FHE for SIMD Arithmetic Logic Units with Amortized $O(1)$ Bootstrapping per Ciphertext

Mingyu Gao[1,2(✉)] and Hongren Zheng[1]

[1] Tsinghua University
[2] Shanghai Qi Zhi Institute
gaomy@tsinghua.edu.cn
zhenghr22@mails.tsinghua.edu.cn

Abstract. The evaluation of both arithmetic and logic operations on machine words (e.g., 64-bit registers) in homomorphic settings is an attractive problem due to its close alignment with real-world programming models. Existing FHE schemes require *iterative* bootstrapping operations with the iteration number scaling with the machine word bit-width n. Prior approaches incur either quadratic cost for multiplication (DM/CGGI), linear cost for logic operations (REFHE [Eurocrypt'26] and Kim [TCHES'26]), or logarithmic cost for multiplication (CPL [Eurocrypt'26]).

We mitigate this inherent barrier by amortizing $O(n)$ iterations across $O(n)$ ciphertexts for scenarios with sufficient machine words. We introduce a *triangle encoding* such that in its conversion to boolean mode, each iteration on a combined ciphertext processes multiple ciphertexts, resulting in an *amortized constant* cost per ciphertext. Additionally, the triangle encoding supports several multiplication operations before needing a refreshing that requires only two CKKS bootstrapping operations, regardless of the bit-width n. We reuse discrete CKKS as our boolean mode, which supports lightweight conversion to the triangle encoding. We also introduce a method to induce SIMD structure for the triangle encoding from the RLWE ring.

Even when we bootstrap after each multiplication, we show a 1.7× to 2× higher throughput compared with the state-of-the-art CPL scheme. We further estimate 15× and 491× higher throughput for multiplication and bitwise operations, respectively, compared with the REFHE scheme. We make our OpenFHE-based prototype open source for ease of reference.

1 Introduction

Fully Homomorphic Encryption (FHE) enables unbounded computation on encrypted data. It was a long-standing open problem [RAD78] until Gentry made a breakthrough [Gen09] by introducing the bootstrapping technique. Subsequent early efforts like [vDGHV10, SV10, BV11, GHS12, GSW13] have made various improvements to make FHE more practical. Most of them are based on modern hardness problems like LWE [Reg05] and its variants like Ring-LWE [LPR13] and Polynomial-LWE [LPR10, LPR13, SSTX09].

Modern schemes often excel at different types of operations. For arithmetic-intensive tasks, there are various choices, including exact schemes such as BGV [BGV12], BFV [Bra12, FV12], CLPX [CLPX18], GBFV [GV25, CHM+25], and dBFV [PZZ25], as well as approximate schemes like CKKS [CKKS17] and later high-precision adaptations like [LLL+21, CCKS23]. Boolean operations, in contrast, are typically supported through scheme-switching with substantial computation overhead [BGGJ20, LHH+21, LMP22, BCK+23].

In this context, a recent trend has emerged towards natively supporting both arithmetic and boolean operations on machine words (e.g., `int64`). This design point is particularly appealing because it closely resembles the hardware arithmetic logic unit (ALU) and aligns with real-world programming models. We broadly categorize these approaches into two groups; a concise summary is provided in Table 1.

Scalar Arithmetic and Boolean Schemes. This category supports arithmetic and boolean operations upon a single scalar machine word. Schemes such as DM [DM15] (with variants like LMKCDEY [LMK+23]) and CGGI [CGGI20] (also known as TFHE) natively support boolean operations by decomposing the scalar into bits (or digits) and encrypting one bit (or digit) per ciphertext. However, arithmetic operations like addition and multiplication are emulated through circuits. In their textbook multiplication circuit, the number of bootstrapping operations required is *quadratic* in the bit-width [BdBB+25].

More recently, REFHE [BFG+26] adopts the polynomial plaintext modulus from CLPX/GBFV in a BGV style. It uses a non-cyclotomic polynomial to directly induce the machine-word algebra inside one ciphertext, with the decomposed bits living in distinct polynomial coefficients. They support leveled arithmetic operations, while boolean operations are possible during *iterative* bootstrapping using the method from [KDE+24], with the number of bootstrapping required being *linear* in the bit-width.

These scalar schemes align perfectly with CPU-like computation models and are flexible and low-footprint for small amounts of data. However, their scalability to data-intensive workloads (e.g., machine learning, databases, analytics, and GPU-style vector processing) can be troublesome because they cannot exploit the amortization benefits of SIMD execution.

SIMD Arithmetic and Boolean Schemes. A variant of CKKS called *discrete* CKKS [DMPS24, BCKS24, BKSS24, AKP25] was recently proposed to support discrete data units (bits or small integers) and look-up table (LUT) over them through bootstrapping. Building upon this, later works advance to high-precision arithmetic. [BK25] focuses on arithmetic over arbitrary plaintext modulus using homomorphic residue number system, while not supporting boolean operations due to lack of exact reduction. For power-of-two machine words with bit-width n (up to 2048 bits), [Kim25b, Kim25c, CPL26] decompose them into k digits with $k = n/4$, and simulate carry propagation for each digit through bootstrapping, therefore supporting both arithmetic and boolean operations.

In more detail, [Kim25c] requires $O(\log k)$ CKKS bootstrapping for reducing the multiplication result to a redundant representation, while requiring $O(k)$

Table 1. Comparison of arithmetic and boolean FHE schemes.

Scheme	Type[†]	Arith[‡]	Bool	A→B	B→A
DM/CGGI	Scalar	$O(n)$ or $O(n^2)$	$O(n)$	N/A	N/A
REFHE	Scalar	Leveled w/ $O(n)$	$O(n)$	$O(n)$	Repack
[Kim25c]	SIMD	$O(\log k)$	Leveled w/ $O(1)$	$O(k)$	Free
[CPL26]	SIMD	$O(\log k)$	Leveled w/ $O(1)$	$O(1)$	Free
Our Scheme	SIMD	Leveled w/ $O(1)$	Leveled w/ $O(1)$	Amort. $O(1)$	Lightweight

[†] Scalar schemes count the number of gate bootstrapping (PBS); SIMD schemes count the number of CKKS bootstrapping.
[‡] "Leveled" stands for allowing leveled execution, and "w/" stands for the number of bootstrapping operations required for refreshing noise. Otherwise the notation is the required number of bootstrapping operations after *each* operation. k is the number of digits and usually $k = n/4$, with n being the bit-width of machine words.

CKKS bootstrapping to reduce to an exact representation; the state-of-the-art [CPL26] improves the construction to requiring $O(\log k)$ CKKS bootstrapping for exact representation by introducing lightweight carry propagation. They require the reduction process after *each multiplication* for carry management and cleaning the higher bits. To support boolean operations like bitwise logic, they require further bootstrapping to decompose digits into bits.

Focusing on the SIMD regime for data-intensive workloads, we conclude that the current constructions are *heavyweight* and *bootstrapping-intensive*. This raises the following question:

Is there a SIMD arithmetic and boolean FHE scheme with less bootstrapping?

in the sense that *leveled* operations are allowed before refreshing/conversion, and the number of CKKS bootstrapping per ciphertext can be amortized to *constant* if there is a sufficient amount of data (inherent reasons for amortization are explained in the technical overview section).

1.1 Our Contributions

We give an affirmative answer by constructing a CKKS-style FHE scheme with arithmetic and boolean modes, such that, from the cryptographical perspective,

1. Enables *several* leveled operations, either all in arithmetic or all in boolean, before a need for refreshing.
2. Takes a *constant* number of CKKS bootstrapping operations for refreshing inside each mode.
3. Supports *lightweight* conversion from boolean to arithmetic mode when the bit-width n is moderate.

4. Takes an *amortized constant* number of CKKS bootstrapping operations *per ciphertext* for conversion from arithmetic to boolean mode, by using $n/4$ CKKS bootstrapping operations to process $n/4$ ciphertexts.

From the application perspective, our scheme

1. Supports both *arithmetic and boolean* (e.g., bitwise logic, bit shift/rotation and comparison) operations, providing a programming interface most developers are familiar with.
2. Supports *machine-word integers* (e.g., 8-bit, 64-bit and 256-bit registers), matching the bit-width of commodity hardware.
3. Supports *SIMD*, matching the execution model of GPU and vector processors for data-intensive workloads.

We envision that our scheme is suitable for *massive* data processing (e.g., more than 16384 machine words) with the goal of *high-throughput* for workloads that have arithmetic and boolean operations on *moderate* bit-width (e.g., 64-bit) data. The comparison of our scheme with previous schemes can be checked in Table 1.

Our main technical novelties consist in

1. *Triangle encoding* that is a balance between arithmetic and boolean representations, enabling leveled arithmetic operations and easy management.
2. *Amortized constant* cost for *inherently* iterative arithmetic-to-boolean conversion, leveraging structural properties of the RLWE problem.
3. *SIMD structure* for the triangle encoding through grouping complex slots.

They are detailed in the technical overview section.

We emphasize that our scheme is an "exact FHE scheme" and achieves *exact correctness* (negligible failing probability) under application-aware security model (See Sec. 6.1 for more discussion on security notions) instead of *approximate correctness* even though we are CKKS-compatible, as we represent data in discrete form and can clearly separate noise from messages (Theorem 1).

We present *bootstrapping performance numbers* in Table 2. In batched setting, we outperform previous works for arithmetic refreshing and conversion to boolean mode; in single-ciphertext setting, we outperform previous works for arithmetic refreshing. Compared with scalar schemes like TFHE-rs and REFHE, we achieve significantly higher throughput. It is worth noting that our refreshing for 256-bit multiplication achieves a 7× lower latency than that of TFHE-rs.

We also provide an OpenFHE-based *open-source* prototype[3] for ease of reference and adoption.

1.2 Technical Overview

We have three design goals: (1) enabling both arithmetic and boolean operations, (2) amortized constant number of bootstrapping per ciphertext, and (3) SIMD.

We first explain the drastic difference between two kinds of operations.

[3] https://github.com/tsinghua-ideal/fhe-simd-alu

Table 2. Performance numbers of bootstrappings. We compare bootstrappings as we have different computation model from previous works, where we enable several operations before a bootstrapping. A2A refreshing enables further arithmetic operations. A2B conversion enables boolean operations after arithmetic operations. Experiments were conducted in a single-threaded CPU setting on the same machine with corresponding codebases. All parameters satisfy 128-bit security. "Latency" (Lat.) is the execution time of all bootstrapping operations, and "Amortized time" (Amort.) is obtained by dividing latency by the number of slots.

Scheme	**Dim**	**BitW**	**#Slots**	**Mult** (Latency)	**Bitwise** (Latency)
TFHE-rs	2^{11}	64	1	11.4s	209.9ms
		256	1	188.9s	876.6ms
REFHE	2^{12}	64	1	200ms	$\approx$ 3.83s (BTS)*
Scheme	**Dim**	**BitW**	**#Slots**	**A2A** (Lat./Amort.)	**A2B** (Lat./Amort.)
[Kim25c]	2^{16}	64	2048	36.5s / 17.8ms (4 BTS)	182s / 89ms (20 BTS)†
		256	512	47.6s / 92.9ms (5 BTS)	656s / 1.3s (69 BTS)†
[CPL26]	2^{16}	64	1024	23.5s / 22.9ms (2 BTS)	35.6s / 35ms (3 BTS)
		256	256	39.8s / 155ms (3 BTS)	66.5s / 260ms (5 BTS)
Our Scheme	2^{16}	64	1024	13.5s / 13.2ms (2 BTS)	80.5s / 79ms (16 BTS)
		256	256	19.5s / 76.0ms (2 BTS)	327s / 1.3s (64 BTS)
Scheme	**Dim**	**BitW**	**#Slots**	**Batched A2B** (Lat./Amort.)	
[Kim25b]	2^{15}	64	16384	206s / 12.6ms (46 BTS for 16 ciphertexts)	
Our Scheme	2^{16}	64	16384	128s / 7.8ms (16 BTS for 16 ciphertexts)	
		256		823s / 50ms (64 BTS for 64 ciphertexts)	

* Rough estimation from REFHE: "Complexity of 64-bit multiplication using TFHE is twice(thrice) the cost of *bootstrapping* in REFHE".
† The data for A2B (Reduce + ExactReduce) is estimated based on A2A (Reduce).

Two Extremes of the Spectrum. Arithmetic representations (friendly for multiplication) and boolean representations (for bitwise logic) are at two extremes of the plaintext algebra spectrum. The former prefers the number to be represented as a "whole" (either in coefficient domain or in frequency domain), while the latter prefers the number to be represented as "bit/digit vector". A conversion between them is inherently *iterative*, which should be able to either (1) assemble the bit/digit vector into a whole, or (2) decompose the whole into a bit/digit vector. Informally, the decomposition is harder than the assembly. Therefore "bit/digit vector" is the primary choice for existing FHE schemes that target both kinds of operations, as it is easier to decompose prior to encryption and simulate the "whole" arithmetic during homomorphic computation.

However, the choice of "bit/digit vector" will inevitably lead to the management of three objects during homomorphic arithmetic: (1) *carries*, in that the computation results of each bit/digit will affect other bits/digits, (2) *arithmetic overflows*, in that multiplying two 64-bit integers will produce a 128-bit integer

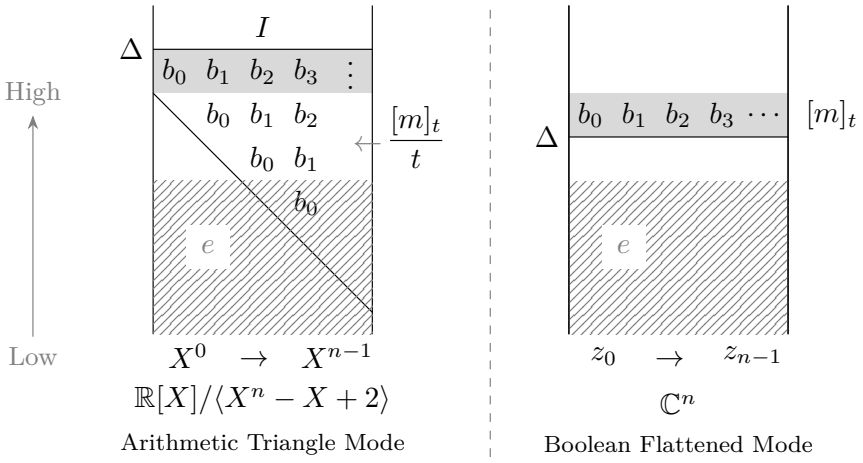


Fig. 1. (Left) The triangle shape of arithmetic encoding from Def. 10 with the $\frac{m}{2^n}$ term not shown. The upper side (gray region) contains essential bits to recover m. The noise e (shadow region) can contaminate part of the triangle. (Right) The flattened shape of boolean encoding, with each bit in each complex slot.

but the higher 64-bit is often required to be discarded, and (3) *noise*, which is inherent to the LWE problem.

Existing schemes need to handle some of them *iteratively*, which appears to be unavoidable in the "bit/digit vector" paradigm. DM/CGGI, [Kim25b], [Kim25c], and [CPL26] need to iteratively simulate carry propagation and leave spaces for arithmetic overflows. REFHE [BFG+26] uses algebraic constraints to propagate carries on the bit vector *at once*, but the noise and arithmetic overflows are closely coupled with the bit vector and require iterative refreshing process. By "at once", we mean a constant number of regular or bootstrapping operations.

Therefore, we drift away from the existing "bit/digit vector" paradigm by introducing the "triangle" encoding (Fig. 1) to mitigate the *iterative* nature using amortization. The encoding could be viewed as a middle ground of the "whole" paradigm and the "bit vector" paradigm. For a message $m \in \mathbb{Z}_{2^n}$, let $[m]_t = \sum_{i=0}^{n-1} b_i X^i$ be the *flattened* polynomial with binary coefficients where b_i is the i-th bit of m, we multiply the flattened polynomial by t^{-1} where $t = X - 2$ in an auxiliary ring to form a triangle: the i-th coefficient contains the lower $i+1$ bits of m. Note that the $(n-1)$-th coefficient is the "whole", and the upper side of the triangle is the "bit vector". If we consider other terms during computation, the general form of the triangle encoding is

$$\Delta I + \frac{\Delta}{X-2}[m]_t + e \qquad \text{in } \mathbb{R}[X]/\langle X^n - X + 2\rangle$$

where Δ is a CKKS-style scaling factor, arithmetic overflow I is a polynomial with integer coefficients, and e is the noise polynomial.

Thanks to the division-by-t and the multiplication of the scaling factor Δ, the arithmetic overflows and noise can be refreshed *at once* as they are represented in a coefficient-wise manner. We also note that, this form can be maintained across addition/multiplication, so this encoding allows *leveled* execution of arithmetic operations until a need for managing the overflow I and/or the noise e.

It is worth noting that while the noise e can contaminate part of the triangle, as long as its upper side is kept clean, we can clearly separate e from the message by a multiplication of $X - 2$ and we obtain *exact correctness*.

Now we discuss the boolean mode and the conversions. We first define the encoding in boolean mode to be "bit vector" (Fig. 1):

$$\Delta(b_0, b_1, \cdots, b_{n-1}) + e \qquad \text{in } \mathbb{C}^n$$

which is exactly discrete CKKS for bits. Bitwise logic and rotation can be naturally implemented as *leveled* CKKS addition/multiplication and rotation. The refreshing inside boolean mode is exactly one discrete CKKS bootstrapping. Note that the "bit vector" is exactly $[m]_t$, therefore a linear transformation and scalar multiplication gives us the "triangle" encoding, which is *lightweight* when the bit-width n is moderate.

Now we face the challenging conversion from the "triangle" mode to the "bit vector" mode. We admit that the conversion is still *iterative* with the presence of the arithmetic overflow I; indeed, such iterative nature appears to be an inherent barrier, as an efficient extraction of bits is equivalent to an efficient homomorphic modular reduction operation, which has been mitigated in various ways in previous literature (e.g., sine approximation). Instead, we crucially observe that it is possible to combine the "triangle" shape with the structural "limitation" of the underlying RLWE hardness problem to *amortize* the cost of such iterative process, which is detailed below.

Amortized Constant at Ciphertext Level. By "limitation" we are referring to the practical parameters of RLWE [ACC+18, BCC+24]. Its ring dimension N ranges from 2^{10} to 2^{17}, while machine words are often small like 64-bit. Therefore, the basic operational unit is a *ciphertext* and is substantially large.

One crucial way to amortize such overhead is through SIMD. We will discuss the *regular* SIMD packing later. Here we discuss a *special* SIMD during arithmetic-to-boolean conversion.

Our *novel* observation is that, each iteration only needs *partial* information from one machine word, therefore *multiple* ciphertexts can be *combined* into one ciphertext for one iteration, and the cost of multiple iterations can be amortized to become *constant*.

In more detail, we only process one "corner" of the triangle for each machine word in each iteration. After one iteration, we remove the corner (and corresponding bits in some other coefficients) from the triangle, resulting in a *smaller* triangle, so such iterative process can continue without complexity blowup.

SIMD Homomorphism Chain. Now we discuss how to enable regular SIMD packing. Previously, SIMD structure is closely coupled with the RLWE ring. As pronounced in schemes like BGV, BFV, CLPX, and GBFV, the available

plaintext algebra is directly related to the choice of the RLWE ring. For example, if we employ the implementation-friendly RLWE ring $\mathbb{Z}[X]/\langle X^N+1, Q\rangle$ with power-of-two cyclotomic polynomial, the available plaintext algebra $\mathbb{Z}_p$ is often limited to cases where p is a prime of special form.

To enable the machine-word algebra, REFHE deviates away from such regime by using a non-cyclotomic arithmetic ring $\mathbb{Z}[X]/\langle X^n - X + 2\rangle$. However, the problem of inducing SIMD structure *within* this ring remains *open*, and we do not intend to answer it here.

Instead, our key observation is to induce SIMD structure through complex slots. Such connection decouples the RLWE ring and the target arithmetic ring. In principle, any ring with a connection to the complex slots can be utilized, as long as the transformation does not "blow up the noise". For convenience and concreteness, we directly use the power-of-two RLWE ring as follows.

First, CKKS connects the power-of-two RLWE ring with $N/2$ numbers of complex slots:

$$\mathbb{Z}[X]/\langle X^N+1, Q\rangle \quad \leftarrow \quad \mathbb{R}[X]/\langle X^N+1\rangle \quad \cong \quad \mathbb{C}^{N/2}$$

The "$\leftarrow$" stands for rounding a polynomial with real coefficients into integer coefficients. We then note a homomorphism $\mathbb{C}^{N/2} = \left(\mathbb{C}^{n/2}\right)^{N/n}$ from NeuJeans and Homomorphic Integer Computer [JPK+24,Kim25c] that *groups* the complex slots. We therefore connect each machine-word algebra to each group of complex slots. New connections proposed by this work are highlighted in blue. The $\uparrow \Delta$ symbol denotes the arithmetic triangle encoding from flattened polynomial to the arithmetic ring $\mathbb{R}[X]/\langle X^n - X + 2\rangle$.

$$\begin{array}{lllll}
\mathbb{Z}[X]/\langle X^N+1, Q\rangle & & & & \\
\quad\uparrow & & & & \\
\mathbb{R}[X]/\langle X^N+1\rangle \;\cong\; \mathbb{C}^{N/2} \;=\; \left(\mathbb{C}^{n/2}\right)^{N/n} & \cong & (\,\mathbb{R}[X]/\langle X^n - X + 2\rangle\,)^{N/n} & & \\
 & & \quad\uparrow\Delta & & \\
 & & \mathbb{Z}_{2^n}^{N/n} & \cong & (\,\mathbb{Z}[X]/\langle X^n - X + 2, X - 2\rangle\,)^{N/n}
\end{array}$$

Fig. 2. The SIMD Homomorphism Chain

As indicated in the tail of the homomorphism chain, we get SIMD slots of machine-word algebra from one RLWE ring.

1.3 Sketch from an Application Perspective

This section intends to give a sketch of the execution flow, parameters, and practical implications of our scheme for concrete understanding.

Computation. First, concrete computation is conducted in the RLWE ring $\mathbb{Z}[X]/\langle X^N+1, Q\rangle$. All our triangle constructions, rings, and homomorphism chain can be understood as *encoding*, that happens prior to the encryption. All homomorphic operations are built upon prior CKKS operations and existing hardware/software acceleration efforts for CKKS can be used for this scheme. Appendix C gives the whole picture of operations needed and offered by our scheme.

Execution Flow. We have two major user-facing ciphertext formats: arithmetic mode and boolean mode, decided at encoding time. Ciphertext in each mode supports *leveled* execution until we are required to *refresh* it, or *convert* it to another mode.

Packing. There are full packing and sparse packing in our scheme, in the same sense as in the CKKS literature. We mainly discuss full packing in this work, but similar algebra applies for sparse packing (Appendix A.1). For full packing, while one ciphertext in arithmetic mode can pack N/n machine words, the boolean mode can only pack $N/(2n)$ machine words, so arithmetic-to-boolean conversion for one ciphertext will produce two ciphertexts, each containing lower and upper halves of the machine words (Fig. 4). These boolean-mode ciphertexts can be combined into one CKKS ciphertext with the upper halves in the imaginary part for bootstrapping.

"Leveled". Unlike previous schemes where the number of available levels is more related to the number of RNS moduli, the number of levels in our scheme depends on the noise. Roughly speaking, for the scaling factor $\log \Delta \approx 43$ in our example parameter, we allow roughly 33 bits of noise for bootstrapping to function. We start from the bottom noise of roughly 20 bits, and each level of multiplication will incur 2 to 6 bits of noise growth. Therefore, the example parameter supports a multiplication circuit of depth 2 to 3 for full multiplication (Def. 14), or 4 to 5 for short multiplication (e.g., plaintexts, fresh ciphertexts, ciphertexts converted from boolean mode). We comment that selecting a larger Δ during parameter generation could allow deeper multiplication. Dynamic scaling factor technique [CCK+25] could also enable deeper multiplication.

For CKKS level consumption, the full multiplication in our scheme takes *two* CKKS levels, though it can be fused with other multiplication to reduce level consumption (e.g., two consecutive full multiplication of three ciphertexts can be fused into multiplication of three ciphertexts and a multiplication with Δt^2, instead of multiplying by Δt each time). The short multiplication takes one CKKS level.

As the behavior of "leveled" is complex in our scheme, we consider software control/compiler for this scheme to be helpful future work (also for security concerns, see Sec. 6.1).

2 Preliminaries

Notations. We use normal font m for an element in a ring like $m \in \mathcal{R}_Q$, $m \in \mathcal{Z}_t$, $m \in \mathbb{Z}_{2^n}$ and $m \in \mathbb{C}$. We will state clearly which ring it is in. When the ring is a

polynomial ring, we may also write it as $m(X)$ when we want to emphasize the role of X. For taking the i-th coefficient of the polynomial, we use the notation m_i and define $m(X) = \sum_{i=0}^{N-1} m_i X^i$.

For a vector of elements, we use bold font like $\boldsymbol{m}$. One example is $\boldsymbol{m} = (m_0, m_1, \ldots, m_{N-1}) \in \mathbb{C}^N$. Vectors are column vectors. We may also sometimes identify a polynomial $m(X)$ as a vector $\boldsymbol{m}$. We use bold capital letter for matrices like $\boldsymbol{A} \in \mathbb{C}^{M \times N}$, and let $A_{i,j}$ be the element in the i-th row and j-th column, for $0 \le i < M$ and $0 \le j < N$. We use $(\cdot\,;\cdot)$ as vertical concatenation of two matrices, namely $(\boldsymbol{A}\,;\boldsymbol{B}) = \begin{bmatrix}\boldsymbol{A}\\ \boldsymbol{B}\end{bmatrix}$.

We use infinity norm $\|\cdot\|_\infty$ for the norm operator $\|\cdot\|$, either on polynomials, vectors, or matrices. We use base 2 for taking logarithm $\log(\cdot)$.

2.1 Rings and their relations

Here we introduce all essential ring notations used in this work. More relations and explicit expressions can be checked in Appendix A.1.

RLWE Ring $\mathcal{R}_Q$. We choose a power-of-two number N. It is the ring dimension for the RLWE ring $\mathcal{R} = \mathbb{Z}[X]/\langle X^N + 1\rangle$. Q is a number related to the RLWE problem. In usual deployment, Q can be decomposed into smaller primes like $Q = Q_L = q_0 q_1 \cdots q_L$. We use the notation $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R} = \mathbb{Z}[X]/\langle X^N + 1, Q\rangle$ and $\mathcal{R}_\mathbb{R} = \mathbb{R}[X]/\langle X^N + 1\rangle$. For concreteness, one can think of $N = 2^{16}$ and $Q \approx 2^{1200}$.

Arithmetic Ring $\mathcal{Z}_\mathbb{R}$. We choose a power-of-two number n. It is the ring dimension for the ring $\mathcal{Z} = \mathbb{Z}[X]/\langle X^n - X + 2\rangle$. We denote the polynomial modulus $t = X - 2$ in $\mathcal{Z}$. We use the notation $\mathcal{Z}_t = \mathbb{Z}[X]/\langle X^n - X + 2, X - 2\rangle$ and $\mathcal{Z}_\mathbb{R} = \mathbb{R}[X]/\langle X^n - X + 2\rangle$. We call $\mathcal{Z}_\mathbb{R}$ the arithmetic ring and it is the primary ring we are working on. For concreteness, one can think of $n = 64$.

Connection to $\mathbb{C}$. For the RLWE ring $\mathcal{R}_\mathbb{R}$, $X^N + 1$ is the $(2N)$-th cyclotomic polynomial. Let $\zeta = \exp\left(2\pi\sqrt{-1}/(2N)\right)$, CKKS chooses $N/2$ primitive roots of unity as $\{\zeta_j : 0 \le j < N/2\}$ where $\zeta_j = \zeta^{5^j}$. Note that they together with their conjugates $\{\overline{\zeta_j} : 0 \le j < N/2\}$ form all the roots for $X^N + 1$ in $\mathbb{C}$.

Therefore, we define the *canonical embedding* $\sigma : \mathcal{R}_\mathbb{R} \to \mathbb{C}^{N/2}$ that $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_{N/2-1})$ where $\sigma_j : \mathcal{R}_\mathbb{R} \to \mathbb{C}$ is defined as

$$\sigma_j(m(X)) = m(\zeta_j) \qquad , 0 \le j < N/2$$

Each σ_j is a ring homomorphism and σ is a ring isomorphism.

For the arithmetic ring $\mathcal{Z}_\mathbb{R}$, $X^n - X + 2$ has n complex roots for $n \ge 2$ and n being a power-of-two, as $X^n - X + 2$ is irreducible over $\mathbb{Q}$. We pick $\{\xi_j : 0 \le j < n/2\}$ such that ξ_j are in the upper half of the complex plane. They are numerically solved using companion matrix. We make an arbitrary choice to sort them using their real part, with the largest (closest to 1) being ξ_0. Similarly, we can define the *$\mathcal{Z}$-canonical embedding* $\tau : \mathcal{Z}_\mathbb{R} \to \mathbb{C}^{n/2}$ that $\tau = (\tau_0, \tau_1, \ldots, \tau_{n/2-1})$ where $\tau_j : \mathcal{Z}_\mathbb{R} \to \mathbb{C}$ is defined as

$$\tau_j(m(X)) = m(\xi_j) \qquad , 0 \le j < n/2$$

Note that τ is a ring isomorphism.

Proposition 1 (Norms of σ and τ). *The operator norm of the canonical embedding σ satisfies $\|\sigma\| \sim \frac{2}{\pi}N$. The operator norm of the $\mathcal{Z}$-canonical embedding τ satisfies $\|\tau\| \sim \frac{2}{\ln 3}n$.*

When $N = n$, the norm behavior of σ and τ only differs by a small constant factor, thanks to the small coefficients of $X^n - X + 2$. Therefore, σ and τ shows similar noise behavior. Similar conclusion holds for σ^{-1} and τ^{-1}. The norm of σ could be found in [CCH+24, Corollary 1], and the norm of τ can be derived using similar techniques. The detailed proof is provided in Appendix A.1.

Isomorphisms in matrix form. It will be convenient to express the isomorphisms $\sigma : \mathcal{R}_{\mathbb{R}} \to \mathbb{C}^{N/2}$ and $\tau : \mathcal{Z}_{\mathbb{R}} \to \mathbb{C}^{n/2}$ as linear transformations in matrix form so we can find its inverse and evaluate them *homomorphically*.

In the RLWE ring $\mathcal{R}_{\mathbb{R}}$, fix a polynomial $m(X)$ and identify it as a vector $\boldsymbol{m} = (m_0, m_1, \ldots, m_{N-1}) \in \mathbb{R}^N$, σ can be understood as an $(N/2) \times N$ Vandermonde matrix $\boldsymbol{U}_\sigma$ such that $\boldsymbol{z} = \boldsymbol{U}_\sigma \boldsymbol{m}$ with $\boldsymbol{z} \in \mathbb{C}^{N/2}$. Consider the $N \times N$ matrix $\mathrm{CRT}_\sigma = \left(\boldsymbol{U}_\sigma \,;\, \overline{\boldsymbol{U}_\sigma}\right)$, its inverse is $\mathrm{CRT}_\sigma^{-1} = \frac{1}{N}\overline{\mathrm{CRT}_\sigma}^T$. We have $(\boldsymbol{z} \,;\, \overline{\boldsymbol{z}}) = \mathrm{CRT}_\sigma \cdot \boldsymbol{m}$, therefore the inverse transform can be evaluated using $\frac{1}{N}\overline{\boldsymbol{U}_\sigma}^T$. We can further divide $\boldsymbol{U}_\sigma$ into two $(N/2) \times (N/2)$ square matrices $\boldsymbol{U}_\sigma = \begin{bmatrix}\boldsymbol{U}_{\sigma,0} & \boldsymbol{U}_{\sigma,1}\end{bmatrix}$. Note that $\boldsymbol{U}_{\sigma,1} = \sqrt{-1} \cdot \boldsymbol{U}_{\sigma,0}$, therefore we can use $\boldsymbol{U}_{\sigma,0}$ for evaluation. We use the notation $\boldsymbol{V}_{\sigma,0} = \overline{\boldsymbol{U}_{\sigma,0}}^T$.

In the arithmetic ring $\mathcal{Z}_{\mathbb{R}}$, fix a polynomial $m(X)$ and identify it as a vector $\boldsymbol{m} = (m_0, m_1, \ldots, m_{n-1})$. τ can be understood as an $(n/2) \times n$ Vandermonde matrix $\boldsymbol{U}_\tau$ such that $\boldsymbol{z} = \boldsymbol{U}_\tau \boldsymbol{m}$ with $\boldsymbol{z} \in \mathbb{C}^{n/2}$.

For $\mathrm{CRT}_\tau = \left(\boldsymbol{U}_\tau \,;\, \overline{\boldsymbol{U}_\tau}\right)$, we denote $\tau^{-1} = \mathrm{CRT}_\tau^{-1}$. Vandermonde matrix theory tells us that the k-th column of the inverse τ^{-1} has the coefficients of the k-th Lagrange polynomial. Let $\ell_k(X) = \sum_{i=0}^{n-1} \ell_{k,i} \cdot X^i$, then τ^{-1} can be expressed as $\tau^{-1} = \begin{bmatrix}\boldsymbol{V}_\tau & \overline{\boldsymbol{V}_\tau}\end{bmatrix}$ where $\boldsymbol{V}_\tau$ is a $n \times (n/2)$ matrix. We further divide them into $(n/2) \times (n/2)$ square matrices $\boldsymbol{U}_\tau = \begin{bmatrix}\boldsymbol{U}_{\tau,0} & \boldsymbol{U}_{\tau,1}\end{bmatrix}$ and $\boldsymbol{V}_\tau = (\boldsymbol{V}_{\tau,0} \,;\, \boldsymbol{V}_{\tau,1})$.

2.2 The RLWE problem and its operations

We state a definition of the RLWE problem used in production for exposition.

Definition 1 (The RLWE problem). *Let N be a power-of-two number and Q be a number, χ_s and χ_e be the secret key and noise distribution. In the ring $\mathcal{R}_Q = \mathbb{Z}[X]/\langle X^N + 1, Q\rangle$, we sample a uniformly random $a \in \mathcal{R}_Q$, a secret key s from χ_s and a noise e from χ_e, let $b = -a \cdot s + e$, we return (b, a).*

Search RLWE problem consists in finding such secret key s from (b, a). Decision RLWE problem consists in distinguishing between (b, a) from a uniformly random sample $(b', a') \in \mathcal{R}_Q^2$.

With the RLWE problem, we are able to encrypt a message $m \in \mathcal{R}_Q$. We state the secret-key encryption for exposition. It can be easily adapted to the public-key version (e.g., [KPZ21]).

Definition 2 (RLWE Encryption/Decryption). *Given $m \in \mathcal{R}_Q$ and a secret key s, we can sample a uniformly random $a \in \mathcal{R}_Q$ and a noise e from χ_e and produce a ciphertext*

$$\begin{aligned} \mathsf{ct} = \mathsf{RLWE.Encrypt}_s(m) &= (-a \cdot s + m + e, a) \quad \in \mathcal{R}_Q^2 \\ \mathsf{RLWE.Decrypt}_s(\mathsf{ct}) &= \langle \mathsf{ct}, (1, s) \rangle = m + e + QI \equiv m + e \pmod{Q} \end{aligned}$$

where I is a polynomial with integer coefficients. The exact decoding procedure is scheme-specific.

We now highlight the shape of I. [4]

Proposition 2 (Distribution of the overflow I in RLWE, [BTH22]). *The I in the decryption formula follows Irwin-Hall distribution parameterized by the secret key distribution χ_s.*

For sparse ternary secret key of Hamming weight 32, we have $\|I\|_\infty < 16$ with negligible failing probability.

We remark that the secret key of Hamming weight 32 is not our main secret key; it is the *encapsulated* key only for ModRaise as defined below.

While in normal cases the overflow is not present in the decrypted plaintext, it is present in the high bits after a modulus raising operation that regains level.

Definition 3 (RLWE Modulus Raising). *Given $\mathsf{ct} = (b, a) = (-a \cdot s + m + e, a) \in \mathcal{R}_{Q_{L-1}}^2$ with $m \in \mathcal{R}_{Q_{L-1}}$. Suppose $Q_L = Q_{L-1} q_L$, we can raise from $\mathcal{R}_{Q_{L-1}}$ to $\mathcal{R}_{Q_L}$ by the following operation*

$$\mathsf{ct}' = \mathsf{RLWE.ModRaise}_{q_L}(\mathsf{ct}) = (b, a) \in \mathcal{R}_{Q_L}^2$$

which is directly putting elements from the smaller ring to the larger ring.

The effect on the message is

$$\mathsf{RLWE.Decrypt}_s(\mathsf{ct}') \equiv m + e + IQ_{L-1} \pmod{Q_L}$$

when $\|I\|_\infty \le q_L$. By using the encapsulated sparse key with small Hamming weight, the $\|I\|$ after ModRaise is less than 16, and this is used to achieve negligible failing probability for CKKS bootstrapping [BTH22].

Conversely, it is possible to use the modulo-by-Q structure to *truncate* the high bits of the message [KN25].

Definition 4 (RLWE Truncation). *Given $\mathsf{ct} = (b, a) = (-as + \Delta I + \Delta m + e, a) \in \mathcal{R}_{Q_\ell}^2$ with I being an integer polynomial and $\|m\| < 1/2$, the truncating operation is defined as*

$$\begin{aligned} \mathsf{ct}' = \mathsf{RLWE.Truncate}(\mathsf{ct}) &= \mathsf{RLWE.Rescale}_{Q_\ell \to q_0}(\mathsf{RLWE.MultScalar}(\mathsf{ct}, Q_\ell/\Delta)) \\ \mathsf{RLWE.Decrypt}_s(\mathsf{ct}') &\approx q_0 m + \frac{q_0}{\Delta} e \in \mathcal{R}_{q_0} \end{aligned}$$

[4] We will talk about another "arithmetic overflow" I in this paper. We deliberately use the same symbol as they are interchangeable after A2A-I.

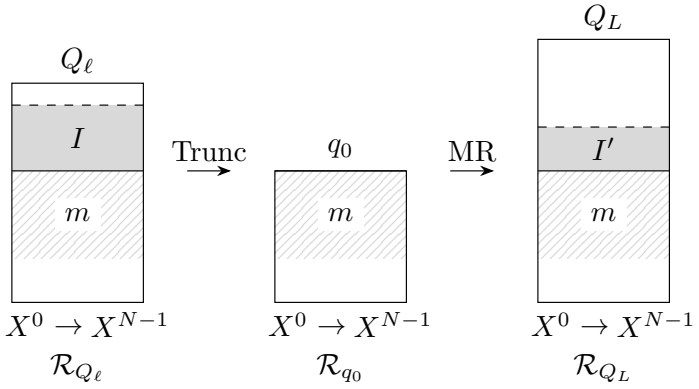


Fig. 3. The RLWE.Truncate and RLWE.ModRaise operations. The input I is truncated and replaced with an I' with possibly smaller norm. The message m is kept. The available modulus is regained to Q_L.

Fig. 3 shows that the combination of truncation and modulus raising can reset the high part while regaining modulus, and will become a core subroutine in our construction. Other common RLWE operations like Add are defined in the Appendix A.2.

2.3 Approximate CKKS

Aside from the arithmetic triangle mode and boolean flattened mode as our main target encoding discussed in the next section, we sometimes need to *reinterpret* the data in $\mathbb{C}^{N/2}$ as approximate CKKS data.

Definition 5 (CKKS Encoding). *For a message $z \in \mathbb{C}$, fix a scaling factor Δ, its encoding in a complex slot $\mathbb{C}$ in CKKS is*

$$\mathsf{CKKS.Ecd}_{\Delta,\mathbb{C}}(z) = \Delta z$$

Given a vector of messages $\boldsymbol{z} = (z_0, z_1, \ldots, z_{N/2-1}) \in \mathbb{C}^{N/2}$, fix a scaling factor Δ, there exists a polynomial $m(X) \in \mathcal{R}_Q = \mathbb{Z}[X]/\langle X^N + 1, Q\rangle$ such that

$$\begin{aligned} m(X) &= \mathsf{CKKS.Ecd}_{\Delta,\mathcal{R}_Q}(\boldsymbol{z}) \\ m(\zeta_j) &\approx \Delta z_j = \mathsf{CKKS.Ecd}_{\Delta,\mathbb{C}}(z_j) \qquad \forall 0 \le j < N/2 \end{aligned}$$

Such $m(X)$ can be constructed using the inverse transform σ^{-1} subject to the condition that $\left\|\sigma^{-1}(\Delta \boldsymbol{z})\right\| < Q/2$.

We often recognize CKKS encoded data with noise $\boldsymbol{e}$ as $\Delta\boldsymbol{z}+\boldsymbol{e}$. The addition and multiplication of $\mathcal{R}_Q$ give addition and multiplication for complex slots, respectively, through the canonical embedding isomorphism. Therefore CKKS.Sine function can be defined using polynomial approximation (Def. 26) [CHK+18].

Using automorphisms in $\mathcal{R}$, operations like CKKS.Rotate and CKKS.Conjugate (Def. 25) can be constructed. With them, linear transformation can be efficiently evaluated to move data between $\mathcal{R}_Q$ and $\mathbb{C}$:

Definition 6 ($\mathcal{R}$-To-$\mathbb{C}$ and $\mathbb{C}$-To-$\mathcal{R}$). *Given a polynomial $m(X) \in \mathcal{R}_Q$ that is represented as $\boldsymbol{m} = (m_0, m_1, \ldots, m_{N-1}) \in \mathbb{R}^N$. Let $\boldsymbol{m} = (\boldsymbol{m}_0 \,;\, \boldsymbol{m}_1)$ such that $\boldsymbol{m}_0 = (m_0, m_1, \ldots, m_{N/2-1})$ and $\boldsymbol{m}_1 = (m_{N/2}, \ldots, m_{N-1})$ we can produce two $w_0(X)$ and $w_1(X)$ that $w_0(X) = \mathsf{CKKS.Ecd}_{\Delta,\mathcal{R}_Q}(\boldsymbol{m}_0)$ and $w_1(X) = \mathsf{CKKS.Ecd}_{\Delta,\mathcal{R}_Q}(\boldsymbol{m}_1)$ using $\boldsymbol{V}_{\sigma,0}$ and CKKS.Conjugate. This is called $\mathcal{R}$-To-$\mathbb{C}$.*

$$w_0(X) = \frac{2}{N}\Re\left[\mathsf{CKKS.LT}_{\boldsymbol{V}_{\sigma,0}}(m(X))\right]$$
$$w_1(X) = \frac{2}{N}\Im\left[\mathsf{CKKS.LT}_{\boldsymbol{V}_{\sigma,0}}(m(X))\right]$$

Then by using $w_0(X)$ and $w_1(X)$, we can restore back $m(X)$ using $\boldsymbol{U}_{\sigma,0}$. This is called $\mathbb{C}$-To-$\mathcal{R}$.

$$m(X) = \mathsf{CKKS.LT}_{\boldsymbol{U}_{\sigma,0}}(w_0(X) + X^{N/2} w_1(X))$$

2.4 Discrete CKKS

Discrete CKKS requires the messages in the complex slots to have discrete shape [BKSS24, AKP25].

Definition 7 (Discrete CKKS Encoding). *For a message $m \in \mathbb{Z}_p$ where p is a power-of-two, fix a scaling factor Δ, its encoding in a complex slot $\mathbb{C}$ in discrete CKKS is*

$$\mathsf{D\text{-}CKKS.Ecd}_{\Delta,\mathbb{C},p}(m) = \Delta\frac{m}{p}$$

Note that m is actually stored in the fractions in the view point of Δ. The general form of discrete CKKS encoded data is $\Delta I + \Delta\frac{m}{p} + e \in \mathbb{C}$ where I is a small integer and e is the noise.

Discrete CKKS allows look-up table LUT evaluation on the m while also cleaning I and e.

Definition 8 (Discrete CKKS LUT Operation). *Given $\Delta I + \Delta\frac{m}{p} + e \in \mathbb{C}$, for arbitrary LUT function $f : \mathbb{Z}_p \to \mathbb{Z}_p$, there exists an operation (parameterized by interpolation order κ) that evaluates the LUT while cleaning the I and e when $|I| < K$ and $\frac{ep}{\Delta}$ is small enough.*

$$\mathsf{D\text{-}CKKS.LUT}_f\left(\Delta I + \Delta\frac{m}{p} + e\right) = \Delta\frac{f(m)}{p} + e' + e''$$

where e' is the noise after noise cleaning with $e' = \frac{\Delta}{p} \cdot O\left(\left|\frac{ep}{\Delta}\right|^{\kappa+1}\right)$, and e'' is the noise caused by homomorphic operations.

We remark that, D-CKKS.LUT supports simultaneous evaluation of multiple LUTs with low costs and this is called *multi-value* LUT evaluation.

3 Arithmetic Triangle Mode

We follow the idea of Flatten operation from [GV25], which generalizes the modulo operation $[\cdot]_t$ from t being an integer to t being a polynomial. Here we fix $t = X-2$. While they present the operation as modulo-by-1, we have a top-down narrative:

Definition 9 (The Flatten operation). *For a (positive) $m \in \mathbb{Z}$, its binary representation is $m = \sum_{i=0}^{\infty} b_i 2^i$ where $b_i \in \{0,1\}$. It can be viewed as a polynomial $\sum_{i=0}^{\infty} b_i X^i$ evaluated at $X = 2$ (An isomorphism between $\mathbb{Z}$ and $\mathbb{Z}[X]/\langle X-2\rangle$). If we further want to modulo the high bits, by introducing the relation $X^n = X-2 = t$ that maps the high bits into the ideal $\langle X-2\rangle$, we can modulo the ideal to get $\mathcal{Z}_t = \mathbb{Z}[X]/\langle X^n - X + 2, X - 2\rangle$ and define*

$$\begin{array}{rccc} [\cdot]_t : & \mathbb{Z} & \to & \mathcal{Z}_t \\ & m & \mapsto & \sum_{i=0}^{n-1} b_i X^i \end{array}$$

Note that the ideal $\langle 2^n\rangle$ in $\mathbb{Z}$ is mapped to the ideal $\langle X-2\rangle$ in $\mathcal{Z} = \mathbb{Z}[X]/\langle X^n - X + 2\rangle$, so $[\cdot]_t$ becomes a ring isomorphism between $\mathbb{Z}_{2^n} = \mathbb{Z}/\langle 2^n\rangle$ and $\mathcal{Z}_t$. More specifically, for $m_1, m_2 \in \mathbb{Z}_{2^n}$ and $[m_1]_t, [m_2]_t \in \mathbb{Z}[X]$, we have

$$\begin{aligned} [m_1]_t + [m_2]_t &= [m_1 + m_2]_t + tI \equiv [m_1 + m_2]_t \in \mathcal{Z}_t \\ [m_1]_t \cdot [m_2]_t &= [m_1 \cdot m_2]_t + tI' + X^n C \equiv [m_1 \cdot m_2]_t \in \mathcal{Z}_t \end{aligned} \tag{1}$$

where I, I' and C are some polynomials in $\mathbb{Z}[X]$. $a \equiv b \in \mathcal{Z}_t$ means the two sides are equal after modulo by $X^n - X + 2$ and $X - 2$. Note that after modulo by $X^n - X + 2$, the $X^n C$ term is be turned into an tC term, and further modulo by t gives 0. Therefore, addition and multiplication in $\mathbb{Z}_{2^n}$ can be carried out in $\mathcal{Z}_t$ if we are able to maintain $\mathcal{Z}_t$ somewhere inside $\mathcal{Z}_{\mathbb{R}} = \mathbb{R}[X]/\langle X^n - X + 2\rangle$.

In RLWE settings, polynomials are noisy. We follow the idea of CKKS by setting a scaling factor Δ so the resulting $\Delta[m]_t$ can be separated from noise. We now have $\Delta[m]_t$ in $\mathcal{Z}_{\mathbb{R}}$ and the overflow from addition and multiplication is in the ideal $\langle\Delta(X-2)\rangle$.

On the other hand, the $\langle\Delta(X-2)\rangle$ ideal is hard to deal with. REFHE introduces modulo-by-ideal operations. We instead follow the idea of GBFV by multiplying by $\frac{1}{t}$ so the $\langle\Delta(X-2)\rangle$ ideal is turned into an integer ideal $\langle\Delta\rangle$. This gives us $\frac{\Delta}{t}[m]_t \in \mathcal{Z}_{\mathbb{R}}$.

Definition 10 (Encoding in $\mathcal{Z}_{\mathbb{R}}$). *For a message $m \in \mathbb{Z}_{2^n}$, fix a scaling factor Δ, the encoding of m in $\mathcal{Z}_{\mathbb{R}}$ is*

$$\mathcal{Z}_{\mathbb{R}}.\mathsf{Ecd}_{\Delta,t}(m) = \frac{\Delta}{t}[m]_t = \Delta\left(\frac{m}{2^n} - \sum_{k=0}^{n-1} \frac{X^k}{2^{k+1}} \underset{0\le i\le k}{[m]}\right)$$

where $\underset{0\le i\le k}{[m]} = \sum_{i=0}^{k} b_i 2^i$ is the lower $k+1$ bits of m.

As a pictorial illustration, the encoded polynomial looks like a triangle when the $\frac{m}{2^n}$ term in the constant coefficient is not drawn (Fig. 1). We remark that the "essential" bits are in the upper side of the triangle, and noise can contaminate lower part of the triangle.

Due to the extra overflow term shown in Eq. 1, the encoded polynomial after arithmetic operations may not be exactly $\frac{\Delta}{t}[m]_t$ but contains an extra term ΔI where I is a polynomial with integer coefficients. Therefore, we generally recognize the triangle encoding to be of the following form.

Definition 11 (General Form of the Triangle Encoding). *The general form of the triangle encoding in $\mathcal{Z}_{\mathbb{R}}$ is*

$$f = \Delta I + \frac{\Delta}{t}[m]_t + e$$

where I is a polynomial with integer coefficients and e is small noise.

Definition 12 (Decoding in $\mathcal{Z}_{\mathbb{R}}$). *Given a polynomial $f = \Delta I + \frac{\Delta}{t}[m]_t + e$ in $\mathcal{Z}_{\mathbb{R}}$, we define the decoding function to be*

$$\mathcal{Z}_{\mathbb{R}}.\mathsf{Dcd}_{\Delta,t}(f) = \left\lfloor \frac{t}{\Delta} f \right\rceil (2) \pmod{2^n}$$

with $\frac{t}{\Delta} f$ happening in $\mathcal{Z}_{\mathbb{R}}$ and "(2)" denoting evaluation at $X = 2$ in $\mathcal{Z}_{\mathbb{R}}$.

We emphasize that the rounding happens before polynomial evaluation as otherwise the noise will amplify through the evaluation.

Theorem 1 (Arithmetic Decoding Correctness). *We can correctly decode if $m \in \mathbb{Z}_{2^n}$, I is a polynomial with integer coefficients, and $\|te\|_\infty < \frac{\Delta}{2}$*

Proof. $\frac{t}{\Delta} f = tI + [m]_t + \frac{t}{\Delta} e$. *Since $\|te\|_\infty < \frac{\Delta}{2}$ and I contains integer coefficients, the rounding operation gives $tI + [m]_t$. When evaluated in $\mathcal{Z}_{\mathbb{R}}$, $[m]_t(2) = m$ and $tI(2)$ is 0.*

We remark that for bootstrapping to function correctly, the requirement on the magnitude of the noise is stricter (Theorem 3, 5, and 6).

As long as the noise exceeds the bounds with negligible probability, the decoding fails with negligible probability. Moreover, the CKKS bootstrapping failing probability is also negligible (Prop. 2 and Def. 3). Therefore, we achieve *exact correctness* by appropriate use of software control on the noise (See Sec. 6.1 for more discussion).

Now we discuss the arithmetic of encoded polynomial in $\mathcal{Z}$.

Definition 13 (Addition in $\mathcal{Z}_{\mathbb{R}}$ is Addition in $\mathbb{Z}_{2^n}$). *For $f, f' \in \mathcal{Z}_{\mathbb{R}}$ with $f = \Delta I + \frac{\Delta}{t}[m]_t + e$ and $f' = \Delta I' + \frac{\Delta}{t}[m']_t + e'$ with $m, m' \in \mathbb{Z}_{2^n}$, we define the addition operation to be*

$$\mathcal{Z}_{\mathbb{R}}.\mathsf{Add}(f, f') = f + f' = \Delta I'' + \frac{\Delta}{t}[m + m']_t + e + e'$$

with $I'' = I + I' + C$ where C has integer coefficients being carries propagated from $[m]_t + [m']_t = [m + m']_t + tC$.

Definition 14 (Multiplication in $\mathcal{Z}_{\mathbb{R}}$ is Multiplication in $\mathbb{Z}_{2^n}$). *For $f, f' \in \mathcal{Z}_{\mathbb{R}}$ with $f = \Delta I + \frac{\Delta}{t}[m]_t + e$ and $f' = \Delta I' + \frac{\Delta}{t}[m']_t + e'$ with $m, m' \in \mathbb{Z}_{2^n}$, we define the multiplication operation to be*

$$\mathcal{Z}_{\mathbb{R}}.\mathsf{Mult}(f, f') = \frac{1}{\Delta^2} \cdot \Delta t \cdot f \cdot f' = \Delta I'' + \frac{\Delta}{t}[m \cdot m']_t + e''$$

Note that this closely resembles the BFV multiplication. We rescale twice to divide by Δ^2 as the constant $\frac{t}{\Delta}$ is not representable with small noise in $\mathcal{Z}_{\mathbb{R}}$, nor is t. Therefore, one full multiplication consumes two CKKS levels. We remark that full multiplications can be fused together and the constant will be Δt^2 or Δt^3, consuming less levels.

Definition 15 (Multiplication Shortcut in $\mathcal{Z}$). *If we have $g = \Delta[m']_t + e'$ in $\mathcal{Z}_{\mathbb{R}}$, the multiplication can be simplified to*

$$\mathcal{Z}_{\mathbb{R}}.\mathsf{MultShort}(f, g) = \frac{1}{\Delta} \cdot f \cdot g = \Delta I'' + \frac{\Delta}{t}[m \cdot m']_t + e''$$

Note that this consumes only one level.

4 The SIMD Homomorphism Chain

We now discuss how to embed $\mathcal{Z}_{\mathbb{R}}$ in $\mathcal{R}_Q$. Rewriting the SIMD homomorphism chain in Fig. 2, we have

$$\mathcal{R}_Q \quad \leftarrow \quad \mathcal{R}_{\mathbb{R}} \quad \cong \quad \mathbb{C}^{N/2} \quad = \quad (\mathbb{C}^{n/2})^{N/n} \quad \cong \quad \mathcal{Z}_{\mathbb{R}}^{N/n} \quad \xleftarrow{\Delta} \quad \mathcal{Z}_t^{N/n}$$

We first demonstrate the case of $N/n = 1$.

Definition 16 (Embedding of $\mathcal{Z}_{\mathbb{R}}$ in $\mathcal{R}_{\mathbb{R}}$). *Given a polynomial $f(X) \in \mathcal{Z}_{\mathbb{R}}$, its embedding in $\mathcal{R}_{\mathbb{R}}$ is*

$$\mathcal{R}_{\mathbb{R}}.\mathsf{Embed}_{\mathcal{Z}_{\mathbb{R}}}(f(X)) = \sigma^{-1}(\tau(f(X)))$$
$$\mathcal{Z}_{\mathbb{R}}.\mathsf{Recover}_{\mathcal{R}_{\mathbb{R}}}(g(X)) = \tau^{-1}(\sigma(g(X)))$$

The rounding operation from $\mathcal{R}_{\mathbb{R}}$ to $\mathcal{R}$ essentially adds rounding error $e \in \mathcal{R}_{\mathbb{R}}$ with each coefficient in $(-1/2, 1/2]$. The correctness of embedding a polynomial from $\mathcal{R}$ in $\mathcal{R}_Q$ requires the coefficients of the polynomial to be smaller than $Q/2$.

Theorem 2 (Correctness of embedding $\mathcal{Z}_{\mathbb{R}}$ in $\mathcal{R}_Q$). *Given a polynomial $f(X) \in \mathcal{Z}_{\mathbb{R}}$ with $\|f\|_\infty = B$, its embedding in $\mathcal{R}_Q$ is*

$$g(X) = \mathcal{R}_Q.\mathsf{Embed}_{\mathcal{Z}_{\mathbb{R}}}(f(X)) = \left[\lfloor \sigma^{-1}(\tau(f(X))) \rceil\right]_Q$$
$$h(X) = \mathcal{Z}_{\mathbb{R}}.\mathsf{Recover}_{\mathcal{R}_{\mathbb{R}}}(g(X)) = \tau^{-1}(\sigma(g(X)))$$

we have $h(X) \approx f(X)$ with small error if $\left\|\sigma^{-1}\right\| \cdot \|\tau\| \cdot B < Q/2$. Let the rounding error be e, the small error is $\tau^{-1}(\sigma(e))$ and its norm is bounded by $\left\|\tau^{-1}\right\| \cdot \|\sigma\| / 2$.

The rescaling in $\mathcal{Z}_\mathbb{R}$ is implemented by rescaling in $\mathcal{R}_Q$:

Proposition 3 (Rescaling in $\mathcal{R}$ is Rescaling in $\mathcal{Z}$). *Given a $g(X) \in \mathcal{R}_Q$ encoding $f(X) \in \mathcal{Z}_\mathbb{R}$ that is encrypted in $\mathsf{ct} = (-as + g + e, a) \in \mathcal{R}_Q^2$, the* RLWE.Rescale *operation gives noisy rescaling/division in $\mathcal{Z}$:*

$$\begin{aligned} \mathsf{ct}' &= \mathsf{RLWE.Rescale}_q(\mathsf{ct}) \quad \in \mathcal{R}_{Q/q}^2 \\ h(X) &= \mathsf{RLWE.Decrypt}_s(\mathsf{ct}') = \left\lfloor \frac{g(X)}{q} \right\rceil + e_{\text{rescale}} \qquad \in \mathcal{R}_{Q/q} \\ f(X) &= \tau^{-1}(\sigma(h(X))) = \left\lfloor \frac{f(X)}{q} \right\rceil + \tau^{-1}(\sigma(e_{\text{rescale}})) + e \end{aligned}$$

Where e is a small error for approximating $\tau^{-1}\left(\sigma\left(\left\lfloor \frac{g(X)}{q} \right\rceil\right)\right)$ with $\left\lfloor \frac{f(X)}{q} \right\rceil$. The rescaling noise in $\mathcal{Z}_\mathbb{R}$ is bounded by $\left\|\tau^{-1}\right\| \cdot \|\sigma\| \cdot \|e_{\text{rescale}}\|$.

Now we discuss the case of $N/n \neq 1$.

Definition 17 (Embedding of N/n copies of $\mathcal{Z}_\mathbb{R}$ in $\mathcal{R}_Q$). *Given N/n polynomial $f^{(i)}(X) \in \mathcal{Z}_\mathbb{R}$ for $0 \leq i < N/n$ that corresponds to $\boldsymbol{m} \in \mathbb{Z}_{2^n}^{N/n}$, its embedding in $\mathcal{R}_\mathbb{R}$ is*

$$\begin{aligned} \mathcal{R}_\mathbb{R}.\mathsf{Embed}_{\mathcal{Z}_\mathbb{R},\#\mathcal{Z}_\mathbb{R}}(\ldots, f^{(i)}(X), \ldots) &= \sigma^{-1}\left(\ldots, \tau(f^{(i)}(X)), \ldots\right) \\ \mathcal{Z}_\mathbb{R}.\mathsf{Recover}_{\mathcal{R}_\mathbb{R},k}(g(X)) &= \tau^{-1}\left(\sigma\left(g(X)\right)_{[kn/2,(k+1)n/2)}\right) \end{aligned}$$

where $\#\mathcal{Z}_\mathbb{R} = N/n$ is the number of $\mathcal{Z}$-slots.

Given that the k-th $\mathcal{Z}$-slot corresponds to $[kn/2, (k+1)n/2)$-th terms in $\mathbb{C}^{N/2}$, we can rotate the $\mathcal{Z}$-slots using CKKS.Rotate.

Definition 18 (Rotation of $\mathcal{Z}$-slots). *For a vector of messages $\boldsymbol{m} = (m_0, m_1, \ldots, m_{N/n-1}) \in \mathbb{Z}_{2^n}^{N/n}$ that is encoded as $f^{(i)}(X) \in \mathcal{Z}_\mathbb{R}$ and embedded in $\mathcal{R}_Q$ as $g(X)$, the rotation of $\boldsymbol{m}$ can be implemented as*

$$\#\mathcal{Z}_\mathbb{R}.\mathsf{Rotate}_k(g(X)) = \mathsf{CKKS.Rotate}_{k\cdot\frac{n}{2}}(g(X))$$

such that the result $h(X)$ corresponds to $\boldsymbol{m}' = (m_k, m_{k+1}, \ldots, m_{N/n-1}, m_0, \ldots)$.

Now we can discuss moving between $\mathcal{R}_\mathbb{R}$ and $\mathcal{Z}_\mathbb{R}$. Similar to the linear transformation in the approximate CKKS section, we define the following operation. We first discuss the $N/n = 1$ case.

Definition 19 ($\mathcal{Z}$-To-$\mathbb{C}$ and $\mathbb{C}$-To-$\mathcal{Z}$). *Given a polynomial $f(X) \in \mathcal{Z}_\mathbb{R}$ that is represented as $\boldsymbol{f} = (f_0, f_1, \ldots, f_{n-1}) \in \mathbb{R}^n$. Let $\boldsymbol{f} = (\boldsymbol{f}_0\,;\, \boldsymbol{f}_1)$ and let $g(X) = \mathcal{R}_\mathbb{R}.\mathsf{Embed}_{\mathcal{Z}_\mathbb{R}}(f(X))$, we can produce two $h_0(X)$ and $h_1(X)$ in $\mathcal{R}_\mathbb{R}$ that $\sigma(h_0(X)) =$*

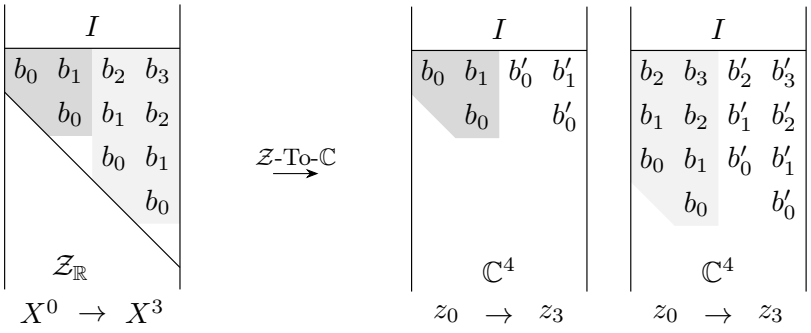


Fig. 4. The $\mathcal{Z}$-To-$\mathbb{C}$ linear transformation for a fully packed ciphertext. It produces two ciphertexts each containing one half of the original $\mathcal{Z}$-coefficients in the $\mathbb{C}$-slots. The example parameter is $n = 4$ and $N = 8$ so there are two $\mathcal{Z}$-slots, and the data b'_i for another $\mathcal{Z}$-slot is not drawn here. $X^0 \rightarrow X^3$ means the coefficients of $\mathcal{Z}_{\mathbb{R}}$ while $z_0 \rightarrow z_3$ means the slots in $\mathbb{C}^4$

$(f_0, f_1, \ldots, f_{n/2-1}) \in \mathbb{C}^{n/2}$ and $\sigma(h_1(X)) = (f_{n/2}, \ldots, f_{n-1}) \in \mathbb{C}^{n/2}$ using $\boldsymbol{V}_{\tau,0}$ and $\boldsymbol{V}_{\tau,1}$. This is called $\mathcal{Z}$-To-$\mathbb{C}$.

$$h_0(X) = 2\Re\left[\mathsf{CKKS.LT}_{\boldsymbol{V}_{\tau,0}}(g(X))\right]$$
$$h_1(X) = 2\Re\left[\mathsf{CKKS.LT}_{\boldsymbol{V}_{\tau,1}}(g(X))\right]$$

Then by using $h_0(X)$ and $h_1(X)$, we can restore back $g(X)$ in $\mathcal{R}_{\mathbb{R}}$ using $\boldsymbol{U}_{\tau,0}$ and $\boldsymbol{U}_{\tau,1}$, thus recovering $f(X)$ in $\mathcal{Z}_{\mathbb{R}}$. This is called $\mathbb{C}$-To-$\mathcal{Z}$.

$$g(X) = \mathsf{CKKS.LT}_{\boldsymbol{U}_{\tau,0}}(h_0(X)) + \mathsf{CKKS.LT}_{\boldsymbol{U}_{\tau,1}}(h_1(X))$$

For the $N/n \neq 1$ case, there are N/n copies of $\mathcal{Z}_{\mathbb{R}}$ and the linear transformations used are actually

$$\boldsymbol{U}_{\tau,0,\#\mathcal{Z}_{\mathbb{R}}} = I_{\#\mathcal{Z}_{\mathbb{R}}} \otimes \boldsymbol{U}_{\tau,0} = \begin{bmatrix} \boldsymbol{U}_{\tau,0} & & \\ & \boldsymbol{U}_{\tau,0} & \\ & & \ddots \end{bmatrix}$$

and similar matrices defined for $\boldsymbol{U}_{\tau,1}$, $\boldsymbol{V}_{\tau,0}$ and $\boldsymbol{V}_{\tau,1}$. In these matrices, only $n-1$ diagonals are non-zero and therefore the cost of its evaluation is $O(2\sqrt{2n})$ rotations by using baby-step-giant-step strategy for $(n/2-1)$ positive and $(n/2-1)$ negative diagonals. We also give an illustration in Fig. 4 on how $\mathcal{Z}$-To-$\mathbb{C}$ can break the bits of one ciphertext $g(X)$ into two ciphertexts $h_0(X)$ and $h_1(X)$ and we call it the "breaking-into-halves" effect.

Combined with the linear transformation in approximate CKKS, we are able to define $\mathcal{Z}$-To-$\mathcal{R}$ and $\mathcal{R}$-To-$\mathcal{Z}$. The overall encoding/decoding and linear transformation process is also illustrated in Fig. 7.

5 Bootstrapping: Refreshing and Conversion

In this section, we use "refreshing" for bootstrapping inside one mode, while using "conversion" for mode conversion where bootstrapping is usually the backbone.

The goal of bootstrapping is to allow more computation to carry on. The current ciphertext may reach a status that (1) the modulus Q is too small where further regular computation cannot be applied, or (2) the noise e is too large where further regular computation might corrupt the message.

For the modulus Q problem, the RLWE.ModRaise operation can regain a large amount of modulus, but it also introduces overflow I in the high bits. Therefore, CKKS.Sine and D-CKKS.LUT were devised to clean the overflow. Note that high bits I are integers in the ring $\mathcal{R}_Q$, whereas cleaning happens in $\mathbb{C}$. So $\mathcal{R}$-To-$\mathbb{C}$ is utilized to move the overflow into $\mathbb{C}$, while also moving the payload message m. The message m needs to be moved back using $\mathbb{C}$-To-$\mathcal{R}$ when finished.

The noise e problem is trickier. In approximate CKKS, the noise is part of the message. Discrete CKKS, on the other hand, handles discrete data and need clear separation of the noise, therefore D-CKKS.LUT is designed with different interpolation order to clean the noise. We emphasize that D-CKKS.LUT requires a large "gap" between $\frac{\Delta}{p}m$ and e for the noise cleaning functionality to be effective (e.g., [AKP25, Fig. 2]).

5.1 Arithmetic-To-Arithmetic (A2A) Refreshing

In arithmetic mode, we are given $f(X) = \Delta I + \frac{\Delta}{t}[m]_t + e$ in $\mathcal{Z}_{\mathbb{R}}$. Our goal is to obtain a new $f'(X) = \Delta I' + \frac{\Delta}{t}[m]_t + e'$ with $\|e'\|$ smaller than $\|e\|$ and $\|I'\|$ usually smaller than $\|I\|$. Note that we do not require cleaning the ΔI completely, which is the key to efficient refreshing.

First, the familiar high I structure reminds us of RLWE.ModRaise. By using (1) $\mathcal{Z}$-To-$\mathcal{R}$, (2) RLWE.Truncate and ModRaise, and (3) $\mathcal{R}$-To-$\mathcal{Z}$, we are able to reset I to a new I' with I' being the small noise in Prop 2. This also regains modulus Q and allows more computation to go on. This is called A2A I.

For the noise e, the line META-BTS and Bootstrapping for BFV/GBFV using CKKS [BCC+22, KSS24, Kim25a] gives us the inspiration that we can bootstrap the noise e alone. Note that if we multiply $f(X)$ by t in $\mathcal{Z}_{\mathbb{R}}$, the message $\frac{\Delta}{t}[m]_t$ also becomes a part of the high I structure(Fig. 6), and can be cleaned by RLWE.Truncate. Given te, we are able to bootstrap it using CKKS.Sine to regain modulus, then multiplying by t^{-1} in $\mathcal{Z}_{\mathbb{R}}$ gives us the e. Subtracting the bootstrapped e from the original ciphertext, we obtain e' that is smaller. It can be summarized as (1) $\mathcal{Z}$-To-$\mathcal{R}$ that fuses multiplication of t, (2) RLWE.Truncate and ModRaise (3) $\mathcal{R}$-To-$\mathbb{C}$, (4) CKKS.Sine (5) $\mathbb{C}$-To-$\mathcal{Z}$ that fuses multiplication by t^{-1} in $\mathcal{Z}_{\mathbb{R}}$ and (6) Subtraction from the original ciphertext. This is called A2A e.

For the full A2A refreshing, we first execute A2A I to reset the large I to a smaller one, then execute A2A e to clean the noise. It is not recommended to reverse the order of them because large I during A2A e will introduce additional noise from linear transformation and will have less noise cleaning effect.

It is worth noting that the arithmetic overflow I is a major cause for noise growth in multiplication like $\mathcal{Z}_{\mathbb{R}}.\mathsf{Mult}$. We may sometimes only invoke $\mathsf{A2A}\ I$ to reduce noise growth of further multiplications, especially when the operands of multiplication have asymmetric noise/overflow.

5.2 Arithmetic-To-Boolean (A2B) Conversion

The arithmetic mode gives us $f(X) = \Delta I + \frac{\Delta}{t}[m]_t + e$ in $\mathcal{Z}_{\mathbb{R}}$ with $[m]_t = \sum_{i=0}^{n-1} b_i X^i$ where b_i is the i-th bit of $m \in \mathbb{Z}_{2^n}$. The boolean mode requires $\Delta \boldsymbol{b} + \boldsymbol{e} = \Delta(b_0, b_1, \ldots, b_{n-1}) + \boldsymbol{e}$ in $\mathbb{C}^n$.

As the arithmetic mode has a triangle shape (Fig. 1) for $\frac{\Delta}{t}[m]_t$, the conversion is the extraction of the bits in upper side of the triangle. This can be achieved by $\mathsf{D\text{-}CKKS.LUT}$ with the target function $\mathsf{LUT}_{\mathsf{MSB}} : \mathbb{Z}_p \to \mathbb{Z}_p$ that extracts the most significant bit.

First we discuss the case where $2^n \le p$, then discuss how to iteratively achieve arithmetic-to-boolean conversion for large 2^n.

For the case $2^n \le p$, $\frac{\Delta}{t}[m]_t$ can be reinterpreted as discrete CKKS data if viewing polynomial as vector.

$$\begin{aligned}
\frac{\Delta}{t}[m]_t &= \Delta\left(\frac{m}{2^n} - \sum_{k=0}^{n-1} \frac{X^k}{2^{k+1}}\,{\scriptstyle [m]_{0\le i\le k}}\right) \\
&\overset{\text{cast}}{=} -\left(\cdots, \frac{\Delta}{2^n}\cdot\frac{1}{2}\cdot 2^{n-k}\,{\scriptstyle [m]_{0\le i\le k}}, \cdots\right) + \left(\frac{m}{2^n}, 0, 0, \ldots\right) \\
&= -\left(\cdots, \Delta.\overline{b_k b_{k-1}\cdots}, \cdots\right) + \left(\Delta\frac{m}{2^n}, 0, 0, \cdots\right)
\end{aligned}$$

where $\overline{.b_k b_{k-1}\cdots}$ means $\frac{1}{2}b_k + \frac{1}{4}b_{k-1} + \cdots$.

We need a preprocessing that removes $\Delta\frac{m}{2^n}$. Note that the constant coefficient contains $\Delta\frac{m}{2^n} - \Delta.\overline{b_0}$ and the first coefficient contains $-\Delta.\overline{b_1 b_0}$. We can replace the constant coefficient with twice of the first coefficient, so that the constant coefficient becomes $-\Delta\overline{b_1.b_0} = \Delta I - \Delta.\overline{b_0}$. This preprocessing is implemented by replacing the values of the first row with twice the second row, for the matrices in $\mathcal{Z}$-To-$\mathbb{C}$ linear transformation. This is called special-b_0 $\mathcal{Z}$-To-$\mathbb{C}$.

The $\mathsf{D\text{-}CKKS.LUT}$ operation is level-consuming and is often carried out during bootstrapping. Therefore the conversion procedure is (1) Special-b_0 $\mathcal{Z}$-To-$\mathcal{R}$ (2) $\mathsf{RLWE.Truncate}$ and $\mathsf{ModRaise}$, (3) $\mathcal{R}$-To-$\mathbb{C}$, (4) $\mathsf{D\text{-}CKKS.LUT}$ that evaluates $\mathsf{LUT}_{\mathsf{MSB}}$.

The $2^n > p$ case needs more iteration. Let $w = \log p$ and $d = \lceil n/w \rceil$. We need d iterations to extract $\overline{.b_{n-1}}$. Note that the coefficient in the $(n-1)$-th coefficient is

$$\begin{aligned}
&\Delta.\overline{b_{n-1}\cdots b_{(d-1)w}\cdots b_{w-1}\cdots b_0} \\
=&\Delta.\overline{b_{n-1}\cdots b_{(d-1)w}} + \frac{\Delta}{p}.\overline{b_{(d-1)w-1}\cdots b_{(d-2)w}} + \cdots + \frac{\Delta}{p^{d-1}}.\overline{b_{w-1}\cdots b_0}
\end{aligned}$$

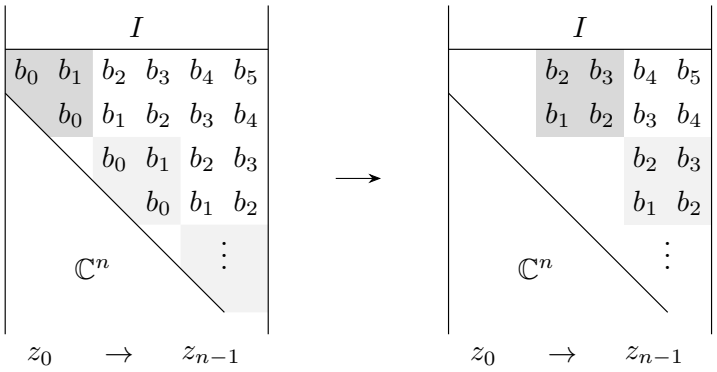


Fig. 5. Iterative process of extracting w-MSB and removing w-LSBs in Arithmetic-To-Boolean Conversion. We set $w = 2$ here for exposition and is not the actual parameter. The thicker gray region is the extracted w-MSB, while the thinner gray region is the removed w-LSBs. The arrow in the middle means next iteration. $z_0 \to z_{n-1}$ means the slots in $\mathbb{C}^n$. The "breaking-into-halves" effect of $\mathcal{Z}$-To-$\mathbb{C}$ in Fig. 4 is not pictured here.

We can iteratively extract $\Delta.\overline{b_{w-1} \cdots b_0}$ using $\mathsf{LUT}_{\mathsf{ID}}$ that evaluates the identity function until we obtain $\Delta.\overline{b_{n-1} \cdots b_{(d-1)w}}$. The iterative process is also shown in Fig. 5.

The conversion procedure is summarized as (1) Special-b_0 $\mathcal{Z}$-To-$\mathbb{C}$ to obtain one *core ciphertext* (with two ciphertexts due to Fig. 4), (2) Pick the least w bits from core ciphertext by masking and execute one iteration: [1] $\mathbb{C}$-To-$\mathcal{R}$, [2] RLWE.Truncate and ModRaise, [3] $\mathcal{R}$-To-$\mathbb{C}$, [4] D-CKKS.LUT that evaluates both $\mathsf{LUT}_{\mathsf{MSB}}$ and $\mathsf{LUT}_{\mathsf{ID}}$, and [5] Subtract $\mathsf{LUT}_{\mathsf{ID}}$ result from core ciphertext. (3) Do the next iteration with the next w bits. (4) When all iterations are finished, combine all ciphertexts obtained using $\mathsf{LUT}_{\mathsf{MSB}}$.

One concern is that in each subtraction step, the number of rotation grows linearly with the bit-width n. We therefore set a threshold A2B-cutoff where bits below the threshold are treated as noise for D-CKKS.LUT, so the number of rotations is constant[5] for one ciphertext in each subtraction step and the overall number of rotations for subtraction is $O(n)$.

We provide a simplified algorithmic listings in Appendix F. For more details, please refer to the codebase.

Batched Arithmetic-To-Boolean (B-A2B) Conversion We further notice that, inside each iteration, only w bits of each n-wide slot is utilized, therefore in each iteration we can bootstrap n/w ciphertexts at once, and the number of CKKS bootstrapping is amortized to *one per ciphertext*.

The conversion procedure for n/w ciphertexts is summarized as (1) Special-b_0 $\mathcal{Z}$-To-$\mathbb{C}$ to obtain n/w *core ciphertexts* ($2(n/w)$ ciphertexts), (2) (Pre-)rotate

[5] For A2B-cutoff$= -16$ and $w = 4$, the number of rotation is 3.

the i-th core ciphertext by $-i \cdot w$ for later merging, (3) Pick the least w bits from n/w core ciphertexts using *masking*, merge them into two CKKS ciphertexts using addition and execute one iteration: [1] $\mathbb{C}$-To-$\mathcal{R}$, [2] RLWE.Truncate and ModRaise, [3] $\mathcal{R}$-To-$\mathbb{C}$, [4] D-CKKS.LUT that evaluates both $\mathsf{LUT}_{\mathsf{MSB}}$ and $\mathsf{LUT}_{\mathsf{ID}}$, [5] Subtract $\mathsf{LUT}_{\mathsf{ID}}$ results from each core ciphertext, with corresponding masking. (4) Do the next iteration with the next w bits. (5) When all iterations are finished, *combine* $\mathsf{LUT}_{\mathsf{MSB}}$ results into n/w ciphertexts using masking, and (post-)rotate the i-th result ciphertext by $i \cdot w$ to restore to the original offset.

Linear transformation $\mathbb{C}$-To-$\mathcal{R}$ can process two ciphertexts at once by merging the messages in the second ciphertext into the imaginary part of the first ciphertext. The subtraction of $\mathsf{LUT}_{\mathsf{ID}}$ results and the combination of $\mathsf{LUT}_{\mathsf{MSB}}$ results consumes the same CKKS level, as they are independent branches.

For complexity analysis, for each core ciphertext, pre-rotation and post-rotation each takes 1 rotation; the subtraction step in each iteration takes constant number of rotations due to the cutoff parameter. The number of rotations for pre-rotation/subtraction/post-rotation for each core ciphertext is $O(n)$.

5.3 Boolean-To-Boolean (B2B) Refreshing

The boolean-to-boolean refreshing is discrete CKKS bootstrapping for bits and is implemented using polynomial approximation of cosine [AKP25]. For completeness, we describe it as (1) $\mathbb{C}$-To-$\mathcal{R}$, (2) RLWE.Truncate and RLWE.ModRaise, (3) $\mathcal{R}$-To-$\mathbb{C}$ and (4) D-CKKS.LUT that is implemented as cosine evaluation.

5.4 Boolean-To-Arithmetic (B2A) Conversion

Given a ciphertext containing $\Delta \boldsymbol{b} = (b_0, \ldots, b_{n-1})$ in $\mathbb{C}^n$, it can be viewed as $\Delta[m]_t$ if linear transformed to $\mathcal{Z}_{\mathbb{R}}$. So the conversion procedure can be fused with a $\mathcal{Z}_{\mathbb{R}}$.Mult operation as is (1) $\mathbb{C}$-To-$\mathcal{Z}$ (2) $\mathcal{Z}_{\mathbb{R}}$.MultShort. If the next operation for m is $\mathcal{Z}_{\mathbb{R}}$.Add, we can directly call a $\mathbb{C}$-To-$\mathcal{Z}$ that fuses the multiplication of t^{-1}.

6 Analysis and Parameter Selection

6.1 Noise Analysis and Security Implications

The security notions of FHE are *unconventional* as FHE is inherently dynamic by allowing evaluation of arbitrary user-chosen circuits. From the attack [LM21, LMSS22] that proposes the $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ security, to recent attacks like [GNSJ24, CCP+24, CSBB24] that incur discussion like *application-aware* $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ security [AAMP26] stating that software controls should be mandated based on user applications,to the very recent attack stating the impossibility of $\mathsf{IND\text{-}CPA}^{\mathsf{D}}$ security for certain schemes [CCRS26], this research area is actively evolving. Among these attacks, a critical connection is the interchangeability between *security* and *correctness*. In short, if the scheme does not achieve correctness, its security is also corrupted, independent of the (R)LWE assumption.

Table 3. Empirical noise growth (in bits). The growth is measured by the difference of the noise (in bits) after a Mult operation of two A2A refreshed ciphertexts, or a MultShort operation of an A2A refreshed ciphertext and a plaintext. The data was collected from experiments for illustration purpose and should not be used as implementation guideline.

BitW n	**Mult** (max/avg)	**MultShort**	**BitW** n	**Mult**	**MultShort**
8	4.3 / 3.9	2.6 / 1.5	64	5.6 / 5.0	3.3 / 2.6
16	5.0 / 4.3	2.4 / 1.7	128	5.8 / 5.4	3.6 / 2.9
32	4.9 / 4.6	2.9 / 2.3	256	6.3 / 5.9	4.0 / 3.4
Arith $\log(\|e_A\|/\Delta)^\dagger = -24.4$		**Arith** $\log\|I_A\|^\ddagger = 2.8$		**Bool** $\log(\|e_B\|/\Delta)^\dagger = -20.4$	

† e_A and e_B stand for the *bottom noise*, which is the noise that cannot be reduced by further bootstrapping, for arithmetic and boolean mode respectively.
‡ The typical arithmetic overflow I_A after A2A I refreshing.

We stick to the *application-aware* security model, relying on software control (e.g., application specification language in [AAMP26], compilers like [ACG+25, EGR25]) to provide *exact correctness* and *application-aware* IND-CPA$^{\mathsf{D}}$ security.

In this section, we provide worst-case noise analysis tools as the basis for software control. Rigorous average-case noise analyses [KPZ21, CCH+24, BCM+25, BJSW25, GZ25] can be used in production with proper validation and are out of the scope of this work. We provide *debugging* utilities in our open source prototype that can extract the current noise upon decryption for examination of the noise behavior.

Arithmetic Mode. We want messages after A2A refreshing to maintain the form $\Delta I_A + \frac{\Delta}{t}[m]_t + e_A$ in $\mathcal{Z}_{\mathbb{R}}$, where e_A is an *arithmetic bottom noise* that cannot be reduced by further bootstrapping. The *minimal overflow* I_A as the result of A2A I satisfies $\|I_A\| < 16$ with negligible failing probability because we use sparse encapsulated secret key during ModRaise.

Theorem 3 (A2A Refreshing Correctness). *To refresh an arithmetic mode ciphertext with messages* $\Delta I + \frac{\Delta}{t}[m]_t + v$ *to bottom noise, we require the input noise* v *to satisfy*

$$C_1 \left(\frac{\|tv\|}{\Delta} \right)^3 < \frac{\|e_A\|}{\Delta}$$

where C_1 *is a constant determined by* CKKS.Sine.

Proof. During A2A e*, we are bootstrapping the noise* tv *in* $\mathbb{C}$. CKKS.Sine *incurs additional error of* $C_1\Delta(\|tv\|/\Delta)^3$*. We require the additional noise to be smaller than the bottom noise.*

For multiplication in arithmetic mode, we have the following noise growth formula:

Theorem 4 (Arithmetic Multiplication Noise Growth). *For two messages $f = \Delta I + \frac{\Delta}{t}[m]_t + v$ and $f' = \Delta I' + \frac{\Delta}{t}[m']_t + v'$, the noise v'' and the overflow I'' after $\mathcal{Z}_{\mathbb{R}}.\mathsf{Mult}$ satisfies*

$$\|v''\| \leq 3n \left[(4\|I\| + 1) \cdot \|v'\| + (4\|I'\| + 1) \cdot \|v\| + \frac{4\|v\|\|v'\|}{\Delta} \right]$$
$$\|I''\| \leq 3n \left(4\|I\| \cdot \|I'\| + \|I\| + \|I'\| + 1\right) + 1$$

For the multiplication of f and $g = \Delta[m_g]_t + v_g$, the noise v''' and the overflow I''' after $\mathcal{Z}_{\mathbb{R}}.\mathsf{MultShort}$ satisfies

$$\|v'''\| \leq 3n \left[(\|I\| + 1) \cdot \|v_g\| + \|v\| + \frac{\|v\|\|v_g\|}{\Delta} \right]$$
$$\|I'''\| \leq 3n \left(\|I\| + 1\right) + 1$$

Proof. Deferred to Appendix E.

We remark that the norm of I is the major cause for noise growth in multiplication. Therefore, we devise $\mathsf{A2A}$ I refreshing to significantly reduce the noise growth of further multiplications.

Boolean Mode. We want to maintain ciphertexts after $\mathsf{B2B}$ refreshing to have messages of form $\Delta\boldsymbol{b} + \boldsymbol{e}_B$, where $\boldsymbol{e}_B$ is the *boolean bottom noise*. For boolean operations, the underlying operation is multiplication in $\mathbb{C}$, so the norm evolution is $\|f \cdot f'\| \leq \|f\| \cdot \|f'\|$. We now state the correctness requirement for bootstrapping operations.

Theorem 5 (B2B Refreshing Correctness). *To refresh a boolean mode ciphertext with messages $\Delta\boldsymbol{b} + \boldsymbol{v}$ to bottom noise, we require*

$$C_2 \left(\frac{\|v\|}{\Delta} \right)^2 < \frac{\|\boldsymbol{e}_B\|}{\Delta}$$

where C_2 is a constant determined by $\mathsf{CKKS.Cosine}$.

Proof. Cosine evaluation will reduce the noise to $C_2\Delta(\|v\|/\Delta)^2$, and we require it to be smaller than bottom noise.

Theorem 6 (A2B Conversion Correctness). *To convert an arithmetic mode ciphertext with messages $\Delta I + \frac{\Delta}{t}[m]_t + v$ to boolean mode, we require the input noise v to satisfy*

$$C_3 \left(\frac{\|v\| p}{\Delta} \right)^{\kappa+1} < \frac{\|\boldsymbol{e}_B\| p}{\Delta}$$

where C_3 is a constant determined by $\mathsf{D\text{-}CKKS.LUT}$.

Proof. See Def. 8 for the analysis.

For illustration purpose, we conducted experiments to record the empirical bottom noise for both arithmetic mode and boolean mode, and record the empirical noise growth of two kinds of multiplications *for bottom ciphertexts* in arithmetic mode with the parameter in Table 5. The results are shown in Table 3.

Table 4. The level consumption of each bootstrapping. The available levels refer to the *level budget* for arithmetic or boolean operations after bootstrapping. The concrete number of available arithmetic or boolean operations depends on both the noise growth and the level consumption.

A2A I	R2C 3		C2Z 1	Avail Arith $L-7$	Z2C 1	C2R 2
A2A e	R2C 3	Sine 9	C2Z(t^{-1}) 1	Avail Arith $L-16$	Z2C(t) 1	C2R 2
B-A2B	R2C 3	LUT 13	Comb† 1	Avail Bool‡ $L-19$	Mask* 1	C2R 2
B2B	R2C 3	Cosine 8		Avail Bool $L-13$		C2R 2

* Ciphertexts from arithmetic mode need 1 level to arrive here using special-b_0 Z2C.
† This level is used for the subtraction of $\mathsf{LUT}_{\mathsf{ID}}$ results from core ciphertexts, and the combination of $\mathsf{LUT}_{\mathsf{MSB}}$ results into output ciphertexts, which are independent branches and can be done in the same level. In A2B, the combination step does not consume level and the available level is $L-18$.
‡ After B-A2B, the level reserved for mask can be used for computation.

6.2 Level Analysis

Let L be the available multiplicative depth for CKKS parameter selection. The overall level consumption for each bootstrapping, and the available arithmetic or boolean spaces are presented in Table 4. For $\mathcal{R}$-To-$\mathbb{C}$ and $\mathbb{C}$-To-$\mathcal{R}$, which corresponds to conventional CoeffsToSlots and SlotsToCoeffs, we allocate 3 and 2 levels, respectively, for reducing the linear transformation evaluation cost [CCS19]. The CKKS.Sine operation, adapted from [BMTH21], requires a polynomial approximation of degree $d_{g_0} = 32$ and takes $L_{\sin} = 6+3$ levels, where 3 comes from three application of double angle formula.

The D-CKKS.LUT operation with interpolation order 1 for $p = 2^w$ where $w = 4$, adapted from [AKP25], requires a polynomial approximation of degree $d_{\exp} = 46$ for the exponential function, and another polynomial evaluation of degree $d = p-1 = 15$ with interpolation order of 1, consuming $6+2+5$ levels, where 2 comes from two application of double angle formula.

The cosine operation requires a polynomial approximation of degree $d_{\cos} = 50$ and consumes $6+2$ level, where the extra 2 comes from one application of double angle formula and one squaring to obtain $\cos^2(\pi x)$.

6.3 Parameter Selection

We select $L = 20$ for all bootstrapping operations to function, and we select $\log_2 \Delta \approx 43$ for all experiments to carry out, therefore there are 21 primes in the moduli chain. We select 7 auxiliary primes (50-bit) to construct P with $\mathsf{dnum} = 3$ for hybrid key switching. The auxiliary prime is larger than Δ by 6

Table 5. Parameters for all experiments

RingDim N	**Scaling Factor** $\log \Delta$	**MulDepth**[†] L	**RLWE Modulus** $\log QP$	**SK Dist** $(h\ ;\ \tilde{h})$
2^{16}	43	20	1254	$(192\ ;\ 32)$

[†] Refers to the CKKS multiplicative depth, not the available multiplicative depth for arithmetic mode in our scheme.

bits for sparse encapsulated key switching in ModRaise to introduce negligible noise. The threshold A2B-cutoff parameter for A2B is set to -16.

Under such selection, the RLWE modulus is $\log QP \approx 1254$ and we therefore select the RLWE ring dimension to be $N = 2^{16}$. The RLWE error distribution is Gaussian with standard deviation $\sigma = 3.2$. We use a sparse secret key of Hamming weight $h = 192$ and a sparse encapsulated secret key of Hamming weight $\tilde{h} = 32$. This achieves 128-bit security according to [BTH22].

We employ the level-specific scaling factor Δ_ℓ approach from [KPP22] for eliminating the scaling factor approximation noise. We slightly alter its prime selection logic by setting the starting scaling factor to the first modulus q_0 as RLWE.Truncate operation resets the rescaling factor to q_0 and RLWE.ModRaise makes it become the starting scaling factor.

We would like to mention that it is also possible to select a uniform ternary main secret key while using the sparse encapsulated key for ModRaise, or stick to a single uniform ternary secret key, for approaching the standard recommendation like [ACC+18, BCC+24]. Our scheme is functional under all these settings, except with a larger noise growth in the former case, or with non-negligible CKKS bootstrapping failing probability and performance penalty (at most 25% by estimation in [AKP25]) in the latter case.

7 Implementation, Benchmark, and Comparison

We implement our scheme based on OpenFHE [BAB+22] v1.4.2 with HEXL [BKS+21]. Our experiments and experiments for [Kim25b, Kim25c, CPL26] using their code were conducted in a single-thread setting on a Linux machine with AMD RYZEN AI MAX+ 395 CPU and 128GB RAM, running Debian 13. The compiler used is clang 19.

The detailed performance of bootstrapping operations for bit-width from 8 to 256 is present in Table 6. The data were collected by executing each bootstrapping 5 times and taking the average, with one extra warm-up run before the collection. A performance comparison of our scheme with previous works can be checked in Table 2.

The experiments for TFHE-rs were conducted in a single thread setting on the same machine with AVX512 acceleration, using `integer/bench.rs`, especially `mul_parallelized` from the commit `222a7e93c` under `RAYON_NUM_THREADS=1`.

Table 6. Detailed performance numbers of each bootstrapping in our scheme. n is the bit-width of the plaintext message, ranging from 8 to 256. The first number in each cell is the total time for bootstrapping, and the second number is the amortized time per ciphertext. This table extends Table 2 with data for more bit-widths.

n	#Slot	A2A I	A2A†	A2B	B-A2B‡	B2B	B2A
8	8192	4.3s/0.5ms	10.1s/1.2ms	10.1s/1.2ms	13.0s/0.8ms	6.0s/0.7ms	0.7s/0.1ms
16	4096	4.7s/1.1ms	10.9s/2.7ms	20.5s/5.0ms	27.0s/1.7ms	5.9s/1.4ms	1.1s/0.3ms
32	2048	5.3s/2.6ms	11.8s/5.8ms	40.8s/20ms	57.7s/3.5ms	5.9s/2.9ms	1.7s/0.8ms
64	1024	6.4s/6.3ms	13.5s/13.2ms	80.5s/79ms	128s/ 7.8ms	5.9s/5.8ms	2.8s/2.7ms
128	512	7.7s/15ms	15.3s/29.9ms	164s/320ms	309s/18.9ms	5.8s/11ms	4.1s/8.0ms
256	256	10.2s/40ms	19.5s/76.1ms	327s / 1.3s	823s / 50ms	5.9s/23ms	6.4s/25ms

† A2A e data can be obtained by subtracting A2A data with A2A I data.
‡ Batched A2B processes $n/4$ ciphertexts for bit-width of n.

Compared with the CPL scheme [CPL26], our multiplication takes 2 CKKS bootstrapping operations (or less for leveled execution), while their scheme takes 2 to 3 CKKS bootstrapping operations; we also concretely outperform their scheme by 1.7× to 2× for multiplication for bit-width from 64 to 256. We remark that one performance advantage comes from our smaller parameter of $L = 20$ compared with their $L = 26$ or $L = 27$. We show worse performance for conversion to boolean mode in a single-ciphertext setting, but we show 4.4× to 5.2× better performance for batched conversion to boolean mode.

On the other hand, the CPL scheme can support very-high-precision arithmetic like $\mathbb{Z}_{2^{2048}}$ and arbitrary modulus through Montgomery reduction, therefore enabling cryptographical applications like RSA2048[6]. Their arithmetic-to-boolean conversion takes only $O(1)$ bootstrapping and is therefore suitable for scenarios with very-high-precision arithmetic interleaved with logic operations, under settings when there are a small amount of plaintext messages.

Compared with [Kim25c], we show similar throughput in single-ciphertext setting for both arithmetic-to-arithmetic refreshing and arithmetic-to-boolean conversion; but we show significanly higher throughput in batched setting.

Compared with [Kim25b] that targets the batched setting under bit-width of 64, their "Batched A2B" can support further multiplications and bitwise operations. We show 1.6× higher throughput for conversion to boolean mode, while showing a similar performance for multiplication. They do not report performance for bit-width of 256.

Compared with REFHE [BFG+26] in 64-bit settings by estimation, our throughput is 15× higher for multiplication and 491× higher for bitwise operations. Compared with TFHE-rs [Zam22], our throughput of 64-bit multiplication is 863× higher while our latency is similar; for 256-bit multiplication our

[6] We could support arbitrary plaintext modulus using similar ideas but it is not further explored in this work.

throughput is 2485× higher and our latency is 9× lower; our throughput of bitwise operation is 27× higher.

Recent developments in low-latency CKKS bootstrapping like [CHKS25, CK25, CS25] could replace our CKKS.Sine call for achieving smaller RLWE parameter and this is an interesting future direction. Recent work supports other kinds of arithmetic (e.g. matrix arithmetic [Par25, GL25]) and they are orthogonal to our work.

A recent work [PZZ25] also supports arithmetic operations on $\mathbb{Z}_{2^{64}}$ and additionally arbitrary plaintext modulus. Their idea is to decompose an integer into digits encrypted in different BFV ciphertexts with redundant representation and execute reduction for managing the multiplication results. However, for bitwise boolean operations and comparison, they will need iterative bootstrapping with the number linear in the bit-width. At the time of writing, bootstrapping performance is not reported, so the performance numbers are not directly comparable.

8 Conclusion

This work presents a CKKS-style FHE scheme supporting both arithmetic and boolean operations on SIMD slots of machine words, for scenarios with massive amount of data. We crucially amortize the inherently iterative process for conversion between arithmetic and boolean representations to *constant* using our novel triangle encoding and structural properties of the RLWE problem. An interesting open problem is whether it is possible to convert from arithmetic to boolean mode using $O(1)$ bootstrapping without amortization.

Acknowledgments. The authors used ChatGPT 5.2 for grammar and spell checking. We thank Yilei Chen, Yuriy Polyakov, Bingwu Xiang, Jeremy Kun and Yi Zhang for feedback on early drafts of this work. We thank Yangyu Chen for providing machines for the experiments. We thank the anonymous reviewers for their valuable feedback and suggestions.

References

AAMP26. Andreea Alexandru, Ahmad Al Badawi, Daniele Micciancio, and Yuriy Polyakov. Application-aware approximate homomorphic encryption: Configuring FHE for practical use. *IACR Communications in Cryptology*, 2(4), 2026.

ACC+18. Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

ACG+25. Asra Ali, Jaeho Choi, Bryant Gipson, Shruthi Gorantala, Jeremy Kun, Wouter Legiest, Lawrence Lim, Alexander Viand, Meron Zerihun Demissie,

and Hongren Zheng. Heir: A universal compiler for homomorphic encryption, 2025.

AKP25. Andreea Alexandru, Andrey Kim, and Yuriy Polyakov. General functional bootstrapping using CKKS. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 304–337. Springer, Cham, August 2025.

BAB+22. Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R. V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Report 2022/915, 2022.

BCC+22. Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. META-BTS: Bootstrapping precision beyond the limit. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 223–234. ACM Press, November 2022.

BCC+24. Jean-Philippe Bossuat, Rosario Cammarota, Ilaria Chillotti, Benjamin R. Curtis, Wei Dai, Huijing Gong, Erin Hales, Duhyeong Kim, Bryan Kumara, Changmin Lee, Xianhui Lu, Carsten Maple, Alberto Pedrouzo-Ulloa, Rachel Player, Yuriy Polyakov, Luis Antonio Ruiz Lopez, Yongsoo Song, and Donggeon Yhee. Security guidelines for implementing homomorphic encryption. *CiC*, 1(4):26, 2024.

BCK+23. Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, Jai Hyun Park, and Damien Stehlé. HERMES: Efficient ring packing using MLWE ciphertexts and application to transciphering. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 37–69. Springer, Cham, August 2023.

BCKS24. Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, and Damien Stehlé. Bootstrapping bits with CKKS. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 94–123. Springer, Cham, May 2024.

BCM+25. Jean-Philippe Bossuat, Anamaria Costache, Christian Mouchet, Lea Nürnberger, and Juan Ramón Troncoso-Pastoriza. Accurate and composable noise estimates for CKKS with application to exact HE computation. *IACR Communications in Cryptology*, 2(2), 2025.

BdBB+25. Mathieu Ballandras, Mayeul de Bellabre, Loris Bergerat, Charlotte Bonte, Carl Bootland, Benjamin R. Curtis, Jad Khatib, Jakub Klemsa, Arthur Meyre, Thomas Montaigu, Jean-Baptiste Orfila, Nicolas Sarlin, Samuel Tap, and David Testé. *TFHE-rs: A (Practical) Handbook*. Zama, first edition, February 2025.

BFG+26. Zvika Brakerski, Offir Friedman, Daniel Golan, Alon Gurny, Dolev Mutzari, and Ohad Sheinfeld. Refhe: Fully homomorphic alu. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026*, pages 183–213, Cham, 2026. Springer Nature Switzerland.

BGGJ20. Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar P. Jetchev. CHIMERA: Combining Ring-LWE-based Fully Homomorphic Encryption Schemes. *Journal of Mathematical Cryptology*, 14(1):316–338, 2020.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

BJSW25. Olivier Bernard, Marc Joye, Nigel P. Smart, and Michael Walter. Drifting towards better error probabilities in fully homomorphic encryption schemes. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VIII*, volume 15608 of *LNCS*, pages 181–211. Springer, Cham, May 2025.

BK25. Dan Boneh and Jaehyung Kim. Homomorphic encryption for large integers from nested residue number systems. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 338–370. Springer, Cham, August 2025.

BKS+21. Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe DM de Souza, Vinodh Gopal, et al. Intel HEXL (release 1.2). https://github.com/intel/hexl, 2021.

BKSS24. Youngjin Bae, Jaehyung Kim, Damien Stehlé, and Elias Suvanto. Bootstrapping small integers with CKKS. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 330–360. Springer, Singapore, December 2024.

BMTH21. Jean-Philippe Bossuat, Christian Mouchet, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 587–617. Springer, Cham, October 2021.

Bra12. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Berlin, Heidelberg, August 2012.

BTH22. Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In Giuseppe Ateniese and Daniele Venturi, editors, *ACNS 2022*, volume 13269 of *LNCS*, pages 521–541. Springer, Cham, June 2022.

BV11. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

CCH+24. Anamaria Costache, Benjamin R. Curtis, Erin Hales, Sean Murphy, Tabitha Ogilvie, and Rachel Player. On the precision loss in approximate homomorphic encryption. In Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen, editors, *SAC 2023*, volume 14201 of *LNCS*, pages 325–345. Springer, Cham, August 2024.

CCK+25. Jung Hee Cheon, Hyeongmin Choe, Minsik Kang, Jaehyung Kim, Seonghak Kim, Johannes Mono, and Taeyeong Noh. Grafting: Decoupled scale factors and modulus in RNS-CKKS. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *ACM CCS 2025*, pages 1098–1112. ACM Press, October 2025.

CCKS23. Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Damien Stehlé. Homomorphic multiple precision multiplication for CKKS and reduced modulus consumption. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 696–710. ACM Press, November 2023.

CCP+24. Jung Hee Cheon, Hyeongmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. Attacks against the IND-CPA^{D} security of exact FHE schemes. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 2505–2519. ACM Press, October 2024.

CCRS26. Marina Checri, Pierre-Emmanuel Clet, Marc Renard, and Renaud Sirdey. Impossibility of CPAD security for a class of FHE schemes. Cryptology ePrint Archive, Paper 2026/203, 2026.

CCS19. Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 34–54. Springer, Cham, May 2019.

CGGI20. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.

CHK+18. 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, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 360–384. Springer, Cham, April / May 2018.

CHKS25. Jung Hee Cheon, Guillaume Hanrot, Jongmin Kim, and Damien Stehlé. SHIP: A shallow and highly parallelizable CKKS bootstrapping algorithm. Cryptology ePrint Archive, Report 2025/784, 2025.

CHM+25. Hyunho Cha, Intak Hwang, Seonhong Min, Jinyeong Seo, and Yongsoo Song. MatriGear: Accelerating authenticated matrix triple generation with scalable prime fields via optimized HE packing. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 2453–2471. IEEE Computer Society Press, May 2025.

CK25. Jean-Sébastien Coron and Robin Köstler. Low-latency bootstrapping for CKKS using roots of unity. Cryptology ePrint Archive, Report 2025/651, 2025.

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, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, Cham, December 2017.

CLPX18. Hao Chen, Kim Laine, Rachel Player, and Yuhou Xia. High-precision arithmetic in homomorphic encryption. In Nigel P. Smart, editor, *CT-RSA 2018*, volume 10808 of *LNCS*, pages 116–136. Springer, Cham, April 2018.

CPL26. Gyeongwon Cha, Dongjin Park, and Joon-Woo Lee. Improved radix-based approximate homomorphic encryption for large integers via lightweight bootstrapped digit carry. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026*, pages 243–273, Cham, 2026. Springer Nature Switzerland.

CS25. Jean-Sébastien Coron and Tim Seuré. PaCo: Bootstrapping for CKKS via partial CoeffToSlot. In Goichiro Hanaoka and Bo-Yin Yang, editors, *ASIACRYPT 2025, Part VII*, volume 16251 of *LNCS*, pages 36–67. Springer, Singapore, December 2025.

CSBB24. Marina Checri, Renaud Sirdey, Aymen Boudguiga, and Jean-Paul Bultel. On the practical CPA^{D} security of "exact" and threshold FHE schemes and libraries. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024,*

Part III, volume 14922 of *LNCS*, pages 3–33. Springer, Cham, August 2024.

DM15. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, Berlin, Heidelberg, April 2015.

DMPS24. Nir Drucker, Guy Moshkowich, Tomer Pelleg, and Hayim Shaul. BLEACH: Cleaning errors in discrete computations over CKKS. *Journal of Cryptology*, 37(1):3, January 2024.

EGR25. Austin Ebel, Karthik Garimella, and Brandon Reagen. Orion: A fully homomorphic encryption framework for deep learning. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '25, page 734–749, New York, NY, USA, 2025. Association for Computing Machinery.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

Gen09. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

GHS12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 850–867. Springer, Berlin, Heidelberg, August 2012.

GL25. Craig Gentry and Yongwoo Lee. Fully homomorphic encryption for matrix arithmetic. Cryptology ePrint Archive, Report 2025/1935, 2025.

GNSJ24. Qian Guo, Denis Nabokov, Elias Suvanto, and Thomas Johansson. Key recovery attacks on approximate homomorphic encryption with non-worst-case noise flooding countermeasures. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.

GSW13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Berlin, Heidelberg, August 2013.

GV25. Robin Geelen and Frederik Vercauteren. Fully homomorphic encryption for cyclotomic prime moduli. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part III*, volume 15603 of *LNCS*, pages 366–397. Springer, Cham, May 2025.

GZ25. Mingyu Gao and Hongren Zheng. A critique on average-case noise analysis in rlwe-based homomorphic encryption. In *Proceedings of the 13th Workshop on Encrypted Computing & Applied Homomorphic Computing*, WAHC '25, page 26–37, New York, NY, USA, 2025. Association for Computing Machinery.

HS14. Shai Halevi and Victor Shoup. Algorithms in HElib. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 554–571. Springer, Berlin, Heidelberg, August 2014.

JPK+24. Jae Hyung Ju, Jaiyoung Park, Jongmin Kim, Minsik Kang, Donghwan Kim, Jung Hee Cheon, and Jung Ho Ahn. NeuJeans: Private neural network inference with joint optimization of convolution and FHE bootstrapping. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 4361–4375. ACM Press, October 2024.

KDE+24. Andrey Kim, Maxim Deryabin, Jieun Eom, Rakyong Choi, Yongwoo Lee, Whan Ghang, and Donghoon Yoo. General bootstrapping approach for rlwe-based homomorphic encryption. *IEEE Transactions on Computers*, 73(1):86–96, 2024.

Kim25a. Jaehyung Kim. Bootstrapping GBFV with CKKS. Cryptology ePrint Archive, Report 2025/888, 2025.

Kim25b. Jaehyung Kim. Efficient homomorphic integer computer from CKKS. *IACR TCHES*, 2025(4):873–898, 2025.

Kim25c. Jaehyung Kim. Faster homomorphic integer computer. Cryptology ePrint Archive, Report 2025/1440, 2025.

KN25. Jaehyung Kim and Taeyeong Noh. Modular reduction in CKKS. *IACR Communications in Cryptology*, 2(2), 2025.

KPP22. Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. Approximate homomorphic encryption with reduced approximation error. In Steven D. Galbraith, editor, *CT-RSA 2022*, volume 13161 of *LNCS*, pages 120–144. Springer, Cham, March 2022.

KPZ21. Andrey Kim, Yuriy Polyakov, and Vincent Zucca. Revisiting homomorphic encryption schemes for finite fields. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 608–639. Springer, Cham, December 2021.

KSS24. Jaehyung Kim, Jinyeong Seo, and Yongsoo Song. Simpler and faster BFV bootstrapping for arbitrary plaintext modulus from CKKS. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 2535–2546. ACM Press, October 2024.

LHH+21. Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. Pegasus: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1057–1073. IEEE, 2021.

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 Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 618–647. Springer, Cham, October 2021.

LM21. Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 648–677. Springer, Cham, October 2021.

LMK+23. Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient FHEW bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 227–256. Springer, Cham, April 2023.

LMP22. Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 130–160. Springer, Cham, December 2022.

LMSS22. Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. Securing approximate homomorphic encryption using differential privacy. In

Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 560–589. Springer, Cham, August 2022.

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Berlin, Heidelberg, May / June 2010.

LPR13. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, Berlin, Heidelberg, May 2013.

MML+23. Johannes Mono, Chiara Marcolla, Georg Land, Tim Güneysu, and Najwa Aaraj. Finding and evaluating parameters for BGV. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23*, volume 14064 of *LNCS*, pages 370–394. Springer, Cham, July 2023.

Par25. Jai Hyun Park. Ciphertext-ciphertext matrix multiplication: Fast for large matrices. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VIII*, volume 15608 of *LNCS*, pages 153–180. Springer, Cham, May 2025.

PZZ25. Chris Peikert, Doron Zarchy, and Guy Zyskind. High-precision exact FHE made simple, general, and fast. Cryptology ePrint Archive, Paper 2025/2321, 2025.

RAD78. Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

SSTX09. Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 617–635. Springer, Berlin, Heidelberg, December 2009.

SV10. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 420–443. Springer, Berlin, Heidelberg, May 2010.

vDGHV10. Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 24–43. Springer, Berlin, Heidelberg, May / June 2010.

Zam22. Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. https://github.com/zama-ai/tfhe-rs.

A Additional Preliminaries

A.1 Rings and their relations

Isomorphisms in matrix form. In the RLWE ring $\mathcal{R}_{\mathbb{R}}$, fix a polynomial $m(X)$ and identify it as a vector $\boldsymbol{m} = (m_0, m_1, \ldots, m_{N-1}) \in \mathbb{R}^N$, σ can be understood

as a $(N/2) \times N$ Vandermonde matrix $\boldsymbol{U}_\sigma$ such that $\boldsymbol{z} = \boldsymbol{U}_\sigma \boldsymbol{m}$ with $\boldsymbol{z} \in \mathbb{C}^{N/2}$.

$$\boldsymbol{U}_\sigma = \begin{bmatrix} 1 & \zeta_0 & \zeta_0^2 & \cdots & \zeta_0^{N-1} \\ 1 & \zeta_1 & \zeta_1^2 & \cdots & \zeta_1^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta_{N/2-1} & \zeta_{N/2-1}^2 & \cdots & \zeta_{N/2-1}^{N-1} \end{bmatrix}$$

Consider the $N \times N$ matrix $\mathrm{CRT}_\sigma = \left(\boldsymbol{U}_\sigma \,;\, \overline{\boldsymbol{U}_\sigma}\right)$, its inverse is $\mathrm{CRT}_\sigma^{-1} = \frac{1}{N}\overline{\mathrm{CRT}_\sigma}^T$. Since we have $(\boldsymbol{z} \,;\, \overline{\boldsymbol{z}}) = \mathrm{CRT}_\sigma \cdot \boldsymbol{m}$, the inverse transform can be evaluated as

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

We can further divide $\boldsymbol{U}_\sigma$ into two $(N/2) \times (N/2)$ square matrices $\boldsymbol{U}_\sigma = \begin{bmatrix} \boldsymbol{U}_{\sigma,0} & \boldsymbol{U}_{\sigma,1} \end{bmatrix}$. Note that $\boldsymbol{U}_{\sigma,1} = \sqrt{-1} \cdot \boldsymbol{U}_{\sigma,0}$. We use the notation $\boldsymbol{V}_{\sigma,0} = \overline{\boldsymbol{U}_{\sigma,0}}^T$ and $\boldsymbol{V}_{\sigma,1} = \overline{\boldsymbol{U}_{\sigma,1}}^T$. Let $\boldsymbol{m} = (\boldsymbol{m}_0 \,;\, \boldsymbol{m}_1)$ where $\boldsymbol{m}_0 = (m_0, m_1, \ldots, m_{N/2-1}) \in \mathbb{R}^{N/2}$ and $\boldsymbol{m}_1 = (m_{N/2}, \ldots, m_{N-1}) \in \mathbb{R}^{N/2}$, we have

$$\begin{aligned} \boldsymbol{m}_0 &= \frac{2}{N}\Re\left[\boldsymbol{V}_{\sigma,0} \cdot \boldsymbol{z}\right] = \frac{1}{N}\left(\boldsymbol{V}_{\sigma,0} \cdot \boldsymbol{z} + \overline{\boldsymbol{V}_{\sigma,0} \cdot \boldsymbol{z}}\right) \\ \boldsymbol{m}_1 &= \frac{2}{N}\Im\left[\boldsymbol{V}_{\sigma,0} \cdot \boldsymbol{z}\right] = \frac{-\sqrt{-1}}{N}\left(\boldsymbol{V}_{\sigma,0} \cdot \boldsymbol{z} - \overline{\boldsymbol{V}_{\sigma,0} \cdot \boldsymbol{z}}\right) \\ \boldsymbol{z} &= \boldsymbol{U}_{\sigma,0}(\boldsymbol{m}_0 + \sqrt{-1} \cdot \boldsymbol{m}_1) \end{aligned}$$

In the arithmetic ring $\mathcal{Z}_\mathbb{R}$, fix a polynomial $m(X)$ and identify it as a vector $\boldsymbol{m} = (m_0, m_1, \ldots, m_{n-1})$. τ can be understood as $(n/2) \times n$ Vandermonde matrix $\boldsymbol{U}_\tau$ such that $\boldsymbol{z} = \boldsymbol{U}_\tau \boldsymbol{m}$ with $\boldsymbol{z} \in \mathbb{C}^{n/2}$.

$$\boldsymbol{U}_\tau = \begin{bmatrix} 1 & \xi_0 & \xi_0^2 & \cdots & \xi_0^{n-1} \\ 1 & \xi_1 & \xi_1^2 & \cdots & \xi_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \xi_{n/2-1} & \xi_{n/2-1}^2 & \cdots & \xi_{n/2-1}^{n-1} \end{bmatrix}$$

For $\mathrm{CRT}_\tau = \left(\boldsymbol{U}_\tau \,;\, \overline{\boldsymbol{U}_\tau}\right)$, we denote $\tau^{-1} = \mathrm{CRT}_\tau^{-1}$. Vandermonde matrix theory tells us that the k-th column of the inverse τ^{-1} has the coefficients of the k-th Lagrange polynomial

$$\ell_k(x) = \frac{1}{n\xi_k^{n-1} - 1}\left(-1 + \xi_k^{n-1} + \sum_{i=1}^{n-1} \xi_k^{n-i-1} X^i\right)$$

Let $\ell_k(X) = \sum_{i=0}^{n-1} \ell_{k,i} \cdot X^i$, τ^{-1} can be expressed as $\tau^{-1} = \begin{bmatrix} \boldsymbol{V}_\tau & \overline{\boldsymbol{V}_\tau} \end{bmatrix}$ where $\boldsymbol{V}_\tau$ is a $n \times (n/2)$ matrix

$$\boldsymbol{V}_\tau = \begin{bmatrix} \ell_{0,0} & \ell_{1,0} & \cdots & \ell_{n/2-1,0} \\ \ell_{0,1} & \ell_{1,1} & \cdots & \ell_{n/2-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ \ell_{0,n-1} & \ell_{1,n-1} & \ldots & \ell_{n/2-1,n-1} \end{bmatrix}$$

This gives us the following relation between $\boldsymbol{z}$ and $\boldsymbol{m}$.

$$\boldsymbol{m} = \boldsymbol{V}_\tau \cdot \boldsymbol{z} + \overline{\boldsymbol{V}_\tau \cdot \boldsymbol{z}} \qquad \boldsymbol{z} = \boldsymbol{U}_\tau \cdot \boldsymbol{m}$$

If we divide them into $(n/2) \times (n/2)$ square matrices $\boldsymbol{U}_\tau = \begin{bmatrix}\boldsymbol{U}_{\tau,0} & \boldsymbol{U}_{\tau,1}\end{bmatrix}$ and $\boldsymbol{V}_\tau = (\boldsymbol{V}_{\tau,0}\,;\boldsymbol{V}_{\tau,1})$, we have

$$\boldsymbol{m}_0 = 2\Re\left[\boldsymbol{V}_{\tau,0} \cdot \boldsymbol{z}\right] \qquad \boldsymbol{m}_1 = 2\Re\left[\boldsymbol{V}_{\tau,1} \cdot \boldsymbol{z}\right]$$
$$\boldsymbol{z} = \boldsymbol{U}_{\tau,0} \cdot \boldsymbol{m}_0 + \boldsymbol{U}_{\tau,1} \cdot \boldsymbol{m}_1$$

Automorphisms. We did not find interesting applications of automorphism for the arithmetic ring $\mathcal{Z}_\mathbb{R}$. We detail the discussion on the RLWE ring $\mathcal{R}$.

Let one possible automorphism index set be $\langle 5, -1\rangle$ for the multiplicative group $\mathbb{Z}_{2N}^*$, let j be an automorphism index, the j-th automorphism is

$$\begin{array}{rccc}\phi_j : & \mathcal{R} & \to & \mathcal{R} \\ & m(X) & \mapsto & m(X^j)\end{array}$$

It is noted that

$$\sigma(\phi_{5^i}(m(X))) = (\ldots, m(\zeta_k^{5^i}), \ldots) = (\ldots, m(\zeta_{k+i \pmod{N/2}}), \ldots)$$
$$\sigma(\phi_{-1}(m(X))) = (\ldots, m(\zeta_k^{-1}), \ldots) = (\ldots, \overline{m(\zeta_k)}, \ldots)$$

which means automorphism indices of form 5^i gives rotation while -1 gives conjugation in the canonical embedding.

Subring and Sparse Packing. We mainly discuss the subring structure for $\mathcal{R} = \mathbb{Z}[X]/\langle X^N + 1\rangle$. It is noted that the ring $\mathcal{R}' = \mathbb{Z}[Y]/\langle Y^\ell + 1\rangle$ can be embedded into $\mathcal{R}$ using $Y = X^{N/\ell}$ when ℓ is a divisor of N. For $\mathcal{R}'$, one primitive root is $\zeta' = \exp\left(2\pi\sqrt{-1}/(2\ell)\right) = \zeta^{N/\ell}$. We can again define a $\mathcal{R}'$-*canonical embedding* σ' and identify an element in $\mathcal{R}'$ with a vector in $\mathbb{C}^{\ell/2}$ using $\zeta'_j = \zeta'^{5^j}$ for $0 \le j < \ell/2$. Now for a $m(Y) \in \mathcal{R}'$ with $\sigma'(m) = \boldsymbol{w} = (w_j)_{0\le j<\ell/2}$, its embedding $\tilde{m}(X) = m(X^{N/\ell})$ in $\mathcal{R}$ has the canonical embedding $\boldsymbol{z} = (z_j)_{0\le j<N/2} = \sigma(\tilde{m})$ as

$$z_j = \tilde{m}(\zeta_j) = m(\zeta_j^{N/\ell}) = m(\zeta'_{j \pmod{\ell/2}}) = w_{j \pmod{\ell/2}} \quad, 0 \le j < N/2$$

so $\boldsymbol{z}$ can be understood as a concatenation of N/ℓ copies of $\boldsymbol{w}$.

Conversely, given a $\tilde{m}(X)$ in $\mathcal{R}$, Galois theory tells us that if we sum over all automorphisms that fixes $\mathcal{R}' \subset \mathcal{R}$, we obtain a $m(X^{N/\ell})$ that can be interpreted as $m(Y) \subset \mathcal{R}'$. This is the field trace function and by legacy we call it the PartialSum function [CHK+18]. Such process can also be viewed as turning the canonical embedding $\boldsymbol{z}$ of $\tilde{m}(X)$ into a concatenation of N/ℓ copies of $\boldsymbol{w}$ for some $m(Y)$ using $\log(N/\ell)$ rotations.

The relevance of subring structure comes from the fact that in certain circumstances we may only want to connect $\mathcal{R}'$ instead of $\mathcal{R}$ with ℓ/n copies of

$\mathcal{Z}_{\mathbb{R}}$, commonly known as *sparse packing*. The linear transformation now, as the $\mathcal{R}' \subset \mathcal{R}$ embedding naturally gives repetition, can be written down as

$$\boldsymbol{m} = \begin{bmatrix}\boldsymbol{m}_0 \\ \boldsymbol{m}_1\end{bmatrix} = \frac{2}{\ell}\Re\left(\begin{bmatrix}\boldsymbol{V}_{\sigma,0} & \\ & \boldsymbol{V}_{\sigma,1}\end{bmatrix} \cdot \begin{bmatrix}\boldsymbol{z} \\ \boldsymbol{z}\end{bmatrix}\right) \qquad \begin{bmatrix}\boldsymbol{z}_0 \\ \boldsymbol{z}_1\end{bmatrix} = \begin{bmatrix}\boldsymbol{U}_{\sigma,0} & \\ & \boldsymbol{U}_{\sigma,1}\end{bmatrix} \cdot \begin{bmatrix}\boldsymbol{m}_0 \\ \boldsymbol{m}_1\end{bmatrix}$$

Further rotation gives $(\boldsymbol{z}_0\,;\,\boldsymbol{z}_1)+(\boldsymbol{z}_1\,;\,\boldsymbol{z}_0) = (\boldsymbol{z}\,;\,\boldsymbol{z})$ and can be viewed as canonical embedding of subring elements. Similar linear transformation for τ can be defined here if we replace every subscript σ with τ (and remove the $\frac{1}{\ell}$ coefficient).

For ease of discussion, we assume *full packing* (i.e., $\ell = N$) throughout this work.

Proof for Prop 1. The operator norm of τ can be found by checking the row-sum of the matrix $\boldsymbol{U}_\tau$, which is $\max_j \sum_{i=0}^{n-1} \xi_j^i = \frac{\xi_j^n - 1}{\xi_j - 1}$. The norm of ξ_j can be understood as $|\xi_j|^n = |\xi_j - 2| \approx 3$, which is reached when ξ_j is near -1, and similarly $\max_j |\xi_j| \approx 1 + \frac{\ln 3}{n}$. Therefore $\|\tau\| \sim \frac{2}{\ln 3} n$.

A.2 RLWE operations

We discuss several interesting *modulo* operations we can do on the ciphertext. They are the cornerstone of various management and bootstrapping operations.

Definition 20 (RLWE Rescaling). *Given* $\mathsf{ct} = (b, a) = (-a \cdot s + m + e, a) \in \mathcal{R}_{Q_L}^2$. *Suppose* $Q_L = Q_{L-1} q_L$, *we can rescale from* $\mathcal{R}_{Q_L}$ *to* $\mathcal{R}_{Q_{L-1}}$ *by the following operation*

$$\mathsf{ct}' = \mathsf{RLWE.Rescale}_{q_L}(\mathsf{ct}) = \frac{\mathsf{ct} + \delta}{q_L} \qquad \text{with} \qquad \delta = ([-b]_{q_L}, [-a]_{q_L})$$

We note the effect on the message as follows. In $\mathcal{R}$, we have $\langle \delta, (1, s) \rangle = -[m + e]_{q_L} + q_L I$, so it follows that

$$\mathsf{RLWE.Decrypt}_s(\mathsf{ct}') = \left\lfloor \frac{m + e}{q_L} \right\rceil + I \pmod{Q_{L-1}}$$

We call I the rescaling noise and it follows the same distribution as in Prop 2 with hamming weight $h = 192$.

We now discuss the arithmetic of the RLWE ring, and its effect on the message m.

Definition 21 (RLWE Addition and Tensor Product). *Given* $\mathsf{ct} = (b, a) = (-as + m + e, a) \in \mathcal{R}_Q^2$ *and* $\mathsf{ct}' = (b', a') = (-a's + m + e', a') \in \mathcal{R}_Q^2$, *the addition and tensor product operations are*

$$\mathsf{RLWE.Add}(\mathsf{ct}, \mathsf{ct}') = (b + b', a + a') \quad \in \mathcal{R}_Q^2$$
$$\mathsf{RLWE.Tensor}(\mathsf{ct}, \mathsf{ct}') = (b \cdot b', a \cdot b' + a' \cdot b, a \cdot a') \quad \in \mathcal{R}_Q^3$$

where addition gives $m + m'$ *and tensor product gives* $m \cdot m'$. $\mathsf{RLWE.Tensor}(\mathsf{ct}, \mathsf{ct}')$ *should be decrypted using* $(1, s, s^2)$.

Definition 22 (RLWE Scalar Multiplication). *Given* $\mathsf{ct} = (b, a) = (-a \cdot s + m + e, a) \in \mathcal{R}_Q^2$*, for a scalar* $k \in \mathcal{R}_Q$ *or* $k \in \mathbb{Z}_Q$*, we define the scalar multiplication operation*

$$\mathsf{ct}' = \mathsf{RLWE.MultScalar}(\mathsf{ct}, k) = (k \cdot b, k \cdot a) \in \mathcal{R}_Q^2$$

The effect on the message is

$$\mathsf{RLWE.Decrypt}_s(\mathsf{ct}') \equiv [km + ke]_{Q_L} \pmod{Q}$$

We now discuss how automorphisms of $\mathcal{R}$ translates into automorphisms in RLWE encrypted ciphertext.

Definition 23 (RLWE Automorphism). *Given* $\mathsf{ct} = (b, a) = (-as + m + e, a) \in \mathcal{R}_Q^2$*, since* $\mathcal{R}_Q = \mathbb{Z}[X]/\langle X^N + 1, Q\rangle$*, we can also recognize elements as polynomials like* $b(X), a(X), m(X)$*. Given the automorphism index set* $\langle 5, -1\rangle \subset \mathbb{Z}_{2N}^*$*. Let* j *be an automorphism index, we define the* j*-th automorphism as*

$$\begin{aligned}\mathsf{ct}' &= \mathsf{RLWE.Automorphism}_j(\mathsf{ct}) = (b(X^j), a(X^j)) \\ &= (-a(X^j)s(X^j) + m(X^j) + e(X^j), a(X^j))\end{aligned}$$

we note that ct' *is an encryption of* $m(X^j)$ *using* $s(X^j)$*.*

The RLWE tensor product and automorphism operation will change the decryption basis, which will propagate along the computation. To ease the management, the key switching technique was proposed [BV11]. It is observed that for an element $d' \in \mathcal{R}_Q$ that should be decrypted using s', we can *homomorphically* decrypt it using $\mathsf{RLWE.Encrypt}_s(s')$ like

$$\begin{aligned}d &= d' \cdot \mathsf{RLWE.Encrypt}_s(s') = \mathsf{RLWE.Encrypt}_s(d's') \\ &= (-as + d's' + d'e, a)\end{aligned}$$

so that d can be now decrypted using s. On the other hand, the direct production of $d \in \mathcal{R}_Q$ may cause a large $d'e$ noise term, so gadget decomposition techniques were proposed to reduce the growth of the noise.

Definition 24 (RLWE Key Switching). *Given* $\mathsf{ct}_{s'} = (b, a) = (-as' + m + e, a)$ *that should be decrypted using* $(1, s')$*, the key switching operation is*

$$\mathsf{ct}_s = \mathsf{RLWE.KeySwitch}_{s' \to s}(\mathsf{ct}_{s'}) = (b, 0) + \mathcal{D}(a) \cdot \mathsf{RLWE.Encrypt}_s(\mathcal{P}(s'))$$

where $\mathcal{D}(\cdot)$ *and* $\mathcal{P}(\cdot)$ *is a pair of decomposition and power-up function such that* $\mathcal{D}(x) \cdot \mathcal{P}(y) = x \cdot y \in \mathcal{R}_Q$*. There exists a pair of* $\mathcal{D}$ *and* $\mathcal{P}$ *such that the additional noise caused is nearly the same as the RLWE rescaling operation.*

The $\mathsf{RLWE.Encrypt}_s(\mathcal{P}(s'))$ *is the evaluation key.*

Typical construction and analysis of $\mathcal{D}$ and $\mathcal{P}$ can be found in [KPZ21,MML+23]. It is further noted that the evaluation key is an encryption of the secret key itself, thus requiring an additional assumption called circular secure assumption.

A.3 CKKS operations

Definition 25 (CKKS Rotation and Conjugation). *Given a polynomial $m(X) \in \mathcal{R}_Q$ that encodes $\boldsymbol{z} = (z_0, z_1, \ldots, z_{N/2-1}) \in \mathbb{C}^{N/2}$, for rotation index i, we define the automorphism index to be $j = 5^i \pmod{2N}$, then we define the rotation and conjugation operation to be*

$$m'(X) = \mathsf{CKKS.Rotate}_i(m(X)) = m(X^j)$$
$$m''(X) = \mathsf{CKKS.Conjugate}(m(X)) = m(X^{-1})$$

where $m(X^j)$ can be homomorphically evaluated using $\mathsf{RLWE.Automorphism}_j$. Note that the k-th slot of $m'(X)$ is now the $(k+i)$-th (modulo $N/2$) slot of $m(X)$, and the k-th slot of $m''(X)$ is the conjugate of the k-th slot of $m(X)$.

Definition 26 (CKKS Sine Operation). *For a CKKS encoded data $qI + \Delta z + e \in \mathbb{C}$ where $\Delta z + e \lll q$, $I \in \mathbb{Z}$ and $|I| \le K$, let $\epsilon = \frac{\Delta z + e}{q}$, there exists a polynomial that evaluates*

$$\begin{aligned}\mathsf{CKKS.Sine}\left(qI + \Delta z + e\right) &= \frac{q}{2\pi} \sin\left(\frac{2\pi}{q}(qI + \Delta z + e)\right) \approx q\left(\epsilon + O(|\epsilon|^3)\right) + e' \\ &= \Delta z + e + qO(|\epsilon|^3) + e'\end{aligned}$$

where $O(|\epsilon|^3)$ is the error from approximating x using $\sin(x)$, and e' consists of noises like polynomial approximation error and multiplication noises.

Using automorphisms in $\mathcal{R}$, we can construct operations like $\mathsf{CKKS.Rotate}$ and $\mathsf{CKKS.Conjugate}$ (Def. 25). With them, we can efficiently evaluate linear transformations. We present a simple Halevi-Shoup style linear transformation [HS14] and there are many optimizations available like using Fast Fourier Transform [CCS19].

Definition 27 (CKKS Linear Transformation). *For a matrix $\boldsymbol{A} \in \mathbb{C}^{N/2 \times N/2}$, given a polynomial $m(X) \in \mathcal{R}_Q$ that encodes $\boldsymbol{z} = (z_0, z_1, \ldots, z_{N/2-1}) \in \mathbb{C}^{N/2}$, there exists an algorithm to evaluate the matrix multiplication*

$$\begin{aligned}\mathsf{CKKS.Ecd}_{\Delta,\mathcal{R}_Q}\left(\boldsymbol{A} \cdot \boldsymbol{z}\right) &= \mathsf{CKKS.LT}_{\boldsymbol{A}}(m(X)) \\ &= \sum_{0 \le i < N/2} u_i(X) \cdot \mathsf{CKKS.Rotate}_i(m(X)) \\ \text{with} \qquad u_i(X) &= \mathsf{CKKS.Ecd}_{\Delta,\mathcal{R}_Q}(\mathrm{diag}_i(\boldsymbol{A}))\end{aligned}$$

where $\mathrm{diag}_i(\boldsymbol{A})$ is the i-th diagonal $(A_{0,i}, A_{1,i+1}, \ldots)$ of $\boldsymbol{A}$.

B Boolean Flattened Mode

Now assume we are able to extract bits from $\mathcal{Z}_{\mathbb{R}}$ encoded polynomial to $\mathbb{C}^n$ via bootstrapping. This is exactly discrete CKKS with plaintext modulus of 2. For illustration purpose, we define some example boolean operations as follows.

Definition 28 (Boolean Encoding in $\mathbb{C}$). *For a message $m \in \mathbb{Z}_{2^n}$ such that $m = \sum_{i=0}^{n-1} b_i 2^i$, let $\boldsymbol{b} = (b_0, b_1, \ldots, b_{n-1})$, its encoding in $\mathbb{C}^n$ is*

$$\mathbb{C}.\mathsf{Ecd}_{\Delta}(m) = \Delta \boldsymbol{b}$$

Definition 29 (Boolean Bitwise Operations in $\mathbb{C}$). *For two messages $\Delta \boldsymbol{b} = \mathbb{C}.\mathsf{Ecd}_{\Delta}(m)$ and $\Delta \boldsymbol{b}' = \mathbb{C}.\mathsf{Ecd}_{\Delta}(m')$, their boolean operation can be achieved by*

$$\begin{aligned}
\mathsf{AND} &= \frac{1}{\Delta} \cdot \Delta \boldsymbol{b} \cdot \Delta \boldsymbol{b}' \\
\mathsf{OR} &= \Delta \boldsymbol{b} + \Delta \boldsymbol{b}' - \frac{1}{\Delta} \cdot \Delta \boldsymbol{b} \cdot \Delta \boldsymbol{b}' \\
\mathsf{XOR} &= \Delta \boldsymbol{b} + \Delta \boldsymbol{b}' - \frac{2}{\Delta} \cdot \Delta \boldsymbol{b} \cdot \Delta \boldsymbol{b}' \\
\mathsf{NOT} &= \Delta \cdot 1 - \Delta \boldsymbol{b}
\end{aligned}$$

Other bitwise boolean operations like NAND *can be achieved similarly.*

The ability of CKKS.Rotate to rotate the $\mathbb{C}$-slots naturally gives us the ability to shift and rotate the bits.

Definition 30 (Bit Rotation/Shifting Operations in $\mathbb{C}$). *For a vector $\Delta \boldsymbol{b} = \Delta(b_0, b_1, \ldots, b_{n-1}) \in \mathbb{C}^n$, if we are given rotation operation and arithmetic operations for $\mathbb{C}^n$, we can achieve bit rotation and bit shifting via masking, rotation and recombination.*

The masking and recombination can be implemented using AND and OR.

Definition 31 (Sign extraction in $\mathbb{C}$). *Given an encoding of m in $\mathbb{C}^n$ as $\mathbb{C}.\mathsf{Ecd}_{\Delta}(m) = \Delta \boldsymbol{b} = \Delta(b_0, b_1, \ldots, b_{n-1})$, the extraction of the sign of m is the extraction of b_{n-1} and can be done through masking and bit shifting.*

With sign extraction, further operations like comparison can be implemented using subtraction in arithmetic mode and sign extraction in boolean mode.

C Whole Picture of Our Scheme

We summarize operations needed and offered by our scheme for ease of reference.

- RLWE namespace: KeyGen, Encrypt, Decrypt, Add, Tensor, Automorphism, KeySwitch, ModRaise, Rescale, Truncate.
- CKKS namespace: Ecd : $\mathbb{C}^{N/2} \to \mathcal{R}_Q$, Dcd, Sine, Rotate, Conjugate, LT, Add, Mult
- D-CKKS namespace: Ecd : $\mathbb{Z}_p \to \mathbb{C}$, Dcd, LUT.
- Arithmetic $\mathcal{Z}_{\mathbb{R}}$ namespace: Ecd : $\mathbb{Z}_{2^n} \to \mathcal{Z}_{\mathbb{R}}$, Dcd, Add, Mult, MultShort.
- Many arithmetic $\#\mathcal{Z}_{\mathbb{R}}$ namespace: Rotate.
- Homomorphism namespace: Embed : $\mathcal{Z}_{\mathbb{R}} \to \mathcal{R}_Q$ and Recover : $\mathcal{R}_Q \to \mathcal{Z}_{\mathbb{R}}$.

- Boolean $\mathbb{C}$ namespace: Ecd : $\mathbb{Z}_{2^n} \to \mathbb{C}^n$, Dcd, AND, OR, NOT, bit shifting/rotation, sign extraction, and many others. Boolean-To-Arithmetic conversion (B2A).
- Bootstrapping namespace: Arithmetic-To-Arithmetic (A2A) for I, A2A e, Arithmetic-To-Boolean (A2B), Batched A2B, Boolean-To-Boolean (B2B).

Each Add and Mult have ciphertext-ciphertext and ciphertext-plaintext versions; boolean operations with two operands also have two versions. Ciphertext-plaintext $\mathcal{Z}_{\mathbb{R}}$.Mult is implemented using $\mathcal{Z}_{\mathbb{R}}$.MultShort.

D Additional Illustrations

D.1 Arithmetic Triangle Encoding after Multiplication by t

If we multiply the arithmetic triangle encoding (Fig. 1) by $t = X - 2$, we obtained a binary representation but the bits are mixed with tI which is hard to separate without consulting iterative process.

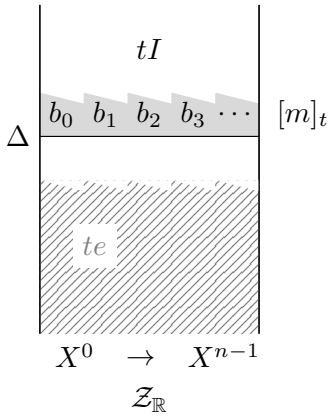


Fig. 6. Arithmetic triangle encoding after multiplication by t.

D.2 Encoding/Decoding and Homomorphic Encoding/Decoding

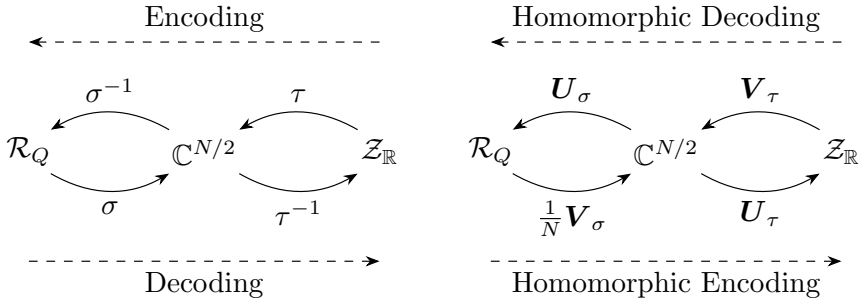


Fig. 7. Encoding/Decoding and Homomorphic Encoding/Decoding. Note that Homomorphic Encoding/Decoding actually evaluate the Decoding and Encoding.

D.3 Arithmetic-To-Arithmetic Refreshing

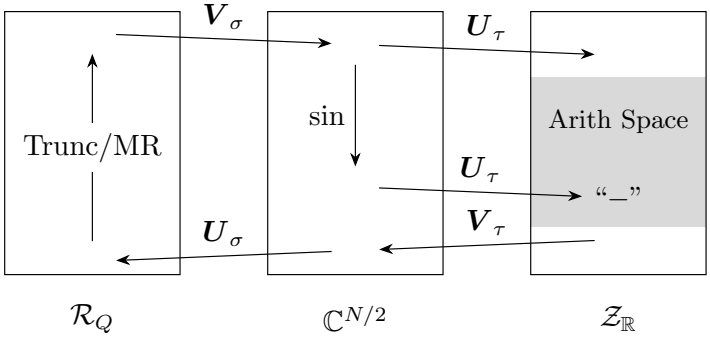


Fig. 8. The Arithmetic-To-Arithmetic Refreshing

D.4 Arithmetic-To-Boolean Conversion

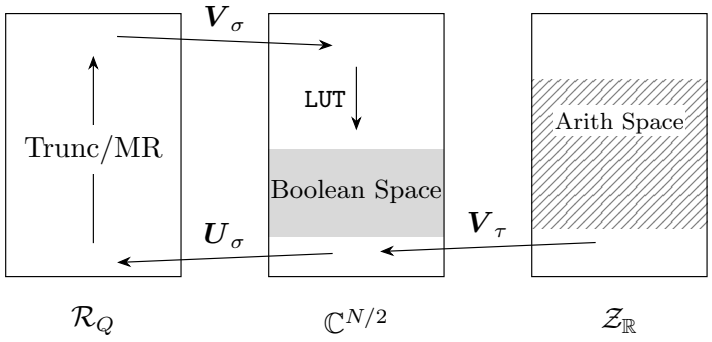


Fig. 9. The Arithmetic-To-Boolean Conversion

D.5 Boolean-To-Boolean and Boolean-To-Arithmetic Conversion

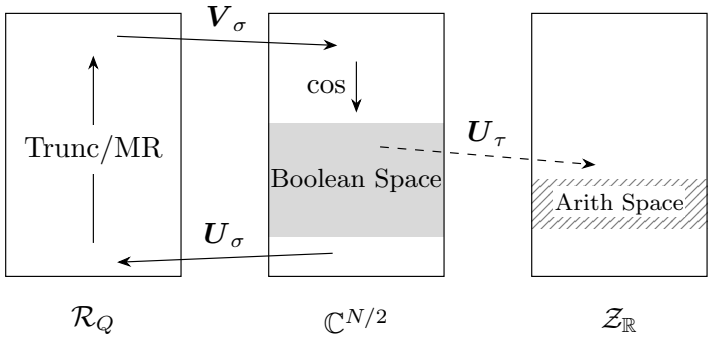


Fig. 10. The Boolean-To-Boolean Refreshing (solid arrows) and the Boolean-To-Arithmetic Conversion (dashed arrow).

E Detailed Proof for Theorem 4

We first examine the evolution of polynomial norm in $\mathcal{Z}$:

Proposition 4 (Expansion in $\mathcal{Z}$). *For $f, g \in \mathcal{Z}_\mathbb{R}$ and $t = X - 2$, we have*

$$\begin{aligned} \|f + g\| &\le \|f\| + \|g\| \\ \|f \cdot g\| &\le 3n \cdot \|f\| \cdot \|g\| \\ \|t \cdot f\| &\le 4 \|f\| \end{aligned}$$

Proof. The proof for $\|f+g\|$ is straightforward from the triangle inequality. The proof for $\|f \cdot g\|$ follows directly from [BFG+26, Proposition 3.1]. For $\|t \cdot f\|$, let $f = \sum_{i=0}^{n-1} f_i X^i$, we have

$$\begin{aligned} t \cdot f &= \sum_{i=0}^{n-1} f_i X^{i+1} - 2\sum_{i=0}^{n-1} f_i X^i \\ &= \sum_{i=1}^{n-1} f_{i-1} X^i - 2\sum_{i=0}^{n-1} f_i X^i + f_{n-1}(X-2) \end{aligned}$$

The constant coefficient is $-2f_0 - 2f_{n-1}$ and the coefficient for X is $(f_0 - 2f_1 + f_{n-1})$. Therefore, $\|t \cdot f\| \le 4\|f\|$.

The $3n$ here is the *worst-case* expansion factor in $\mathcal{Z}_{\mathbb{R}}$.

Theorem 7 (Arithmetic Multiplication Noise Growth (Restated)). *For two messages $f = \Delta I + \frac{\Delta}{t}[m]_t + v$ and $f' = \Delta I' + \frac{\Delta}{t}[m']_t + v'$, the noise v'' and the overflow I'' after $\mathcal{Z}_{\mathbb{R}}$.*Mult *satisfies*

$$\begin{aligned} \|v''\| &\le 3n\left[(4\|I\|+1)\cdot\|v'\| + (4\|I'\|+1)\cdot\|v\| + \frac{4\|v\|\,\|v'\|}{\Delta}\right] \\ \|I''\| &\le 3n\,(4\|I\|\cdot\|I'\| + \|I\| + \|I'\| + 1) + 1 \end{aligned}$$

*For the multiplication of f and $g = \Delta[m_g]_t + v_g$, the noise v''' and the overflow I''' after $\mathcal{Z}_{\mathbb{R}}$.*MultShort *satisfies*

$$\begin{aligned} \|v'''\| &\le 3n\left[(\|I\|+1)\cdot\|v_g\| + \|v\| + \frac{\|v\|\,\|v_g\|}{\Delta}\right] \\ \|I'''\| &\le 3n\,(\|I\|+1) + 1 \end{aligned}$$

*Proof. $\mathcal{Z}_{\mathbb{R}}$.*Mult *can be expressed as*

$$\begin{aligned} \frac{t}{\Delta} f \cdot f' &= \frac{t}{\Delta}(\Delta I + \frac{\Delta}{t}[m]_t + v)(\Delta I' + \frac{\Delta}{t}[m']_t + v') \\ &= \Delta(tI\cdot I' + I\cdot[m']_t + I'\cdot[m]_t) \\ &\quad + \frac{\Delta}{t}[m]_t\cdot[m']_t \\ &\quad + tI\cdot v' + tI'\cdot v + [m]_t\cdot v' + [m']_t\cdot v + \frac{t}{\Delta}v\cdot v' \\ &= \Delta I'' + \frac{\Delta}{t}[m'']_t + v'' \end{aligned}$$

with

$$\begin{aligned} I'' &= tI\cdot I' + I\cdot[m']_t + I'\cdot[m]_t + C, \\ [m'']_t &= [m\cdot m']_t, \\ [m]_t\cdot[m']_t &= [m'']_t + tC \\ v'' &= t(I\cdot v' + I'\cdot v) + [m]_t\cdot v' + [m']_t\cdot v + \frac{t}{\Delta}v\cdot v'. \end{aligned}$$

We have $\|[m]_t\| \leq 1$ *and* $\left\|\frac{[m]_t}{t}\right\| \leq 1$ *from Def. 9 and Def. 10. For* $\|C\|$*, we have*

$$\|C\| = \left\|\frac{[m]_t \cdot [m']_t - [m'']_t}{t}\right\| \leq \left\|\frac{[m]_t \cdot [m']_t}{t}\right\| + \left\|\frac{[m'']_t}{t}\right\| \leq 3n + 1$$

By using Prop. 4, we obtain the formula for $\|I''\|$ *and* $\|v''\|$*.*

Similarly, for $\mathcal{Z}_{\mathbb{R}}.\mathsf{MultShort}$*, we have*

$$\begin{aligned}\frac{1}{\Delta} f \cdot g &= \frac{1}{\Delta}(\Delta I + \frac{\Delta}{t}[m]_t + v)(\Delta[m_g]_t + v_g) \\ &= \Delta(I \cdot [m_g]_t) + \frac{\Delta}{t}[m]_t \cdot [m_g]_t + [m_g]_t \cdot v + I \cdot v_g + \frac{1}{t}[m]_t \cdot v_g + \frac{1}{\Delta} v \cdot v_g \\ &= \Delta I''' + \frac{\Delta}{t}[m''']_t + v'''\end{aligned}$$

with

$$\begin{aligned} I''' &= I \cdot [m_g]_t + C', \\ [m''']_t &= [m \cdot m_g]_t, \\ [m]_t \cdot [m_g]_t &= [m''']_t + tC' \\ v''' &= I \cdot v_g + [m_g]_t \cdot v + \frac{1}{t}[m]_t \cdot v_g + \frac{1}{\Delta} v \cdot v_g. \end{aligned}$$

We have $\|C'\| \leq 3n + 1$ *and therefore obtain the formula for* $\|I'''\|$ *and* $\|v'''\|$*.*

F Detailed Algorithmic Listings for Arithmetic-To-Boolean Conversions

F.1 Arith-To-Boolean

F.2 Batched Arith-To-Boolean Conversion

Algorithm 1 A2B: Arithmetic-to-Boolean conversion for full packing. The treatment of the two ciphertexts inside the core ciphertext is not shown.

Input: Arithmetic ciphertext ct, word size n, chunk width w satisfying $w|n$, cutoff parameter $\mathsf{A2B\text{-}cutoff}$

Output: Two boolean mode ciphertexts ret_0 and ret_1 satisfying the "breaking-into-halves" effect

$\mathsf{core} \leftarrow \mathsf{Z2C}_{\boldsymbol{V}_\tau, \mathsf{special}\ b_0}(\mathsf{ct})$ ▷ $\mathsf{special}$-b_0 changes the first row of $\boldsymbol{V}_\tau$
▷ Core ciphertext contains two ciphertexts due to "breaking-into-halves"
$d \leftarrow n/w$
ret_0 and ret_1 are empty CKKS ciphertexts
for $\mathsf{iter} = 0$ to $d - 1$ **do**
  Prepare CKKS plaintext $\mathsf{Mask}_{\mathsf{iter}}$
    ▷ Slots have value "1" for the $[\mathsf{iter} \cdot w, (\mathsf{iter} + 1) \cdot w)$-th bits for each word
  $\mathsf{ct}_{\mathsf{target}} \leftarrow \mathsf{CKKS.Mult}(\mathsf{core}, \mathsf{Mask}_{\mathsf{iter}})$
  $(\mathsf{ct}_{\mathsf{ID}}, \mathsf{ct}_{\mathsf{MSB}}) \leftarrow \mathsf{D\text{-}CKKS.LUT}_{\mathsf{ID},\mathsf{MSB}}(\mathsf{ct}_{\mathsf{target}})$ ▷ LUT size 2^w
  $\mathsf{ret} \leftarrow \mathsf{CKKS.Add}(\mathsf{ret}, \mathsf{ct}_{\mathsf{MSB}})$ ▷ Update ret_0 or ret_1 depending on iter
  for $\mathsf{nextIter} = \mathsf{iter}$ to $d - 1$ **do**
    $\rho \leftarrow (\mathsf{nextIter} - \mathsf{iter}) \cdot w$
    if $-\rho \leq \mathsf{A2B\text{-}Cutoff}$ **then**
      break ▷ Ensuring $O(n)$ rotations
    end if
    $\mathsf{ct}_{\mathrm{scaled}} \leftarrow \mathsf{CKKS.Mult}(\mathsf{ct}_{\mathsf{ID}}, 2^{-\rho})$
    $\mathsf{ct}_{\mathrm{scaled}} \leftarrow \mathsf{CKKS.Rotate}(\mathsf{ct}_{\mathrm{scaled}}, -\rho)$
    $\mathsf{core} \leftarrow \mathsf{CKKS.Sub}(\mathsf{core}, \mathsf{ct}_{\mathrm{scaled}})$
  end for
end for
return $\{\mathsf{ret}_0, \mathsf{ret}_1\}$

Algorithm 2 B-A2B: Batched Arithmetic-to-Boolean conversion for full packing. Some details are omitted for simplicity, such as the treatment of the two ciphertexts inside the core ciphertext and the management of $\mathsf{ret}_{i,0}$ and $\mathsf{ret}_{i,1}$.

Input: ALU word size n, chunk width w satisfying $w|n$, n/w numbers of arithmetic ciphertext ct_i $(0 \le i < n/w)$, cutoff parameter $\mathsf{A2B\text{-}cutoff}$

Output: $2n/w$ boolean mode ciphertexts $\{\mathsf{ret}_{i,0}, ret_{i,1}\}$ satisfying the "breaking-into-halves" effect

$d \leftarrow n/w$
for $i = 0$ to $d-1$ **do**
  $\mathsf{core}_i \leftarrow \mathsf{Z2C}_{\boldsymbol{V}_\tau, \mathsf{special}\ b_0}(\mathsf{ct}_i)$ ▷ special-b_0 changes the first row of $\boldsymbol{V}_{\tau,0}$
end for
for $j = 1$ to $d/2-1$ **do** ▷ Pre-rotate
  $\mathsf{core}_j \leftarrow \mathsf{Rotate}(\mathsf{core}_j, -j \cdot w)$
  $\mathsf{core}_{j+d/2} \leftarrow \mathsf{Rotate}(\mathsf{core}_{j+d/2}, -j \cdot w)$
end for
$\mathsf{ret}_{i,0}$ and $\mathsf{ret}_{i,1}$ are empty CKKS ciphertexts for all i from 0 to $d-1$
for $\mathsf{iter} = 0$ to $d-1$ **do**
  Prepare CKKS plaintext $\mathsf{Mask}_{j,\mathsf{iter}}$
    ▷ j refers to the pre-rotation amount.
    ▷ Slots have value "1" for the $[\mathsf{iter} \cdot w, (\mathsf{iter}+1) \cdot w)$-th bits for each word.
  $\mathsf{ct}_i \leftarrow \mathsf{CKKS.Mult}(\mathsf{core}_i, \mathsf{Mask}_{(i \bmod d/2),\mathsf{iter}})$ ▷ For i from 0 to $d-1$
  $\mathsf{ct}_{\mathrm{comb},0}$ and $\mathsf{ct}_{\mathrm{comb},1}$ are empty CKKS ciphertexts
    ▷ For combining n/w ciphertexts into 2 ciphertexts
  for $j = 0$ to $n/(2w)-1$ **do**
    $\mathsf{ct}_{\mathrm{comb},0} = \mathsf{Add}(\mathsf{ct}_{\mathrm{comb},0}, \mathsf{ct}_j)$
    $\mathsf{ct}_{\mathrm{comb},1} = \mathsf{Add}(\mathsf{ct}_{\mathrm{comb},1}, \mathsf{ct}_{j+d/2})$
  end for
  $\mathsf{ct}_{\mathrm{comb}} = \mathsf{ct}_{\mathrm{comb},0} + X^{N/2} \cdot \mathsf{ct}_{\mathrm{comb},1}$ ▷ Combine real values into complex values
  $(\mathsf{ct}_{\mathsf{ID},0}, \mathsf{ct}_{\mathsf{ID},1}, \mathsf{ct}_{\mathsf{MSB},0}, \mathsf{ct}_{\mathsf{MSB},1}) \leftarrow \mathsf{D\text{-}CKKS.LUT}_{\mathsf{ID},\mathsf{MSB}}(\mathsf{ct}_{\mathrm{comb}})$ ▷ LUT size 2^w
  $\mathsf{ret}_i \leftarrow \mathsf{CKKS.Add}(\mathsf{ret}_i, \mathsf{ct}_{\mathsf{MSB},\lfloor i/(d/2) \rfloor} \odot \mathsf{Mask}_{(i \bmod d/2),\mathsf{iter}})$
    ▷ For all i from 0 to $d-1$
  for $\mathsf{nextIter} = \mathsf{iter}$ to $d-1$ **do**
    $\rho \leftarrow (\mathsf{nextIter} - \mathsf{iter}) \cdot w$
    if $-\rho \le \mathsf{A2B\text{-}Cutoff}$ **then**
      break ▷ Ensuring $O(n)$ rotations per ciphertext
    end if
    Prepare CKKS plaintexts $\mathsf{ScaleMask}_{j,\mathsf{iter}}$ $(0 \le j < d/2)$
      ▷ Slots have value $2^{-\rho}$ for the $[\mathsf{iter} \cdot w, (\mathsf{iter}+1) \cdot w)$-th bits for each word of the j-th pre-rotated ciphertexts
    for i from 0 to $d-1$ **do**
      $\mathsf{ct}_{i,\mathrm{scaled}} \leftarrow \mathsf{CKKS.Mult}(\mathsf{ct}_{\mathsf{ID},\lfloor i/(d/2) \rfloor}, \mathsf{ScaleMask}_{(i \bmod d/2),\mathsf{iter}})$
      $\mathsf{ct}_{i,\mathrm{scaled}} \leftarrow \mathsf{CKKS.Rotate}(\mathsf{ct}_{i,\mathrm{scaled}}, -\rho)$
      $\mathsf{core}_i \leftarrow \mathsf{CKKS.Sub}(\mathsf{core}_i, \mathsf{ct}_{i,\mathrm{scaled}})$
    end for
  end for
end for
for $j = 1$ to $d/2-1$ **do** ▷ Post-rotate
  $\mathsf{ret}_j \leftarrow \mathsf{Rotate}(\mathsf{ret}_j, j \cdot w)$
  $\mathsf{ret}_{j+d/2} \leftarrow \mathsf{Rotate}(\mathsf{ret}_{j+d/2}, j \cdot w)$
end for
return $\{\mathsf{ret}_{i,0}, \mathsf{ret}_{i,1}\}$ for all i from 0 to $d-1$