeq \mathsf{StC}\left(\mathsf{Im2}\left(\mathsf{Pr}_{hn\to n}\left(\mathsf{Tr}_{N/2\to hn}\left(\sum_{u=0}^{2n-1} S_u(X)\times E_u(X)\right)\right)\right)\right).$$

Finally, since SlotToCoeff is based on a homomorphic DFT computation to achieve $\mathcal{O}(\log n)$ complexity, the coefficients m_a are, in fact, recovered in bit-reversed order. To recover the m_a coefficients with the normal order, we must therefore encode the slots in the polynomials $S_u(X)$ and $E_u(X)$ with bit-reversed order.

Bootstrapping. As previously, we claim that this equation is compatible with bootstrapping, since it can be homomorphically evaluated over ciphertexts, such that eventually, we obtain a refreshed ciphertext of the same plaintext $m(X) = \sum_{a=0}^{n-1} m_a X^{aN/n}$, but under a larger modulus $q \cdot p$.

As in the single slot case, the bootstrapping key consists of CKKS encryptions of the polynomials $S_u(X)$ for $0 \le u < 2n$; see Alg. 1 below. The products $S_u(X) \times E_u(X)$ are evaluated homomorphically on ciphertexts using the ExtMultR procedure.[1] Similarly, the operators $\mathsf{Tr}_{N/2\to hn}$, $\mathsf{Pr}_{hn\to 1}$ and Im2 are applied homomorphically on ciphertexts; see Alg. 2 below.

The multiplicative depth is now $\ell = \log_2 h + 1 + \ell_{\mathsf{StC}}$, where ℓ_{StC} is the depth of SlotToCoeff (see Appendix D). Therefore, we set the big modulus $Q = \Delta^\ell \cdot q \cdot p$. The total number of homomorphic operations is $\mathcal{O}(n + \log N)$.

Algorithm 1 Bootstrapping key generation, multiple slots

Input: A length N secret key $\boldsymbol{s}$ with Hamming weight h, and $B = N/h$
Output: A bootstrapping key $\mathsf{cs} = (\mathsf{cs}_u)_{0\le u<2n}$

for all $0 \le u < 2n$ **do**
  for all $0 \le k < B/(2n)$ **and** $0 \le b < h$ **and** $0 \le a < n$ **do**
    $\tilde{s}^u_{k\cdot hn+b\cdot n+a} = s_{b\cdot B+u\cdot B/(2n)+k}$
  $S_u(X) \leftarrow \mathsf{Ecd}((\tilde{s}^u_\iota)_{0\le\iota<N/2})$
  $\mathsf{cs}_u \leftarrow \mathsf{Enc}_{\mathsf{pk}}(S_u(X))$
return $(\mathsf{cs}_u)_{0\le u<2n}$

Algorithm 2 Bootstrapping, for at most $B/2$ slots

Input: A modulus q, a bootstrapping key $(\mathsf{cs}_u)_{0\le u<2n}$, an RLWE ciphertext ct containing $n \le B/2$ slots, where $B = N/h$, and $\delta = (q/(4\pi\Delta))^{1/h}$
Output: A refreshed ciphertext ct' modulo $p \cdot q$.

$\mathbf{C} \leftarrow \mathsf{DecMat}(\mathsf{ct})$
$\mathsf{acc} \leftarrow (0, 0)$
for $u = 0$ to $2n - 1$ **do**
  for all $0 \le k < B/(2n)$ **and** $0 \le b < h$ **and** $0 \le a < n$ **do**
    $e^u_{k\cdot hn+b\cdot n+a} = \exp(2i\pi \cdot \mathbf{C}_{b\cdot B+u\cdot B/(2n)+k,a}/q) \cdot \delta$
  $E_u(X) \leftarrow \mathsf{Ecd}((e^u_\iota)_{0\le\iota<N/2})$
  $T_u \leftarrow \mathsf{ExtMultR}(\mathsf{cs}_u, E_u)$
  $\mathsf{acc} \leftarrow \mathsf{Add}(\mathsf{acc}, T_u)$
return $\mathsf{SlotToCoeff}(\mathsf{Im2}(\mathsf{Pr}_{hn\to n}(\mathsf{Tr}_{N/2\to hn}(\mathsf{acc}))))$

[1] As an optimization, we may compute the rescaling operation inside the ExtMultR procedure only after computing the sum over $0 \le u < 2n$. This way, we only rescale once compared to $2n$ rescalings.

Bootstrapping up to N slots. The previous bootstrapping algorithm is limited to bootstrapping up to $n \leq n_{\mathsf{max}} = B/2 = N/(2h)$ components. However, it can be easily extended to support more slots, specifically $n' \leq N$, by shifting the coefficients of the input ciphertext and applying the previous bootstrapping procedure as a black box for each group of n_{max} coefficients. We refer to Appendix G for the details. The number of homomorphic operations remains $\mathcal{O}(n + \log N)$ for n slots, while the depth remains unchanged. Note that our bootstrapping method is highly parallelizable; with n processors, it achieves the same $\mathcal{O}(\log N)$ complexity as the original CKKS bootstrapping.

6 Implementation and performance comparison

In this section, we implement both the original CKKS bootstrapping and our new bootstrapping, and we compare the performances of the two algorithms.

6.1 Fixing the parameters

We use the standard Lattice Estimator [APS15] to fix the parameters for security against the best known attacks, including the hybrid lattice attack taking advantage of the sparse key distribution (see Section 4.5).

In Table 1 we summarize the maximal size of the largest modulus $Q = q_L$ in the ladder, as a function of $\log_2 N$, for $\lambda = 100$ bits of security. We account for the fact that the switching keys $\mathsf{evk}, \mathsf{rk}_r$, and ck are all encrypted modulo $P \cdot Q$, with $P = Q$. Consequently, the effective largest modulus is Q^2. For both bootstrapping algorithms and as in [CHK+18a], we take a fixed Hamming weight $h = 64$. Moreover, for both bootstrapping, we leave a single multiplicative level after bootstrapping, but this can be easily adjusted as needed. More precisely, we start with a ciphertext modulo the smallest modulus q, and after bootstrapping we obtain a refreshed ciphertext modulo $p \cdot q$.

$\log_2 N$	12	13	14	15	16	17
Maximal $\log_2 Q$	60	122	247	501	1012	2045

Table 1. Maximal modulus size $\log_2 Q$, as a function of $\log_2 N$, for $\lambda = 100$ bits of security. The effective modulus is Q^2.

CKKS bootstrapping parameters. To fix the parameters for the original CKKS bootstrapping, we follow a similar approach as in [CHK+18a]. For a bit size $n_q = \log_2 q$, since we must have $\|m\|_\infty \leq q^{2/3}$, we use $n_p = \lfloor n_q \cdot 2/3 \rceil$ bits of precision for the coefficients of m, and we let $p = 2^{n_p}$. We use q as the lowest modulus and $Q = q \cdot p \cdot \Delta^\ell$ as the largest modulus, where $\ell = d + r + 2\ell_{\mathsf{StC}}$ represents the bootstrapping depth. For CoeffToSlot and SlotToCoeff, we use a homomorphic DFT evaluation with radix $r = 2^2$, which has depth $\ell_{\mathsf{CtS}} = \ell_{\mathsf{StC}} = 1 + \lfloor (\log_2 n)/2 \rfloor$, except for $n = 1$ where $\ell_{\mathsf{CtS}} = \ell_{\mathsf{StC}} = 0$. We note that a smaller depth can be obtained by increasing the radix, at the cost of more homomorphic operations; for example, [CHH18] uses a depth of 3 or 4. In Appendix D.8, we analyze the impact of increasing the radix on the running time.

After bootstrapping, the refreshed ciphertext is defined modulo $q \cdot p$. This means that during bootstrapping, we use Δ as the rescaling factor, and eventually, for a single homomorphic

multiplication of refreshed ciphertexts, we use a smaller p as the rescaling factor. We fix $\Delta = q \cdot 2^{d+r}$. We provide the corresponding parameters in Table 2, grouped into two sets, for 11 bits and 24 bits of precision respectively, each varying over a number of slots $n \leq 2^{10}$.

Parameter	$\log_2 N$	$\log_2 p$	$\log_2 q$	$\log_2 \Delta$	d	r	n	ℓ	$\log_2 Q$	precision
Set-I	16	18	27	43	7	9	1	16	733	11 bits
							2^{10}	28	1249	
Set-II	17	31	46	63	7	10	1	17	1148	24 bits
							2^{10}	29	1904	

Table 2. Parameter sets for the original CKKS bootstrapping, for 100 bits of security.

New bootstrapping parameters. We follow a similar approach as for the original CKKS bootstrapping. We also use $n_p = \lfloor n_q \cdot 2/3 \rceil$ bits of precision for the coefficients of m, and we let $p = 2^{n_p}$. We also use q as the lowest modulus, and $Q = q \cdot p \cdot \Delta^\ell$ as the largest modulus, where $\ell = \log_2 h + 1 + \ell_{\mathsf{StC}}$ is the depth of our bootstrapping. We fix $\Delta = q \cdot 2^{10}$. Again, the refreshed ciphertext after bootstrapping is defined modulo $q \cdot p$. For comparability, in Table 3, we provide two similar sets of parameters, taking again 11 bits and 24 bits of precision, for a maximum of $n = 64$ slots. Note that we can bootstrap more slots as explained in Appendix G.

Parameter	$\log_2 N$	$\log_2 p$	$\log_2 q$	$\log_2 \Delta$	n	ℓ	$\log_2 Q$	precision
Set-I	15	17	25	35	1	7	287	11 bits
					64	11	427	
Set-II	16	31	46	56	1	7	469	24 bits
					64	11	693	

Table 3. Parameter sets for our new bootstrapping algorithm, for 100 bits of security.

By comparing tables 2 and 3, we observe that due to its lower multiplicative depth ℓ, our new bootstrapping can use a smaller modulus Q and therefore a smaller ring dimension N.

6.2 Counting operations

We first theoretically compare the number of homomorphic operations required in both bootstrappings. We obtain the following costs, expressed in the equivalent number of ciphertext multiplications (see Appendix H for the details):

$$T_{\mathsf{CKKS}} = \frac{1}{2}\log_2 N + d + r + \frac{13}{4}\log_2 n - 2a$$
$$T_{\text{new}} = \frac{1}{2}\log_2(N/2) + \log_2 h + \frac{n+1}{2} + \frac{11}{8}\log_2 n - a$$

where $a = 7/8$ for even $\log_2 n$, and $a = 1$ for odd $\log_2 n$. The main difference is that for n slots, the original CKKS bootstrapping scales as $\mathcal{O}(\log n)$, whereas our new bootstrapping scales as $\mathcal{O}(n)$ only. We summarize the asymptotic complexities of CKKS bootstrapping approaches in Table 4, expressed in the number of homomorphic operations.

In Table 5, we provide a comparison of the original CKKS bootstrapping and our new bootstrapping for the Set-I parameters. Thanks to its smaller depth, our new bootstrapping

	Complexity
Original CKKS bootstrapping [CHK+18a]	$\mathcal{O}(\log n + \log N)$
Blind rotation bootstrapping [KDE+21]	$\mathcal{O}(n \cdot N)$
Our new bootstrapping	$\mathcal{O}(n + \log N)$

Table 4. Complexities of different bootstrapping approaches for CKKS, expressed in number of ciphertext multiplications, for a ring dimension N and n slots.

can profit from a smaller ring dimension and a smaller modulus Q. To account for the smaller $N = 2^{15}$ (instead of $N = 2^{16}$), we halve the number of equivalent ciphertext multiplications for our bootstrapping when calculating the number of operations in Table 5, to reflect the quasi-linear complexity of polynomial multiplication. Note that in Table 5, we do not account for the smaller modulus size for Q in our bootstrapping, even though this factor will further improve the concrete running time in our favor (see Table 6).

n	1	2	4	8	16	32	64	128	256	512	1024
CKKS	23	26	29	32	36	39	42	45	49	52	55
New bootstrapping	7	8	9	11	14	18	27	54	108	216	432

Table 5. Number of ciphertext multiplications (rounded up to the next integer) for Set-I parameters, with $N = 2^{16}$, $d = 7$, $r = 9$ for CKKS, and $N = 2^{15}$, $h = 64$ and $n_{\mathsf{max}} = 64$ for our bootstrapping.

We observe that for few slots ($n \leq 16$), our bootstrapping is significantly more efficient. Additionally, it scales relatively well for small values of n; for instance, the number of operations for $n = 16$ is only double that for $n = 1$. The crossover point occurs at $n = 128$ slots, beyond which our approach scales linearly with n, whereas CKKS grows more slowly as $\mathcal{O}(\log n)$.

6.3 Running time comparison

We present an implementation on an Intel Core i7 (2.6 GHz) using the parameters from Tables 2 and 3, which ensure 100-bit security. Our implementation is built on the SageMath library in Python. While most FHE libraries are written in C++ for efficiency, we found the SageMath/Python framework to be more flexible for a proof-of-concept implementation, while still offering reasonable performance. The computational cost is primarily dominated by polynomial multiplications, which are handled by the NTL C++ library within SageMath.

However, our implementation does not leverage RNS arithmetic, which was introduced for CKKS in [CHK+18b] and can provide an order-of-magnitude improvement in efficiency. This optimization is fully compatible with our new bootstrapping algorithm, and we would expect the same performance ratio between the original CKKS bootstrapping and our new bootstrapping.

The timing results are presented in Table 6. We see that due to its smaller multiplicative depth, our bootstrapping is between 7 and 8 times faster than CKKS bootstrapping for a single slot ($n = 1$). As illustrated in Figure 6, the crossover point lies around $n = 128$ slots for the Set-I parameters, and around $n = 256$ for the Set-II parameters, after which the original CKKS bootstrapping scales as $\mathcal{O}(\log n)$, whereas our new bootstrapping scales as $\mathcal{O}(n)$, thus it becomes unpractical for $n > 1024$.

Number of slots		1	2	4	8	16	32	64	128	256	512	1024
Set-I	CKKS boot.	29 s	35 s	43 s	55 s	68 s	81 s	95 s	110 s	129 s	147 s	160 s
	New boot.	4 s	5 s	7 s	11 s	16 s	26 s	45 s	93 s	189 s	377 s	752 s
Set-II	CKKS boot.	113 s	131 s	163 s	205 s	236 s	280 s	338 s	382 s	436 s	492 s	565 s
	New boot.	14 s	17 s	22 s	31 s	43 s	68 s	113 s	225 s	452 s	909 s	1814 s

Table 6. Running time comparison of the original CKKS bootstrapping and our new bootstrapping, for a variable number of slots n.

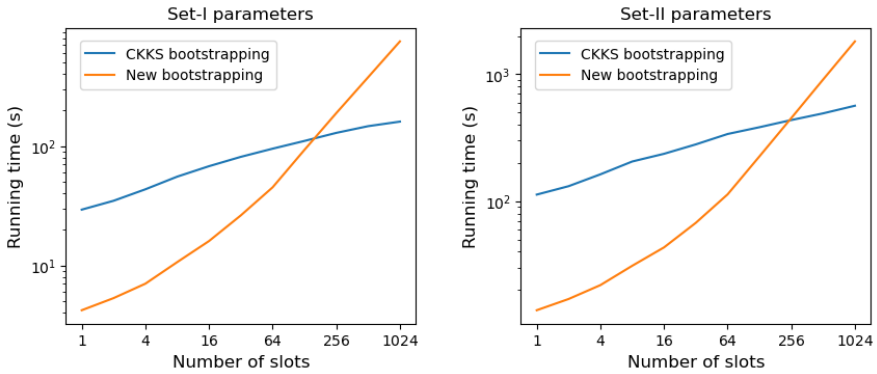


Fig. 6. Comparison of the running times of CKKS bootstrapping and our new bootstrapping, for Set-I parameters (left), and for Set-II parameters (right).

7 Conclusion

In this paper, we presented an alternative bootstrapping algorithm for the CKKS scheme, based on embedding the additive group modulo q into the circle group of complex numbers, which can be evaluated natively in CKKS. Due to its lower multiplicative depth, our new bootstrapping algorithm can operate with a smaller ring dimension N, resulting in significant efficiency improvements for few slots n. However, unlike the original CKKS bootstrapping, which scales as $\mathcal{O}(\log n)$, our method scales as $\mathcal{O}(n)$, making it less efficient for large n.

This new bootstrapping approach could be particularly useful in scenarios where circuit evaluation requires a dynamic number of slots, offering an alternative to scheme-switching methods. Given that bootstrapping is a fundamental algorithm, we believe it is valuable to explore new techniques that offer different mathematical foundations and performance trade-offs.

References

Alb17. Martin R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 103–129, 2017.

AP14. Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 297–314. Springer, 2014.

APS15. Martin Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9, 10 2015.

BCC+22. Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. META-BTS: Bootstrapping precision beyond the limit. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 223–234, New York, NY, USA, 2022. Association for Computing Machinery.

BCKS24. Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, and Damien Stehlé. Bootstrapping bits with CKKS. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part II*, volume 14652 of *Lecture Notes in Computer Science*, pages 94–123. Springer, 2024.

BGV11. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electron. Colloquium Comput. Complex.*, page 111, 2011.

Bra12. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.

CCKK24. Jung Hee Cheon, Hyeongmin Choe, Minsik Kang, and Jaehyung Kim. Grafting: Complementing RNS in CKKS. Cryptology ePrint Archive, Paper 2024/1014, 2024.

CCKS23. Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Damien Stehlé. Homomorphic multiple precision multiplication for CKKS and reduced modulus consumption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 696–710, New York, NY, USA, 2023. Association for Computing Machinery.

CCS19. Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In Yuval Ishai and Vincent Rijmen, editors, *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*, volume 11477 of *Lecture Notes in Computer Science*, pages 34–54. Springer, 2019. https://eprint.iacr.org/2018/1043.pdf.

CDKS20. Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient homomorphic conversion between (ring) LWE ciphertexts. Cryptology ePrint Archive, Paper 2020/015, 2020. `https://eprint.iacr.org/2020/015`.

CGGI16. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33, 2016.

CHH18. Jung Hee Cheon, Kyoohyung Han, and Minki Hhan. Faster homomorphic discrete fourier transforms and improved FHE bootstrapping. Cryptology ePrint Archive, Paper 2018/1073, 2018.

CHK+18a. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *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*, volume 10820 of *Lecture Notes in Computer Science*, pages 360–384. Springer, 2018.

CHK+18b. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 347–368. Springer, 2018.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *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*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017.

DM15. Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640. Springer, 2015.

DMPS24. Nir Drucker, Guy Moshkowich, Tomer Pelleg, and Hayim Shaul. BLEACH: cleaning errors in discrete computations over CKKS. *J. Cryptol.*, 37(1):3, 2024.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. volume 9, pages 169–178, 05 2009.

GHS12a. Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead, 2012.

GHS12b. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.

GSW13. Craig Gentry, Amit Sahai, and Brent Waters. 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*, pages 75–92, 2013.

JKA+21. Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Cheon, and Younho Lee. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021:114–148, 08 2021.

JM22. Charanjit S. Jutla and Nathan Manohar. Sine series approximation of the mod function for bootstrapping of approximate he. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 491–520, Cham, 2022. Springer International Publishing.

KDE+21. Andrey Kim, Maxim Deryabin, Jieun Eom, Rakyong Choi, Yongwoo Lee, Whan Ghang, and Donghoon Yoo. General bootstrapping approach for RLWE-based homomorphic encryption. Cryptology ePrint Archive, Paper 2021/691, 2021. `https://eprint.iacr.org/2021/691`.

KPK+23. Seonghak Kim, Minji Park, Jaehyung Kim, Taekyung Kim, and Chohong Min. 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*, page 161–187, Berlin, Heidelberg, 2023. Springer-Verlag.

KPP22. Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. Approximate homomorphic encryption with reduced approximation error. In *Topics in Cryptology – CT-RSA 2022: Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1–2, 2022, Proceedings*, page 120–144, Berlin, Heidelberg, 2022. Springer-Verlag.

LLK+22. Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and HyungChul Kang. High-precision bootstrapping for approximate homomorphic encryption by error variance minimization. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 551–580, Cham, 2022. Springer International Publishing.

LLL+21. Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. 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*, page 618–647, Berlin, Heidelberg, 2021. Springer-Verlag.

LMSS23. Changmin Lee, Seonhong Min, Jinyeong Seo, and Yongsoo Song. Faster TFHE bootstrapping with block binary keys. In Joseph K. Liu, Yang Xiang, Surya Nepal, and Gene Tsudik, editors, *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security, ASIA CCS 2023, Melbourne, VIC, Australia, July 10-14, 2023*, pages 2–13. ACM, 2023.

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.

May21. Alexander May. How to meet ternary LWE keys. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 701–731. Springer, 2021.

SSTX09. Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 617–635. Springer, 2009.

SV14. N. P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. *Designs, Codes and Cryptography*, 71(1):57–81, Apr 2014.

SYL+23. S. Shen, H. Yang, Y. Liu, Z. Liu, and Y. Zhao. CARM: CUDA-accelerated RNS multiplication in word-wise homomorphic encryption schemes for internet of things. *IEEE Transactions on Computers*, 72(07):1999–2010, jul 2023.

A Proof of lemmas 1, 2, 3, 4 and 5

For the following proofs, we will use two facts. For real numbers $r_1, r_2 \in \mathbb{R}$, it holds that $\|\lfloor r_1 \pm r_2 \rceil\|_\infty \leq \|\lfloor r_1 \rceil \pm \lfloor r_2 \rceil\|_\infty + 1$. Moreover, for $r \in \mathbb{R}[X]/(X^N+1)$ and $s \in \mathbb{Z}[X]/(X^N+1)$ with hamming weight h and $\|s\|_\infty = 1$, we have that $\|s \cdot \lfloor r(X) \rceil - \lfloor s \cdot r(X) \rceil\|_\infty \leq h \leq N$, which can be seen by applying the real number case to polynomial multiplication.

A.1 Proof of Lemma 1

Let $\mathsf{pk} = (-a's + e', a')$ be the public key and ct be the ciphertext, i.e. $\mathsf{ct} = \mathsf{Enc}_{\mathsf{pk}}(m) = v \cdot \mathsf{pk} + (m + e_0, e_1) = ((-a's + e')v + m + e_0, a'v + e_1)$, which gives

$$\begin{aligned}\langle \mathsf{ct}, \mathsf{sk} \rangle &= (-a's + e')v + m + e_0 + s(a'v + e_1) \pmod{q_L} \\ &= m + e_0 + e'v + se_1 \pmod{q_L}.\end{aligned}$$

Writing $\langle \mathsf{ct}, \mathsf{sk} \rangle = m + e_{\mathsf{clean}} \pmod{q_L}$, we get for a signed binary secret key:

$$\|e_{\mathsf{clean}}\|_\infty \leq \|e_0\|_\infty + \|e' \cdot v\|_\infty + \|s \cdot e_1\|_\infty \leq \kappa + N \cdot \kappa \cdot 1 + N \cdot 1 \cdot \kappa \leq 3N\kappa.$$

A.2 Proof of Lemma 2

Let $\gamma = q_\ell / q_{\ell'}$. We write $\mathsf{ct} = (b, a)$, which gives $\mathsf{ct}' = (\lfloor b/\gamma \rceil, \lfloor a/\gamma \rceil)$. We have $\langle \mathsf{ct}, \mathsf{sk} \rangle = b + s \cdot a + \xi \cdot q_\ell$ for some $\xi \in \mathbb{Z}$, which gives:

$$\langle \mathsf{ct}, \mathsf{sk} \rangle / \gamma = b/\gamma + s \cdot a/\gamma + \xi \cdot q_{\ell'},$$

and therefore we obtain:

$$\begin{aligned}\lfloor \langle \mathsf{ct}, \mathsf{sk} \rangle / \gamma \rceil &= \lfloor b/\gamma \rceil + s \cdot \lfloor a/\gamma \rceil - e_{\mathsf{rs}} \pmod{q_{\ell'}} \\ &= \langle \mathsf{ct}', \mathsf{sk} \rangle - e_{\mathsf{rs}} \pmod{q_{\ell'}},\end{aligned}$$

where $e_{\mathsf{rs}} = \lfloor b/\gamma \rceil - b/\gamma + s \cdot (\lfloor a/\gamma \rceil - a/\gamma) + \langle \mathsf{ct}, \mathsf{sk} \rangle / \gamma - \lfloor \langle \mathsf{ct}, \mathsf{sk} \rangle / \gamma \rceil$. This implies $\|e_{\mathsf{rs}}\|_\infty \leq \|s\|_1 + 1 \leq 2N$, with $\langle \mathsf{ct}', \mathsf{sk} \rangle = \lfloor \langle \mathsf{ct}, \mathsf{sk} \rangle \cdot q_{\ell'} / q_\ell \rceil + e_{\mathsf{rs}} \pmod{q_{\ell'}}$ as required.

A.3 Proof of Lemma 3

We use as an input an evaluation key $\mathsf{evk} = (b', a') = (-a's + e' + Ps^2, a')$ with $\|e'\|_\infty \leq \kappa$ and ciphertexts ct_i satisfying $\langle \mathsf{ct}_i, \mathsf{sk} \rangle = b_i + s \cdot a_i \pmod{q_\ell}$ for $i \in \{1, 2\}$. We have:

$$\langle \mathsf{ct}_1, \mathsf{sk} \rangle \cdot \langle \mathsf{ct}_2, \mathsf{sk} \rangle = b_1 b_2 + s(b_1 a_2 + b_2 a_1) + s^2 a_1 a_2 \pmod{q_\ell}.$$

Letting $(d_0, d_1, d_2) := (b_1 b_2, b_1 a_2 + b_2 a_1, a_1 a_2)$, we rewrite the above as:

$$\langle \mathsf{ct}_1, \mathsf{sk} \rangle \cdot \langle \mathsf{ct}_2, \mathsf{sk} \rangle = d_0 + s d_1 + s^2 d_2 \pmod{q_\ell}.$$

We consider d_2 over $\mathbb{Z}[X]/(X^N+1)$, with $\|d_2\|_\infty \leq q_\ell$. By definition, $\mathsf{ct}_{\mathsf{mult}} = (d_0, d_1) + \lfloor P^{-1} d_2 \cdot \mathsf{evk} \rceil$, where we evaluate $d_2 \cdot \mathsf{evk}$ modulo Pq_L and then divide the latter by P over the reals. This implies, using $q_\ell | q_L$ and some $\gamma_1, \gamma_2 \in \mathbb{Z}$:

$$\begin{aligned} d_2 \cdot \mathsf{evk} &= d_2 \cdot (-a's + e' + Ps^2, a') + Pq_L(\gamma_1, \gamma_2) \qquad \text{over } \mathbb{Z}, \text{ and} \\ \lfloor P^{-1} d_2 \cdot \mathsf{evk} \rceil &= \left(\lfloor P^{-1} d_2 (-a's + e') \rceil + d_2 s^2, \lfloor P^{-1} a' d_2 \rceil \right) \pmod{q_\ell}. \end{aligned}$$

The above now gives:

$$\begin{aligned} \langle \mathsf{ct}_{\mathsf{mult}}, \mathsf{sk} \rangle &= d_0 + d_1 s + \lfloor P^{-1} d_2 (-a's + e') \rceil + d_2 s^2 + s \lfloor P^{-1} a' d_2 \rceil \pmod{q_\ell} \\ &= \langle \mathsf{ct}_1, \mathsf{sk} \rangle \cdot \langle \mathsf{ct}_2, \mathsf{sk} \rangle + e_{\mathsf{mult}} \pmod{q_\ell}, \end{aligned}$$

and now we can estimate the size of e_{mult} as follows:

$$\begin{aligned} \|e_{\mathsf{mult}}\|_\infty &= \left\| \lfloor P^{-1} d_2 (-a's + e') \rceil + s \lfloor P^{-1} d_2 a' \rceil \right\|_\infty \\ &\leq \left\| \lfloor P^{-1} d_2 (-a's + e') \rceil + \lfloor s \cdot P^{-1} d_2 a' \rceil \right\|_\infty + \|s\|_1 \\ &\leq \left\| - \lfloor s P^{-1} d_2 a' \rceil + \lfloor P^{-1} d_2 e' \rceil + \lfloor s P^{-1} d_2 a' \rceil \right\|_\infty + \|s\|_1 + 1 \\ &\leq P^{-1} \cdot N q_\ell \kappa + N + 2. \end{aligned}$$

Finally, if $P \geq N q_\ell \kappa$, then $\|e_{\mathsf{mult}}\|_\infty \leq 2N$.

A.4 Proof of Lemma 4

Letting $\mathsf{ct}_{\mathsf{mult}} \leftarrow \mathsf{Mult}_{\mathsf{pk}}(\mathsf{ct}_1, \mathsf{ct}_2)$, we obtain by applying Lemma 3:

$$\begin{aligned} \langle \mathsf{ct}_{\mathsf{mult}}, \mathsf{sk} \rangle &= \langle \mathsf{ct}_1, \mathsf{sk} \rangle \cdot \langle \mathsf{ct}_2, \mathsf{sk} \rangle + e_{\mathsf{mult}} \pmod{q_\ell} \\ &= (\mathsf{Ecd}(x_1) + e_1)(\mathsf{Ecd}(x_2) + e_2) + e_{\mathsf{mult}} \pmod{q_\ell} \\ &= \mathsf{Ecd}(x_1)\mathsf{Ecd}(x_2) + e_m + e_{\mathsf{mult}} \pmod{q_\ell}, \end{aligned}$$

for $e_m = e_1 e_2 + \mathsf{Ecd}(x_1) e_2 + \mathsf{Ecd}(x_2) e_1$, and $\|e_{\mathsf{mult}}\|_\infty \leq 2N$. Letting $\mathsf{ct}_{\mathsf{multR}} \leftarrow \mathsf{RS}(\mathsf{ct}_{\mathsf{mult}})$, applying Lemma 2, and using $q_\ell / q_{\ell-1} = \Delta$, we get:

$$\begin{aligned} \langle \mathsf{ct}_{\mathsf{multR}}, \mathsf{sk} \rangle &= \lfloor \langle \mathsf{ct}_{\mathsf{mult}}, \mathsf{sk} \rangle / \Delta \rceil + e_{\mathsf{rs}} \pmod{q_{\ell-1}} \\ &= \lfloor (\mathsf{Ecd}(x_1)\mathsf{Ecd}(x_2) + e_m + e_{\mathsf{mult}}) / \Delta \rceil + e_{\mathsf{rs}} \pmod{q_{\ell-1}}, \end{aligned}$$

where $\|e_{\mathsf{rs}}\|_\infty \leq 2N$. By writing $\mathsf{Ecd}(x_1)\mathsf{Ecd}(x_2)/\Delta = \mathsf{Ecd}(x_1 x_2) + e_{\mathsf{ecd}}$, we now reformulate the above as:

$$\begin{aligned} \langle \mathsf{ct}_{\mathsf{multR}}, \mathsf{sk} \rangle &= \mathsf{Ecd}(x_1 x_2) + \lfloor e_{\mathsf{ecd}} + (e_m + e_{\mathsf{mult}})/\Delta \rceil + e_{\mathsf{rs}} \pmod{q_{\ell-1}} \\ &= \mathsf{Ecd}(x_1 x_2) + e_{\mathsf{multR}} \pmod{q_{\ell-1}} \end{aligned}$$

Therefore, we have:

$$\|e_{\mathsf{multR}}\|_\infty \leq |e_{\mathsf{ecd}}| + \frac{\|e_m\|_\infty}{\Delta} + \frac{\|e_{\mathsf{mult}}\|_\infty}{\Delta} + \|e_{\mathsf{rs}}\|_\infty + 1.$$

We first bound the error e_{ecd}:

$$\begin{aligned} e_{\mathsf{ecd}} &= \frac{1}{\Delta} \cdot (\mathsf{Ecd}(x_1)\mathsf{Ecd}(x_2) - \Delta \cdot \mathsf{Ecd}(x_1 x_2)) = \frac{1}{\Delta} \cdot (\lfloor \Delta x_1 \rceil \lfloor \Delta x_2 \rceil - \Delta \lfloor \Delta x_1 x_2 \rceil) \\ &= \frac{1}{\Delta} \cdot ((\Delta x_1 + \varepsilon_1)(\Delta x_2 + \varepsilon_2) - \Delta(\Delta x_1 x_2 + \varepsilon_{12})) \\ &= \frac{1}{\Delta} (\Delta x_1 \varepsilon_2 + \Delta x_2 \varepsilon_1 + \varepsilon_1 \varepsilon_2 - \Delta \varepsilon_{12}) \end{aligned}$$

for some $|\varepsilon_1|, |\varepsilon_2|, |\varepsilon_{12}| \leq 1/2$. Using $|x_i| \leq \nu$, this implies

$$|e_{\mathsf{ecd}}| \leq \nu + 1.$$

Recall that $e_m = e_1 e_2 + \mathsf{Ecd}(x_1) e_2 + \mathsf{Ecd}(x_2) e_1$. Using $\mathsf{Ecd}(x_i) \in \mathbb{Z}$ and $|\mathsf{Ecd}(x_i)| = |\lfloor x_i \cdot \Delta \rceil| \leq \Delta\nu + 1$, this gives:

$$\|e_m\|_\infty \leq NE^2 + 2E \cdot (\Delta\nu + 1) \leq 2NE^2 + 2\nu E\Delta.$$

Now, for e_{multR}, we have:

$$\|e_{\mathsf{multR}}\|_\infty \leq \nu + 1 + \frac{2NE^2}{\Delta} + 2\nu E + \frac{2N}{\Delta} + 2N + 1.$$

Assuming $E^2 \leq \Delta$ and $\nu < N$, we eventually get:

$$\|e_{\mathsf{multR}}\|_\infty \leq 2\nu E + 8N.$$

A.5 Proof of Lemma 5

The proof is similar to the proof of Lemma 3. We input the key-switching key $\mathsf{swk} = (b', a')$, for which $b' = -a's + e' + Ps' \pmod{Pq_L}$. Let $\mathsf{ct} = (b, a)$ be the input ciphertext under $\mathsf{sk}' = (1, s')$. We consider a over $\mathbb{Z}[X]/(X^N + 1)$ with $\|a\|_\infty \leq q_\ell$. The output ciphertext is defined as $\mathsf{ct}' = (b, 0) + \lfloor P^{-1} a \cdot \mathsf{swk} \rceil \pmod{q_\ell}$, where we evaluate $a \cdot \mathsf{swk}$ modulo Pq_L and then divide by P over the reals. As in the proof of Lemma 3, this gives:

$$\lfloor P^{-1} a \cdot \mathsf{swk} \rceil = \left(\lfloor P^{-1} a(-a's + e') \rceil + as', \lfloor P^{-1} a' a \rceil \right) \pmod{q_\ell}.$$

The decryption equation of ct' now indeed satisfies:

$$\begin{aligned} \langle \mathsf{ct}', \mathsf{sk} \rangle &= b + \lfloor P^{-1} a(-a's + e') \rceil + as' + s \cdot \lfloor P^{-1} a' a \rceil \pmod{q_\ell} \\ &= \langle \mathsf{ct}, \mathsf{sk}' \rangle + \lfloor P^{-1} a(-a's + e') \rceil + s \cdot \lfloor P^{-1} a' a \rceil \pmod{q_\ell} \\ &= \langle \mathsf{ct}, \mathsf{sk}' \rangle + e_{\mathsf{ks}} \pmod{q_\ell}. \end{aligned}$$

Finally, we estimate the size of e_{ks} as follows:

$$\begin{aligned} \|e_{\mathsf{ks}}\|_\infty &= \left\| \lfloor P^{-1} a(-a's + e') \rceil + s \cdot \lfloor P^{-1} a' a \rceil \right\|_\infty \\ &\leq \left\| \lfloor P^{-1} a(-a's + e') \rceil + \lfloor sP^{-1} a' a \rceil \right\|_\infty + \|s\|_1 \\ &\leq \left\| - \lfloor P^{-1} aa's \rceil + \lfloor P^{-1} ae' \rceil + \lfloor sP^{-1} a' a \rceil \right\|_\infty + \|s\|_1 + 1 \\ &\leq N + 2 + \left\| P^{-1} a \cdot e' \right\|_\infty \leq N + 2 + P^{-1} \cdot Nq_\ell \kappa \leq N + 3 \leq 2N. \end{aligned}$$

B The trace and product operators

B.1 The trace operator

For a power-of-two a, we define the operator $\mathsf{Tr}_{N/2\to a}$ as:

$$\mathsf{Tr}_{N/2\to a} = (\mathsf{id} + \Psi_a) \circ (\mathsf{id} + \Psi_{2a}) \circ (\mathsf{id} + \Psi_{4a}) \circ \cdots \circ (\mathsf{id} + \Psi_{N/4}).$$

One can show recursively that the $\mathsf{Tr}_{N/2\to a}$ operator computes partial sums of the slots sharing the same index modulo a, namely for any $\boldsymbol{z} \in \mathbb{C}^{N/2}$:

$$\mathsf{Tr}_{N/2\to a}(\mathsf{iDFT}(z_0, \ldots, z_{N/2-1})) = \mathsf{iDFT}(\underbrace{\boldsymbol{t}, \ldots\ldots, \boldsymbol{t}}_{N/(2a) \text{ rep.}})$$

where $\boldsymbol{t} = (t_k)_{0\le k<a}$, with $t_k = z_k + z_{k+a} + \cdots + z_{k+(N/(2a)-1)\cdot a}$. Therefore, the $\mathsf{Tr}_{N/2\to a}$ operator outputs an encoding of the a slots $t_0, \ldots, t_{a-1}$, which are repeated $N/(2a)$ times to get $N/2$ slots; see Fig. 7 for an illustration.

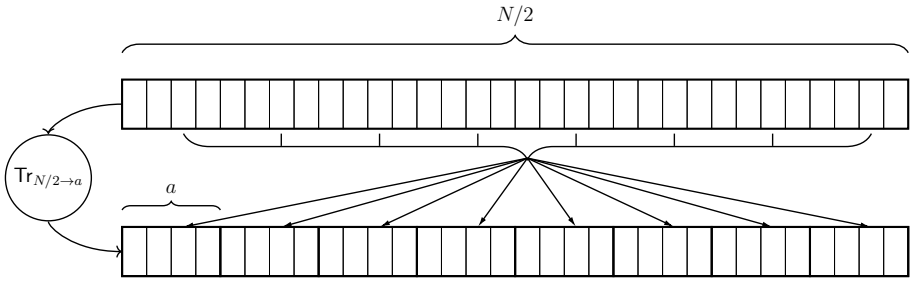


Fig. 7. Illustration of the $\mathsf{Tr}_{N/2\to a}$ operator with $N/2 = 32$ slots and $a = 4$. Each slot in the output polynomial is the sum of all input slots with the same index modulo a. Therefore, the output polynomial contains identical blocks of a slots each.

Namely, the operator $\mathsf{Tr}_{N/2\to N/4} = \mathsf{id} + \Psi_{N/4}$ computes the sum of pairs of slots sharing the same index modulo $N/4$. Assuming the property for $\mathsf{Tr}_{N/2\to a}$ and using $\mathsf{Tr}_{N/2\to a/2} = (\mathsf{id} + \Psi_{a/2}) \circ \mathsf{Tr}_{N/2\to a}$, the operator $\mathsf{Tr}_{N/2\to a/2}$ computes the sum of slots with the same index modulo $a/2$.

As described in Alg. 3 below, the $\mathsf{Tr}_{N/2\to a}$ operator can be homomorphically evaluated, requiring $\log_2(N/(2a))$ automorphisms and key switchings, while consuming no multiplicative levels.

Algorithm 3 Homomorphic trace operator

Input: A ciphertext ct, a power-of-two parameter a
Output: Computes $\mathsf{Tr}_{N/2\to a}(\mathsf{ct})$

1: $r = N/4$
2: **while** $r \ge a$ **do**
3: $\quad \mathsf{ct} = \mathsf{Add}(\mathsf{ct}, \mathsf{Rot}(\mathsf{ct}, r))$
4: $\quad r = r/2$
5: **return** ct

B.2 The product operator

Algorithm 4 Homomorphic product operator

Input: A ciphertext ct, two power-of-two parameters $a > b$
Output: Computes $\mathsf{Pr}_{a\to b}(\mathsf{ct})$

1: $r = a/2$
2: **while** $r \geq b$ **do**
3: $\quad \mathsf{ct} = \mathsf{MultR}_{\mathsf{evk}}(\mathsf{ct}, \mathsf{Rot}(\mathsf{ct}, r))$
4: $\quad r = r/2$
5: **return** ct

C The original CKKS bootstrapping procedure

Bootstrapping a single coefficient. We recall the original CKKS bootstrapping approach [CHK+18a]. For simplicity, we first consider the bootstrapping of a single coefficient, that is a CKKS ciphertext ct encrypting a message $m \in \mathbb{Z}$, with $\langle \mathsf{ct}, \mathsf{sk} \rangle = m + e \pmod q$, where q is the smallest modulus in the ladder.

We first perform a modulus lifting to a larger modulus Q, which gives:

$$\langle \mathsf{ct}, \mathsf{sk} \rangle = qI + m + e \pmod Q$$

for some $I \in \mathbb{Z}[X]/(X^N + 1)$. The goal of bootstrapping is to get rid of the qI term. Using the $\mathsf{Tr}_{N/2\to 1}$ operator and also $\mathsf{Re2} = \mathsf{id} + \mathsf{conj}$, we have $\mathsf{Re2}(\mathsf{Tr}_{N/2\to 1}(qI + m)) = qN \cdot \tilde{I} + N \cdot m$ where $\tilde{I} \in \mathbb{Z}$. Therefore, by applying these operators homomorphically, we can obtain a new ciphertext ct' such that

$$\langle \mathsf{ct}', \mathsf{sk} \rangle = qN \cdot \tilde{I} + Nm + e' \pmod Q$$

for a small error e', so we must remove the term $qN \cdot \tilde{I}$, where now $\tilde{I} \in \mathbb{Z}$ instead of $I \in \mathbb{Z}[X]/(X^N + 1)$.

Such a ciphertext ct' corresponds to an encoding of $x = (qN \cdot \tilde{I} + Nm)/\Delta \in \mathbb{R}$. Eventually, we want to obtain an encryption of m, which corresponds to an encoding of $m/\Delta \in \mathbb{R}$. Therefore, we homomorphically evaluate on ct' a polynomial $P(t)$ such that $m/\Delta \simeq P(x)$. This yields the condition $m/\Delta \simeq P((qN \cdot \tilde{I} + Nm)/\Delta)$, which must hold for any small $\tilde{I} \in \mathbb{Z}$. We expect the integer $\tilde{I}$ to be small because the secret key sk has a small Hamming weight.

For $P(t)$, we can therefore use a polynomial approximation of a periodic function with period qN/Δ, with the appropriate scaling factor:

$$P(t) \simeq \frac{q}{2\pi\Delta} \sin\left(\frac{2\pi\Delta \cdot t}{qN}\right).$$

One could use a Taylor series approximation of the sine function, but the required degree would be too large. Instead, we work over the complex numbers and first consider a degree-d Taylor series approximation of the complex exponential function, with a scaling factor 2^r as input:

$$Q(t) = \sum_{k=0}^{d} \frac{1}{k!} \left(\frac{2i\pi\Delta \cdot t}{qN2^r}\right)^k \simeq \exp\left(\frac{2i\pi\Delta \cdot t}{qN2^r}\right).$$

Due to the scaling factor 2^r, much fewer coefficients are needed in the Taylor series approximation. Note that $Q(t)$ is a polynomial with complex coefficients, so when homomorphically evaluating

$Q(t)$ on ct', we consider the encryptions of the real part and the imaginary part separately. Finally, we can write

$$P(t) = \frac{q}{2\pi\Delta} \cdot \mathsf{Im}\left(Q(t)^{2^r}\right), \tag{14}$$

which we evaluate homomorphically by performing r successive squarings. Eventually, we only keep the encryption of the imaginary part and apply the scaling factor. Finally, we obtain a new ciphertext ct'' such that

$$\begin{aligned}\langle \mathsf{ct}'', \mathsf{sk}\rangle &\simeq \mathsf{Ecd}(P(x)) \simeq \mathsf{Ecd}(m/\Delta) \pmod{q'} \\ &= m + e'' \pmod{q'}\end{aligned}$$

for a small error e''. Therefore, we have obtained a new encryption of m for the modulus q'. The previous evaluation procedure has a multiplicative depth of $\ell = d + r + 1$. Therefore, starting from the largest modulus $Q = q \cdot p \cdot \Delta^\ell$, after ℓ rescalings, we end up with a modulus $q' = q \cdot p$. This enables to perform one more homomorphic multiplication on ct', thereby achieving bootstrapping. We can reduce the depth to $\ell = d + r$ by using a scaling factor δ in $Q(t)$ such that $\delta^{2^r} = q/(2\pi\Delta)$.

Moreover, the depth required to evaluate a polynomial of degree d can be reduced to $\mathcal{O}(\log d)$ at the expense of additional multiplications [CKKS17]. For our parameters, where $d = 7$ is relatively small, this approach slightly reduces the largest modulus Q. However, in practice, we did not observe a significant improvement in running time.

Bootstrapping multiple components. The bootstrapping of multiple components proceeds similarly, which we describe briefly below. For a power-of-two n with $2 \le n \le N/2$, we consider the plaintext space $\mathbb{Z}[X^{N/n}]/(X^N + 1) \simeq \mathbb{Z}[Y]/(Y^n + 1)$ for $Y = X^{N/n}$; therefore, the plaintext space has n non-trivial coefficients. Thus, we start from a ciphertext ct with $\langle \mathsf{ct}, \mathsf{sk}\rangle = m + e \pmod{q}$ for $m \in \mathbb{Z}[X^{N/n}]/(X^N + 1)$.

1. Modulus raising: As previously, we consider the same ct modulo a larger modulus Q, which gives $\langle \mathsf{ct}, \mathsf{sk}\rangle = qI + m + e \pmod{Q}$ for a small $I \in \mathbb{Z}[X]/(X^N + 1)$, where $m(X) = \sum_{i=0}^{n-1} m_i \cdot X^{i \cdot N/n}$.
2. Error packing: We homomorphically apply the $\mathsf{Tr}_{N/2 \to n/2}$ operator, which yields a new ciphertext ct_1, for which $\langle \mathsf{ct}_1, \mathsf{sk}\rangle = q\tilde{I} \cdot N/n + m \cdot N/n + e \pmod{Q}$, where $\tilde{I} \in \mathbb{Z}[X^{N/n}]/(X^N + 1)$.
3. Coefficients to slots: We use the $\mathsf{CoeffToSlot}$ procedure recalled previously to homomorphically compute a new ciphertext ct_2, such that $\langle \mathsf{ct}_2, \mathsf{sk}\rangle = \mathsf{Ecd}(m_0 + q\tilde{I}_0, \ldots, m_{n-1} + q\tilde{I}_{n-1}) + e_2 \pmod{Q_2}$.
4. Approximate modular reduction: The polynomial approximation of the reduction function modulo q gets independently applied on each of the n slots, which then yields a new ciphertext ct_3, for which $\langle \mathsf{ct}_3, \mathsf{sk}\rangle = \mathsf{Ecd}(m_0, \ldots, m_{n-1}) + e_3 \pmod{Q_3}$.
5. Slots to coefficients: We use the $\mathsf{SlotToCoeff}$ procedure recalled previously to homomorphically compute a new ciphertext ct_4, such that $\langle \mathsf{ct}_4, \mathsf{sk}\rangle = m(X) + e_4 \pmod{q \cdot p}$, for a modulus $q \cdot p$, which is larger than the initial modulus q.

Since the complexity of $\mathsf{CoeffToSlot}$ and $\mathsf{SlotToCoeff}$ is $\mathcal{O}(\log n)$ for n slots (using the homomorphic Fast Fourier Transform; see Appendix D), the complexity of the original CKKS bootstrapping for n slots is also $\mathcal{O}(\log n)$.

D Fast DFTs and the SlotToCoeff operation

D.1 The DFT and iDFT isomorphisms

Recall that the M-th cyclotomic polynomial is defined as:

$$\Phi_M(x) = \prod_{\substack{1 \le j \le M \\ \gcd(j,M)=1}} \left(x - e^{2I\pi j/M}\right)$$

For a power-of-two M, we have $\Phi_M(x) = x^N + 1$ where $N = M/2$. For a power-of-two N, the integer 5 has order $N/2$ modulo $2N$, and generates $\mathbb{Z}_{2N}^*$ together with the integer $\{-1\}$. Let $\zeta = e^{I\pi/N}$ be a primitive $2N$-th root of unity. Then letting $\zeta_j := \zeta^{5^j}$ for $0 \le j < N/2$, the set $\{\zeta_j, \overline{\zeta_j} : 0 \le j < N/2\}$ forms the set of the primitive $2N$-th roots of unity.

The variant of the Fourier transform DFT_N over $\mathcal{S}_N$ considered in [CKKS17] is defined as

$$\begin{aligned} \mathsf{DFT}_N : \mathcal{S}_N &\to \mathbb{C}^{N/2} \\ p &\mapsto \left(p\left(\zeta^{5^0}\right), p\left(\zeta^{5^1}\right), \ldots, p\left(\zeta^{5^{N/2-1}}\right)\right). \end{aligned}$$

We denote by $\mathsf{iDFT}_N : \mathbb{C}^{N/2} \to \mathcal{S}_N$ the inverse transform.

The DFT_N function is a ring homomorphism from the ring $\mathcal{S}_N$ (the coefficients space) to the ring of complex vectors in $\mathbb{C}^{N/2}$ (the slots space). Its purpose is therefore to map polynomial addition and multiplication in the ring $\mathcal{S}_N$ to component-wise addition and multiplication in $\mathbb{C}^{N/2}$.

D.2 DFT and iDFT on sub-rings

We consider any divisor n of N with $1 \le n \le N/2$, and a polynomial p in the subring $\mathbb{R}[X^{N/n}]/(X^N + 1)$. We can write $p(X) = \tilde{p}(Y) \in \mathbb{R}[Y]/(Y^n + 1)$ where $Y = X^{N/n}$. Then

$$\mathsf{DFT}_N(p) = \left(p(\zeta^{5^i})\right)_{0 \le i < N/2} = \left(\tilde{p}(\zeta^{5^i \cdot N/n})\right)_{0 \le i < N/2}$$

Since 5 has order $n/2$ modulo $2n$, $\mathsf{DFT}_N(p)$ is periodic with period $n/2$, with N/n repetitions of the same pattern:

$$\mathsf{DFT}_N(p) = (\underbrace{\mathsf{DFT}_n(\tilde{p}), \ldots, \mathsf{DFT}_n(\tilde{p})}_{N/n \text{ rep.}}).$$

Conversely, for any $\boldsymbol{z} \in \mathbb{C}^{n/2}$, we have:

$$\mathsf{iDFT}_N\big(\underbrace{(\boldsymbol{z}, \ldots, \boldsymbol{z})}_{N/n \text{ rep.}}\big) = \mathsf{iDFT}_n(\boldsymbol{z})(X^{N/n})$$

D.3 Computing DFT and iDFT

Let $m(X) = \sum_{i=0}^{N-1} m_i X^i \in \mathbb{R}[X]/(X^N + 1)$ and let $\boldsymbol{m} = (m_0, \ldots, m_{N-1}) \in \mathbb{R}^N$ be the vector of its coefficients. Letting $\zeta = e^{I\pi/N}$ and $\zeta_j := \zeta^{5^j}$ for $0 \le j < N/2$ with $\zeta = e^{I\pi/N}$, we have

$\boldsymbol{z} = \mathsf{DFT}_N(m) = U \cdot \boldsymbol{m} \in \mathbb{C}^{N/2}$ where

$$U = \begin{bmatrix} 1 & \zeta_0 & \dots & \zeta_0^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta_{N/2-1} & \dots & \zeta_{N/2-1}^{N-1} \end{bmatrix}$$

is the $(N/2) \times N$ Vandermonde matrix generated by $\{\zeta_j : 0 \leq j < N/2\}$. The decoding algorithm DFT is therefore a linear transformation from $\mathbb{R}^N$ to $\mathbb{C}^{N/2}$, given by the matrix U.

We consider the full Vandermonde matrix CRT generated by the set $\{\zeta_j, \overline{\zeta_j} : 0 \leq j < N/2\}$, which gives $\mathsf{CRT} = (U; \overline{U})$, and $(\boldsymbol{z}, \overline{\boldsymbol{z}}) = (U\boldsymbol{m}; \overline{U}\boldsymbol{m}) = \mathsf{CRT} \cdot \boldsymbol{m}$. We can then compute $\boldsymbol{m} = \mathsf{CRT}^{-1}(\boldsymbol{z}, \overline{\boldsymbol{z}})$, where $\mathsf{CRT}^{-1} = \frac{1}{N}\overline{\mathsf{CRT}}^T$. This enables to write the coefficients $\boldsymbol{m}$ of the polynomial $m(X) = \mathsf{iDFT}_N(\boldsymbol{z})$ as the linear transformation from $\mathbb{C}^{N/2}$ to $\mathbb{R}^N$:

$$\boldsymbol{m} = \frac{1}{N}\left(\overline{U}^T \cdot \boldsymbol{z} + U^T \cdot \overline{\boldsymbol{z}}\right)$$

D.4 Fast computation of the DFT and bit-reversed order

The drawback of the technique from the previous section to compute the DFT is that its complexity is $\mathcal{O}(N^2)$ operations. In this section, we recall the fast computation of the DFT function, with complexity $\mathcal{O}(N \log N)$.

We work in the ring $\mathcal{S} = \mathbb{R}[X]/(X^N + 1)$. Using $X^N + 1 = (X^{N/2} - I)(X^{N/2} + I)$, we have the isomorphism:

$$\begin{aligned} \Phi : \mathbb{R}[X]/(X^N + 1) &\simeq \mathbb{C}[X]/(X^{N/2} - I) \\ a &\mapsto (a' = a \bmod (X^{N/2} - I)). \end{aligned}$$

The coefficients $a'_j \in \mathbb{C}$ of a' can be computed as $a'_j = a_j + I \cdot a_{j+N/2}$. Conversely, we have $a_j = (a'_j + \bar{a}'_j)/2$ and $a_{j+N/2} = (a'_j - \bar{a}'_j)/(2I)$. Therefore, the real part of the complex coefficients of a' originates from the first $N/2$ coefficients of $a \in \mathcal{S}$, while the imaginary part comes from the last $N/2$ coefficients of a.

We now consider the ring $\mathbb{C}[X]/(X^{N/2} - I)$. We use $\zeta = e^{I\pi/N}$. Using $\zeta^{N/4} = e^{I\pi/4}$ and $\zeta^{5N/4} = e^{5I\pi/4} = -\zeta^{N/4}$, we can further split the ring as follows:

$$\mathbb{C}[X]/(X^{N/2} - I) \simeq \mathbb{C}[X]/(X^{N/4} - \zeta^{N/4 \cdot (5^0 \bmod 8)}) \times \mathbb{C}[X]/(X^{N/4} - \zeta^{N/4 \cdot (5^1 \bmod 8)}).$$

We can generalize this approach as follows. Fix any integer $k \geq 1$ such that $2^k < N$. For any $0 \leq j < 2^{k-1}$, consider the root $\zeta^{N/2^k \cdot 5^j}$. It has two square roots $\zeta^{N/2^{k+1} \cdot 5^j}$ and $-\zeta^{N/2^{k+1} \cdot 5^j}$. Moreover, we have $5^{2^{k-1}} = 1 + 2^{k+1} \pmod{2^{k+2}}$. This gives, for any $0 \leq j < 2^{k-1}$:

$$5^{j+2^{k-1}} = 5^j \cdot (1 + 2^{k+1}) = 5^j + 2^{k+1} \pmod{2^{k+2}}.$$

This enables to write the two square roots as $\zeta^{N/2^{k+1} \cdot 5^j}$ and $\zeta^{N/2^{k+1} \cdot 5^{j+2^{k-1}}}$, which yields the isomorphism for $0 \leq j < 2^{k-1}$:

$$\begin{aligned} \mathbb{C}[X]/(X^{N/2^k} - \zeta^{N/2^k \cdot 5^j}) \simeq \mathbb{C}[X]/\left(X^{N/2^{k+1}} - \zeta^{N/2^{k+1} \cdot 5^j}\right) \\ \times \mathbb{C}[X]/\left(X^{N/2^{k+1}} - \zeta^{N/2^{k+1} \cdot 5^{j+2^{k-1}}}\right). \end{aligned}$$

For the corresponding isomorphism with $u = \zeta^{N/2^{k+1} \cdot 5^j}$

$$\Phi_k(a) = \left(a' = a \bmod (X^{N/2^{k+1}} - u),\ a'' = a \bmod (X^{N/2^{k+1}} + u)\right),$$

The coefficients of a' and a'' can be computed efficiently via

$$a'_i = a_i + u \cdot a_{i+m}$$
$$a''_i = a_i - u \cdot a_{i+m}$$

for $0 \leq i < m$, for $m = N/2^{k+1}$. This is known as the Cooley-Tukey butterfly.

We can then apply this technique recursively. After step k for $1 \leq k \leq \log_2(N/2)$, we arrive at the isomorphism:

$$\mathbb{C}[X]/(X^{N/2} - I) \simeq \prod_{i=0}^{2^k-1} \mathbb{C}[X]/\left(X^{N/2^{k+1}} - \zeta^{N/2^{k+1} \cdot 5^{\mathsf{br}_k(i)}}\right).$$

where $\mathsf{br}_k(i)$ is the bit-reversal of the k-bit integer i. It has the properties $\mathsf{br}_k(2i) = \mathsf{br}_{k-1}(i)$ and $\mathsf{br}_k(2i+1) = 2^{k-1} + \mathsf{br}_{k-1}(i)$.

Eventually, for $N = 2^\ell$, we obtain the evaluation of the input polynomial at the roots $\zeta^{5^{\mathsf{br}_{\ell-1}(i)}}$ for $0 \leq i < N/2$, therefore in bit-reversed order. Recall that we have defined $\mathsf{DFT}_N(p)$ as $\left(p(\zeta^{5^i})\right)_{0 \leq i < N/2}$. Let Br_k denote the function taking as input a sequence of 2^k integers and outputting it in bit-reversed order. With this notation, the above procedure yields a sequence $(v_i)_{0 \leq i < N/2} = \left(p(\zeta^{5^{\mathsf{br}_{\ell-1}(i)}})\right)_{0 \leq i < N/2} = \mathsf{Br}_{\ell-1}(\mathsf{DFT}_N(p))$. In total, we obtain the following fast DFT_N algorithm, from normal order (no) to bit-reversed order (bo).

Algorithm 5 Fast computation of DFT_N, $\mathsf{no} \to \mathsf{bo}$

Input: A polynomial $p(X) \in \mathbb{R}[X]/(X^N + 1)$, with $p(X) = \sum_{j=0}^{N-1} p_j X^j$, $\zeta = \exp(I\pi/N)$ and $N = 2^\ell$.

Output: A sequence $(v_i)_{0 \leq i < N/2} = \mathsf{Br}_{\ell-1}(\mathsf{DFT}_N(p)) = \left(p(\zeta^{5^{\mathsf{br}_{\ell-1}(i)}})\right)_{0 \leq i < N/2}$.

$(v_j)_{0 \leq j < N/2} = (p_j + I \cdot p_{j+N/2})_{0 \leq j < N/2}$
for $k = 1$ to $\ell - 1$ **do**
  for $j = 0$ to $2^{k-1} - 1$ **do**
    $u = \zeta^{5^{\mathsf{br}_{k-1}(j)} \cdot N/2^{k+1}}$
    for $i = 0$ to $N/2^{k+1} - 1$ **do**
      $a, b \leftarrow v_{j \cdot N/2^k + i},\ v_{j \cdot N/2^k + i + N/2^{k+1}}$
      $v_{j \cdot N/2^k + i} = a + u \cdot b$
      $v_{j \cdot N/2^k + i + N/2^{k+1}} = a - u \cdot b$
return $(v_i)_{0 \leq i < N/2}$

We describe the corresponding iDFT algorithm, that computes the same operations in reverse order.

Algorithm 6 Fast computation of iDFT_N, $\mathsf{bo} \rightarrow \mathsf{no}$

Input: A sequence $(v_i)_{0 \leq i < N/2}$, $\zeta = \exp(I\pi/N)$ and $N = 2^\ell$.
Output: A polynomial $p(x) \in \mathbb{R}[X]/(X^N + 1)$, such that $(v_i)_{0 \leq i < N/2} = \mathsf{Br}_{\ell-1}(\mathsf{DFT}_N(p))$

1: **for** $k = \ell - 1$ downto 1 **do**
2: **for** $j = 0$ to $2^{k-1} - 1$ **do**
3: $u = \zeta^{5^{\mathsf{br}_{k-1}(j)} \cdot N/2^{k+1}}$
4: **for** $i = 0$ to $N/2^{k+1} - 1$ **do**
5: $a, b \leftarrow v_{j \cdot N/2^k + i},\ v_{j \cdot N/2^k + i + N/2^{k+1}}$
6: $v_{j \cdot N/2^k + i} = (a + b)/2$
7: $v_{j \cdot N/2^k + i + N/2^{k+1}} = u^{-1} \cdot (a - b)/2$
8: **return** $p(x) = \sum_{i=0}^{N/2-1} \mathsf{Re}(v_i) x^i + \mathsf{Im}(v_i) x^{i+N/2}$

In Algorithm 7, we describe the same DFT_N algorithm as in Alg. 5, but with the input coefficients of the polynomial in bit-reversed order, and the output coefficients in normal order. In fact, we use a modified bit-reversed indexing, in which the first and second half of the coefficients are encoded separately in bit-reversed order. This will later facilitate the homomorphic evaluation of the algorithm.

Due to this modified bit-reversed encoding, Line 1 of Alg. 5 remains the same. At Line 3, we can equivalently run j' from 0 to $2^{k-1} - 1$, and let $j = \mathsf{br}_{k-1}(j')$. At lines 6, 7, 8, we have

$$\mathsf{br}_{\ell-1}(j \cdot N/2^k + i) = \mathsf{br}_{\ell-k-1}(i) \cdot 2^k + \mathsf{br}_{k-1}(j)$$
$$\mathsf{br}_{\ell-1}(j \cdot N/2^k + i + N/2^{k+1}) = \mathsf{br}_{\ell-k-1}(i) \cdot 2^k + 2^{k-1} + \mathsf{br}_{k-1}(j)$$

This gives the following algorithm.

Algorithm 7 Fast computation of DFT_N, $\mathsf{bo}' \rightarrow \mathsf{no}$

Input: A sequence $(p_i)_{0 \leq i < N/2} \in \mathbb{R}^{N/2}$, $\zeta = \exp(I\pi/N)$ and $N = 2^\ell$.

Output: A sequence $(v_i)_{0 \leq i < N/2} = \mathsf{DFT}(p) = \left(p(\zeta^{5^i})\right)_{0 \leq i < N/2}$, where $p(X) = \sum_{j=0}^{N/2-1} p_{\mathsf{br}_{\ell-1}(j)} X^j + \sum_{j=0}^{N/2-1} p_{N/2+\mathsf{br}_{\ell-1}(j)} X^{N/2+j}$.

1: $(v_j)_{0 \leq j < N/2} = (p_j + I \cdot p_{j+N/2})_{0 \leq j < N/2}$
2: **for** $k = 1$ to $\ell - 1$ **do**
3: **for** $j = 0$ to $2^{k-1} - 1$ **do**
4: $u = \zeta^{5^j \cdot N/2^{k+1}}$
5: **for** $i = 0$ to $N/2^{k+1} - 1$ **do**
6: $a, b \leftarrow v_{i \cdot 2^k + j},\ v_{i \cdot 2^k + 2^{k-1} + j}$
7: $v_{i \cdot 2^k + j} = a + u \cdot b$
8: $v_{i \cdot 2^k + 2^{k-1} + j} = a - u \cdot b$
9: **return** $(v_i)_{0 \leq i < N/2}$

D.5 Definition of the SlotToCoeff function

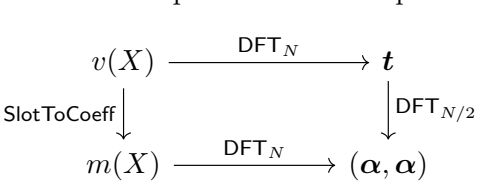
For convenience, we define the $\mathsf{SlotToCoeff}_N$ operation over the cyclotomic ring $\mathcal{S}_N$, but the procedure will indeed be the same over the integer ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$. The $\mathsf{SlotToCoeff}_N$ function takes as input a polynomial $v(X)$ encoding $N/2$ slots $\boldsymbol{t} = (t_0, \ldots, t_{N/2-1}) \in \mathbb{R}^{N/2}$, that is $v(X) = \mathsf{iDFT}_N(\boldsymbol{t})$. It outputs a polynomial $m(X) \in \mathbb{R}[X^2]/(X^N + 1)$, whose $N/2$ coefficients

are the t_j's:

$$m(X) = \mathsf{SlotToCoeff}_N(v) = \sum_{j=0}^{N/2-1} t_j \cdot X^{2j}.$$

We can therefore write $m(X) = \tilde{m}(X^2)$ for $\tilde{m}(Y) = \sum_{j=0}^{N/2-1} t_j \cdot Y^j$, and we let $\boldsymbol{\alpha} = \mathsf{DFT}_{N/2}(\tilde{m}) \in \mathbb{C}^{N/4}$. This implies $\mathsf{DFT}_N(m) = (\boldsymbol{\alpha}, \boldsymbol{\alpha}) \in \mathbb{C}^{N/2}$.

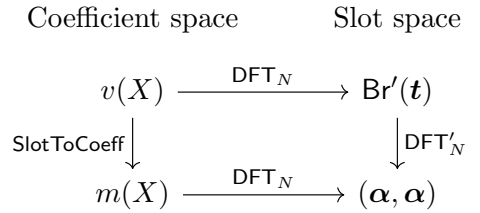
We generalize the DFT function to take as input the coefficient vector of a polynomial, instead of the polynomial itself, with the same output. Therefore, since $\boldsymbol{\alpha} = \mathsf{DFT}_{N/2}(\tilde{m})$ and by definition $\tilde{m}(Y) = \sum_{j=0}^{n-1} t_j \cdot Y^j$, we can write $\boldsymbol{\alpha} = \mathsf{DFT}_{N/2}(\boldsymbol{t})$. In total, we can summarize the above in a commutative diagram:



From the above commutative diagram, the goal is therefore to compute the function $\mathbb{R}^{N/2} \to \mathbb{C}^{N/2}$, $\boldsymbol{t} \to (\mathsf{DFT}_{N/2}(\boldsymbol{t}), \mathsf{DFT}_{N/2}(\boldsymbol{t}))$ over the slot space, homomorphically over the coefficient space.

D.6 The SlotToCoeff algorithm

In fact, we cannot compute $\boldsymbol{\alpha} = \mathsf{DFT}_{N/2}(\boldsymbol{t})$ directly with a fast $\mathcal{O}(\log N)$ algorithm. Either the input or the output must be indexed in bit-reversed order. We will use the function $\mathsf{DFT}_{N/2,\mathsf{bo}'\to\mathsf{no}}$ computed by Alg. 7, whose input is bit-reversed and whose output is in normal order, which gives $\mathsf{DFT}_{N/2,\mathsf{bo}'\to\mathsf{no}}(\boldsymbol{t}') = \mathsf{DFT}_{N/2}(\boldsymbol{t})$ where $\boldsymbol{t}' = \mathsf{Br}'(\boldsymbol{t})$. We therefore have the updated commutative diagram:



where we define the function $\mathsf{DFT}'_N(\boldsymbol{t}) = (\mathsf{DFT}_{N/2,\mathsf{bo}'\to\mathsf{no}}(\boldsymbol{t}), \mathsf{DFT}_{N/2,\mathsf{bo}'\to\mathsf{no}}(\boldsymbol{t}))$. The goal is therefore to evaluate the function $\mathsf{DFT}'_N(\boldsymbol{t}')$ homomorphically over the coefficient space.

Algorithm 8 Fast computation of DFT'_N

Input: A sequence $(p_i)_{0\le i<N/2} \in \mathbb{R}^{N/2}$, $\zeta = \exp(I\pi/N)$ and $N = 2^\ell$.
Output: A sequence $(v_i)_{0\le i<N/2} = \mathsf{DFT}'(\boldsymbol{p})$

1: $(v_j)_{0\le j<N/2} = (\boldsymbol{w}, \boldsymbol{w})$, where $\boldsymbol{w} = (p_j + I \cdot p_{j+N/4})_{0\le j<N/4}$
2: **for** $k = 1$ to $\ell - 2$ **do**
3: **for** $j = 0$ to $2^{k-1} - 1$ **do**
4: $u = \zeta^{5^j \cdot N/2^{k+1}}$
5: **for** $i = 0$ to $N/2^{k+1} - 1$ **do**
6: $a, b \leftarrow v_{i\cdot 2^k + j},\ v_{i\cdot 2^k + 2^{k-1} + j}$
7: $v_{i\cdot 2^k + j} = a + u \cdot b$
8: $v_{i\cdot 2^k + 2^{k-1} + j} = a - u \cdot b$
9: **return** $(v_i)_{0\le i<N/2}$

To perform the homomorphic evaluation of the previous algorithm, we first rewrite the operations in vector form. For a vector $\boldsymbol{v} = (v_i)_{0 \le i < N/2}$, we write

$$\mathsf{Rot}_r(\boldsymbol{r}) = (v_{r+j \bmod N/2})_{0 \le j < N/2} = (v_r, \ldots, v_{N/2-1}, v_0, \ldots, v_{r-1})$$

the rotation of $\boldsymbol{v}$ by r positions to the left. For a vector $\boldsymbol{a}$, we denote by $\boldsymbol{a}^n$ the repetition n times of $\boldsymbol{a}$.

For the first line, we can write:

$$\boldsymbol{v} = \boldsymbol{p} \odot ((1)^{N/4}, (I)^{N/4}) + \mathsf{Rot}_{N/4}(\boldsymbol{p}) \odot ((I)^{N/4}, (1)^{N/4})$$

We can rewrite the first butterfly equation as:

$$v_{i \cdot 2^k + j} \leftarrow v_{i \cdot 2^k + j} + \zeta^{5^j \cdot N/2^{k+1}} \cdot (\mathsf{Rot}_{2^{k-1}}(\boldsymbol{v}))_{i \cdot 2^k + j}$$

Similarly, we can rewrite the second butterfly equation as:

$$v_{i \cdot 2^k + 2^{k-1} + j} \leftarrow (\mathsf{Rot}_{-2^{k-1}}(\boldsymbol{v}))_{i \cdot 2^k + 2^{k-1} + j} - \zeta^{5^j \cdot N/2^{k+1}} \cdot v_{i \cdot 2^k + 2^{k-1} + j}$$

We can therefore write:

$$\boldsymbol{v} \leftarrow \boldsymbol{v} \odot \boldsymbol{\omega}_0 + \mathsf{Rot}_{2^{k-1}}(\boldsymbol{v}) \odot \boldsymbol{\omega}_1 + \mathsf{Rot}_{-2^{k-1}}(\boldsymbol{v}) \odot \boldsymbol{\omega}_2$$

where

$$\boldsymbol{\omega}_0 = \left((1)^{2^{k-1}}, \left(-\zeta^{5^j \cdot N/2^{k+1}} \right)_{0 \le j < 2^{k-1}} \right)^{N/2^{k+1}}$$

$$\boldsymbol{\omega}_1 = \left(\left(\zeta^{5^j \cdot N/2^{k+1}} \right)_{0 \le j < 2^{k-1}}, \ (0)^{2^{k-1}} \right)^{N/2^{k+1}}$$

$$\boldsymbol{\omega}_2 = \left((0)^{2^{k-1}}, \ (1)^{2^{k-1}} \right)^{N/2^{k+1}}$$

Finally, we can now define the SlotToCoeff algorithm, which is a homomorphic evaluation of the previous algorithm (Alg. 8).

Algorithm 9 SlotToCoeff

Input: A polynomial $v = \mathsf{iDFT}_N(\mathsf{Br}'(\boldsymbol{t}))$ for $\boldsymbol{t} \in \mathbb{R}^{N/2}$, $\zeta = \exp(I\pi/N)$ and $N = 2^\ell$.

Output: A polynomial $m(X) = \mathsf{SlotToCoeff}_N(v) \simeq \sum_{j=0}^{N/2-1} t_j \cdot X^{2j}$

1: $v \leftarrow v \times \mathsf{Ecd}((1)^{N/4}, (I)^{N/4})) + \Psi_{N/4}(v) \times \mathsf{Ecd}((I)^{N/4}, (1)^{N/4})$
2: **for** $k = 1$ to $\ell - 2$ **do**
3: $\quad \boldsymbol{w}_0 = \left((1)^{2^{k-1}}, \left(-\zeta^{5^j \cdot N/2^{k+1}} \right)_{0 \le j < 2^{k-1}} \right) \in \mathbb{C}^{2^k}$
4: $\quad W_0 = \mathsf{Ecd}(\boldsymbol{w}_0)$
5: $\quad \boldsymbol{w}_1 = \left(\left(\zeta^{5^j \cdot N/2^{k+1}} \right)_{0 \le j < 2^{k-1}}, \ (0)^{2^{k-1}} \right) \in \mathbb{C}^{2^k}$
6: $\quad W_1 = \mathsf{Ecd}(\boldsymbol{w}_1)$
7: $\quad \boldsymbol{w}_2 = \left((0)^{2^{k-1}}, \ (1)^{2^{k-1}} \right) \in \mathbb{C}^{2^k}$
8: $\quad W_2 = \mathsf{Ecd}(\boldsymbol{w}_2)$
9: $\quad v \leftarrow v \times W_0 + \Psi_{2^{k-1}}(v) \times W_1 + \Psi_{-2^{k-1}}(v) \times W_2$
10: **return** v

The above SlotToCoeff algorithm is essentially the same as in [CHH18], although in the latter it is described via a matrix approach. It has multiplicative depth $\log_2(N/2)$, and requires $2\log_2(N/2) - 1$ homomorphic rotations and $3\log_2(N/2) - 1$ external multiplications. Note that the polynomials W_0, W_1, and W_2 are independent of the input and can thus be precomputed.

D.7 SlotToCoeff for n slots

We have the updated commutative diagram:

$$\begin{array}{ccccc}
\text{Coefficient space} & & \text{Coefficient space} & & \text{Slot space} \\
\mathbb{Z}[X^{N/(2n)}]/(X^N+1) & & \mathbb{Z}[Y]/(Y^{2n}+1) & & \mathbb{C}^n \\
v(X) & \longrightarrow & \tilde{v}(Y) & \xrightarrow{\mathsf{DFT}_{2n}} & \mathsf{Br}'(\boldsymbol{t}) \\
\Big\downarrow{\scriptstyle \mathsf{SlotToCoeff}_{N,n}} & & \Big\downarrow{\scriptstyle \mathsf{SlotToCoeff}_{2n}} & & \Big\downarrow{\scriptstyle \mathsf{DFT}'_{2n}} \\
m(X) & \longrightarrow & \tilde{m}(Y) & \xrightarrow{\mathsf{DFT}_{2n}} & (\boldsymbol{\alpha}, \boldsymbol{\alpha})
\end{array}$$

Algorithm 10 $\mathsf{SlotToCoeff}_{N,n}$

Input: A polynomial $v = \mathsf{iDFT}_N(\mathsf{Br}'(\boldsymbol{t})^{N/(2n)})$ for $\boldsymbol{t} \in \mathbb{R}^n$, $\zeta = \exp(I\pi/(2n))$ and $n = 2^\ell$.

Output: A polynomial $m(X) = \mathsf{SlotToCoeff}_{N,n}(v) \simeq \sum_{j=0}^{n-1} t_j \cdot X^{j \cdot N/n}$

$v \leftarrow v \times \mathsf{Ecd}((1)^{n/2}, (I)^{n/2})) + \Psi_{n/2}(v) \times \mathsf{Ecd}((I)^{n/2}, (1)^{n/2})$
for $k = 1$ to $\ell - 1$ **do**
  $\boldsymbol{w}_0 = \left((1)^{2^{k-1}}, \left(-\zeta^{5^j \cdot n/2^k}\right)_{0 \le j < 2^{k-1}}\right) \in \mathbb{C}^{2^k}$
  $W_0 = \mathsf{Ecd}(\boldsymbol{w}_0)$
  $\boldsymbol{w}_1 = \left(\left(\zeta^{5^j \cdot n/2^k}\right)_{0 \le j < 2^{k-1}}, \ (0)^{2^{k-1}}\right) \in \mathbb{C}^{2^k}$
  $W_1 = \mathsf{Ecd}(\boldsymbol{w}_1)$
  $\boldsymbol{w}_2 = \left((0)^{2^{k-1}}, \ (1)^{2^{k-1}}\right) \in \mathbb{C}^{2^k}$
  $W_2 = \mathsf{Ecd}(\boldsymbol{w}_2)$
  $v \leftarrow v \times W_0 + \Psi_{2^{k-1}}(v) \times W_1 + \Psi_{-2^{k-1}}(v) \times W_2$
return v

D.8 Radix implementation

As in [CHH18], we can reduce the multiplicative depth by using radices. For a power-of-two radix r, [CHH18] shows that the homomorphic linear transform requires roughly $2r \log_r n$ homomorphic rotations and $3r \log_r n$ external multiplications. The advantage of the radix-based approach is that it yields a smaller multiplicative depth of around $\log_r n$ compared to the initial depth of $\log_2 n$. However, since the complexity increases as the radix does, the radix variant can be seen as a trade-off, which becomes disadvantageous for a too large r. As also described in [CHH18], without affecting the depth, the number of homomorphic rotations can be even further reduced to $O(\sqrt{r} \log_r n)$ by using a baby-step giant-step approach.

In practice, the authors of [CHH18] suggest to use $r = 2^5$ for a full slots implementation of CKKS, *i.e.* with common parameters $n = N/2$ and $N = 2^{15}$ or $N = 2^{16}$. They show that the running times for radices ranging from 2^1 to 2^4 are quite similar. However, for few slots n, we can use a smaller radix since a radix r is only of avail if $r \le \sqrt{n}$. For our implementation, we chose to use a radix $r = 2^2$, which yields a depth of $1 + \lfloor (\log_2 n)/2 \rfloor$.

Practical experiments. Table 7 presents a running time comparison of our new bootstrapping for different radix values with $n = 64$ slots. While increasing the radix reduces the size of the largest modulus Q, the optimal radix value for $n = 64$ is found to be $r = 4$.

Parameter	r	ℓ	$\log_2 Q$	time
Set-I	2	13	497	47 s
	4	11	427	45 s
	8	10	392	55 s

Table 7. Running time of our new bootstrapping for Set-I parameters and $n = 64$, for various radix values r.

E Proof of Theorem 1

We start with fresh encryptions of the secret-key bits s_k. By Lemma 1, their initial noise size is bounded by $E = 3N\kappa$. By Lemma 4, since we perform external products with encodings of real values of absolute value less than $\nu = 1$, the size of the noise of each product is at most $2\nu E + 8N \leq 14N\kappa$.

For the product tree, we must consider the error obtained when computing the product of the encryptions of two complex values, whose real and imaginary part are encrypted separately. Given the plaintexts $z_1 = x_1 + i \cdot y_1$ and $z_2 = x_2 + i \cdot y_2$, we have $z_1 z_2 = x_1 x_2 - y_1 y_2 + i \cdot (x_1 y_2 + x_2 y_1)$. Therefore, for $x = x_1 x_2 - y_1 y_2$, we must compute the homomorphic difference:

$$\mathsf{ct}_x \leftarrow \mathsf{MultR}_{\mathsf{evk}}(\mathsf{ct}_{x_1}, \mathsf{ct}_{x_2}) - \mathsf{MultR}_{\mathsf{evk}}(\mathsf{ct}_{y_1}, \mathsf{ct}_{y_2}),$$

and similarly for ct_y for $y = x_1 y_2 + x_2 y_1$. Since all plaintext values x_1, y_1, x_2, y_2 are bounded by 1 in absolute value, the errors in $\mathsf{MultR}(\mathsf{ct}_{x_1}, \mathsf{ct}_{x_2})$ and $\mathsf{MultR}(\mathsf{ct}_{y_1}, \mathsf{ct}_{y_2})$ are upper bounded by $2E + 8N$, where E is an upper bound of the errors in ct_{x_1}, ct_{x_2}, ct_{y_1} and ct_{y_2}. Therefore, the error magnitudes of ct_x and ct_y are both upper-bounded by $4E + 16N$.

We now consider the product tree of depth $\ell = \log_2 N$. For $0 \leq i \leq \ell$, we can upper-bound the noise of the current ciphertext at level i by $u_i N\kappa$, where $u_0 = 14$, and the recursive relation $u_{i+1} = 4u_i + 16$ holds. One can show that $u_i \leq 20 \cdot 2^{2i} - 6$ by a recursive approach. For $i = \ell$, the error size after all multiplications is thus upper-bounded by $20 \cdot 2^{2\ell} N\kappa = 20N^3\kappa$. We must therefore assume that $\Delta \geq (20N^3\kappa)^2$ since we require $\Delta \geq E^2$ in each multiplication.

After the final scaling by $q/(2\pi\Delta)$, and by applying Lemma 4 again, the noise size becomes $20N^3\kappa \cdot q/(2\pi\Delta) + 8N$. Therefore, for $\Delta \geq 4N^3\kappa q$, the size of the noise is at most $9N$. This implies that by writing $m = \langle \mathsf{ct}, \mathsf{sk} \rangle(0) = \langle \boldsymbol{s}, \boldsymbol{c} \rangle \in \mathbb{Z}$, our final ciphertext ct' satisfies:

$$\langle \mathsf{ct}', \mathsf{sk} \rangle = \mathsf{Ecd}(F(m)) + e' \pmod{p \cdot q},$$

for some e' with $\|e'\|_\infty \leq 9N$ and

$$F(m) = \frac{q}{2\pi\Delta} \sin\left(\frac{2i\pi m}{q}\right).$$

Given the condition $|m| < q^{2/3}$, we obtain by bounding the residual term in the Taylor expansion:

$$\left|F(m) - \frac{m}{\Delta}\right| \leq \frac{q}{2\pi\Delta} \cdot \frac{1}{3!} \cdot \left(\frac{2\pi|m|}{q}\right)^3 \leq \frac{7|m|^3}{q^2\Delta} \leq \frac{7}{\Delta}.$$

This implies $|\mathsf{Ecd}(F(m)) - m| \leq \Delta \cdot |F(m) - m/\Delta| + 1 \leq 8$. We can eventually write:

$$\langle \mathsf{ct}', \mathsf{sk} \rangle = \langle \mathsf{ct}, \mathsf{sk} \rangle(0) + e_{\mathsf{bt}} \pmod{p \cdot q},$$

where $\|e_{\mathsf{bt}}\|_\infty \leq 10N$ as claimed.

F Algorithms for the single slot bootstrapping

Algorithm 11 Bootstrapping key generation, single slot

Input: A length N secret key $\boldsymbol{s}$ with Hamming weight h and $B = N/h$
Output: A bootstrapping key $\mathsf{cs} = (\mathsf{cs}_0, \mathsf{cs}_1)$

1: **for all** $0 \leq j < B/2$ **and** $0 \leq b < h$ **do**
2: $\quad \tilde{s}^0_{j \cdot h + b} = s_{b \cdot B + j}$
3: $\quad \tilde{s}^1_{j \cdot h + b} = s_{b \cdot B + B/2 + j}$
4: $S_0(X), S_1(X) \longleftarrow \mathsf{Ecd}((\tilde{s}^0_\iota)_{0 \leq \iota < N/2}), \mathsf{Ecd}((\tilde{s}^1_\iota)_{0 \leq \iota < N/2})$
5: **return** $\mathsf{cs} = (\mathsf{Enc}_{\mathsf{pk}}(S_0(X)), \mathsf{Enc}_{\mathsf{pk}}(S_1(X)))$

Algorithm 12 Bootstrapping, single slot

Input: A modulus q, a bootstrapping key cs, an RLWE ciphertext $\mathsf{ct} = (b, a)$, and $\delta = (q/(4\pi\Delta))^{1/h}$
Output: A refreshed ciphertext ct'

1: $\boldsymbol{c} = (a_0 + b_0, -a_{N-1}, -a_{N-2}, \ldots, -a_1)$
2: **for all** $0 \leq j < B/2$ **and** $0 \leq b < h$ **do**
3: $\quad e^0_{j \cdot h + b} = \exp(2i\pi \cdot c_{b \cdot B + j}/q) \cdot \delta$
4: $\quad e^1_{j \cdot h + b} = \exp(2i\pi \cdot c_{b \cdot B + B/2 + j}/q) \cdot \delta$
5: $E_0, E_1 \longleftarrow \mathsf{Ecd}((e^i_\iota)_{0 \leq \iota < N/2}), \mathsf{Ecd}((e^1_\iota)_{0 \leq \iota < N/2})$
6: $T_0, T_1 \longleftarrow \mathsf{ExtMultR}(\mathsf{cs}_0, E_0), \mathsf{ExtMultR}(\mathsf{cs}_1, E_1)$
7: **return** $\mathsf{ct}' = \mathsf{Im2}(\mathsf{Pr}_{h \to 1}(\mathsf{Tr}_{N/2 \to h}(T_0 + T_1)))$

G Bootstrapping algorithm for more than $B/2$ slots

The bootstrapping algorithm from Section 5.2 is limited to bootstrapping up to $n \leq n_{\mathsf{max}} = B/2 = N/(2h)$ components. However, it can be easily extended to support more slots, specifically $n' \leq N$, by shifting the coefficients of the input ciphertext and applying the previous bootstrapping procedure as a black box for each group of n_{max} coefficients.

More precisely, the previous bootstrapping algorithm (Alg. 2) inputs a ciphertext ct for the plaintext polynomial $m(X) = \sum_{i=0}^{N-1} m_i \cdot X^i$, and outputs a refreshed ciphertext for the polynomial $m'(X) = \sum_{i=0}^{n-1} m_{i \cdot N/n} \cdot X^{i \cdot N/n}$. In other words, only the n coefficients m_i of $m(X)$ such that $i \equiv 0 \pmod{N/n}$ are kept and bootstrapped, while the others are lost, because of the initial $C \leftarrow \mathsf{DecMat}(\mathsf{ct})$ procedure which only considers the coefficients of $X^{i \cdot N/n}$ in $m(X)$.

To bootstrap $n' > n$ coefficients, we can therefore proceed by repeatedly shifting the coefficients of the input plaintext (homomorphically on the input ciphertext), by a shifting polynomial $X^{-j \cdot N/n'}$. We obtain a bootstrapped ciphertext, which we shift back by $X^{j \cdot N/n'}$. We then compute the sum for all $0 \leq j < n'/n$ of all such ciphertexts. At the end, we obtain a refreshed ciphertext for all the n' components. We describe the concrete algorithm below. The number of homomorphic operations becomes $\mathcal{O}(n' + (n'/n) \cdot \log N)$, where $n = n_{\mathsf{max}} = B/2$ and $n' \leq N/2$, which gives $n'/n \leq h$. Assuming $h = \mathcal{O}(1)$, the complexity remains $\mathcal{O}(n' + \log N)$, while the depth remains unchanged.

Algorithm 13 Bootstrapping, more than $B/2$ slots

Input: A modulus q, a bootstrapping key $(\mathsf{CS}_u)_{0\le u<2n}$, an RLWE ciphertext $\mathsf{ct} = (b, a)$ containing $n' \ge n$ slots
Output: A refreshed ciphertext ct'

$\mathsf{acc} \leftarrow (0, 0)$
for $j = 0$ to $n'/n - 1$ **do**
  $\mathsf{ct}_s \leftarrow \mathsf{ExtMult}(\mathsf{ct}, X^{-j\cdot N/n'})$
  $\mathsf{ct}'_s \leftarrow \mathsf{Bootstrap}(q, \mathsf{cs}, \mathsf{ct}_s)$
  $\mathsf{acc} \leftarrow \mathsf{Add}(\mathsf{acc}, \mathsf{ExtMult}(\mathsf{ct}'_s, X^{j\cdot N/n'}))$
return acc

H Number of operations

For simplicity, we consider only the operations involving polynomial multiplications, since they dominate the asymptotic running time. A ciphertext multiplication requires 6 polynomial multiplications, including 2 multiplications modulo Q^2 instead of modulo Q, which roughly take twice in running time; therefore, an equivalent of 8 polynomial multiplications. An external product requires 2 multiplications only. A key switching requires 2 multiplications modulo Q^2, so an equivalent of 4 polynomial multiplications. We summarize the relative costs in Table 8.

Operation	MultR	ExtMultR	KS	Rot	Im2
Relative cost	1	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$

Table 8. Relative cost of each operation, compared to a ciphertext multiplication $\mathsf{MultR}_{\mathsf{evk}}$.

Using a radix-4 implementation, the CoeffToSlot and SlotToCoeff operations both require $(7\log_2 n)/2 - 2$ external multiplications (ExtMultR) for even $\log_2 n$, and $(7\log_2 n)/2 - 3/2$ for odd $\log_2 n$. They also both require $2\log_2 n - 1$ rotations (Rot). Therefore, the total complexity relative to a ciphertext multiplication is $15(\log_2 n)/8 - a$, where $a = 7/8$ for even $\log_2 n$, and $a = 1$ for odd $\log_2 n$.

CKKS bootstrapping. As recalled in Section 1, the original CKKS bootstrapping comprises the following operation. The error packing applies the $\mathsf{Tr}_{N/2\to n/2}$ operator (for $n > 2$) and therefore requires $\log_2(N/n)$ rotations; this is also true for $n = 1$. The polynomial evaluation requires $d + r$ ciphertext multiplications. Including CoeffToSlot and SlotToCoeff, the total complexity of the CKKS bootstrapping in terms of ciphertext multiplications is therefore, for $n \ge 2$:

$$\begin{aligned} T_{\mathsf{CKKS}} &= \frac{1}{2}\cdot\log_2(N/n) + d + r + 15\cdot(\log_2 n)/4 - 2a \\ &= \frac{1}{2}\log_2 N + d + r + \frac{13}{4}\log_2 n - 2a. \end{aligned}$$

For $n = 1$, we have $T_{\mathsf{CKKS}} = (\log_2 N)/2 + d + r$.

Our new bootstrapping. Our new bootstrapping performs $2n$ external multiplications. The $\mathsf{Tr}_{N/2\to hn}$ operator requires $\log_2(N/(2nh))$ rotations, and the $\mathsf{Pr}_{hn\to n}$ operator requires $\log_2 h$ rotations and ciphertext multiplications. The Im2 operator executes a single rotation. Therefore,

the total complexity including SlotToCoeff is:

$$\begin{aligned} T_{\text{new}} &= \frac{2n}{4} + \frac{1}{2}\log_2(N/(2nh)) + \frac{1}{2}\log_2 h + \log_2 h + \frac{1}{2} + 15 \cdot (\log_2 n)/8 - a \\ &= \frac{1}{2}\log_2(N/2) + \log_2 h + \frac{n+1}{2} + \frac{11}{8}\log_2 n - a. \end{aligned}$$